01 Dec 1985

Industrial Simulation with Animation

Edward T. Hammerand

Chung You Ho
Missouri University of Science and Technology

INDUSTRIAL SIMULATION WITH ANIMATION

Edward T. Hammerand* and C. Y. Ho

CSc-85-4

Department of Computer Science

University of Missouri-Rolla

Rolla, Missouri 65401 (314) 341-4491

*This report is substantially the M.S. thesis of the first author, completed December, 1985.

ABSTRACT

This thesis examines and evaluates the new simulation language PCModel. Prior to the arrival of PCModel, simulation via computer typically resulted in pages of statistics compiled over the duration of the simulation. PCModel's approach is to simulate the model on the display before the user in real time. Additionally, user interaction is supported to allow changes to be made throughout the simulation run.

The evaluation of PCModel is accomplished through inspection of a pair of examples already simulated in a conventional simulation language. The examples show the relative strong and weak points of the language, as well as demonstrating how PCModel is used.

ACKNOWLEDGEMENTS

The author would like to extend his sincere appreciation to his advisor, Dr. Peter C. Ho, for his guidance and support. Thanks are also extended to committee members Dr. Darrow F. Dawson and Dr. Arlan R. DeKock for their assistance in completing this text.

Special thanks are also extended to Clark L. Wilson and Miles Hufft who, as advanced mathematics instructors in high school, helped to decide which track my career would take.

Gratitude is expressed also to my family and friends who were supportive and understanding during the writing of this work.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

I. INTRODUCTION

Simulation has been posed as a problem in computer science for as long as computers have been available for use. Many approaches have been taken to the problem, some using conventional programming languages like FORTRAN and others using languages dedicated to simulation, like GPSS. It is the purpose of this section to examine how a language like PCModel was a natural step in the evolution of languages and how PCModel differs from what has gone before.

A. REVIEW OF THE LITERATURE

The purpose of a simulation typically falls into one of the three categories described by Mittra[1]: (1) description of a current system for prediction of behavior, (2) exploration of a hypothetical system, or (3) design of an improved system. To implement the simulation for whatever reason, a computer is generally employed; this for the same reasons computers are used elsewhere -- they can perform the necessary operations that would otherwise not be feasible. The first and third categories listed are concerned with simulations done by business and industry, while the second is more academic in nature. However, the purpose is basically the same for all three; a simulation is performed in order to determine what would be the result of a proposed system or change to an existing system without having to actually build or modify anything. Academic simulations result in theoretical information aimed at the problem being explored, while business simulation output influences the amount of time and money to be spent on alternative courses of action. For this reason alone the capability to predict the outcome of the given alternatives is both very useful and attractive.

The implementation of a simulation is developed using a programming language. The programming language, in turn, has been created with a field of use in mind. Those that are concerned with the simulation of systems are thus relatively restricted when compared with other languages of a more global nature, such as FORTRAN. Stephenson[2] points out that languages should be developed in as simple a fashion as possible, while at the same time incorporating the sophistication necessary for the solution of intricate problems by experienced programmers. Culik[3] further suggests that the type of problem to be solved should be defined first; then the features and divisions of the language can be created with specific types of use in mind. The combination of these two considerations should be employed in the development of a language, and this is indeed the case for PCModel. First, it has a relatively small instruction set, relying on combinations of instructions to accomplish tasks of a more sophisticated nature. As a programmer becomes more comfortable with it, he is able to assimilate in his repertoire sequences of codings to perform standard tasks. The simplicity of the instruction set allows such sequences to be altered to fit specific circumstances with relative ease.

Secondly, PCModel is intended to handle a specific class of problems. David A. White, the designer of PCModel, states that it is designed "to model the movement of manufactured assemblies through the assembly process."[4] This statement defines the nature of PCModel clearly enough so that one may decide up front if it is inappropriate for the problem at hand. The ability to determine whether or not a given language would be an effective tool for simulation is important. Meier[5] points out that the differences between simulation languages are wide enough to make the proper choice of a language a very important part of the simulation and evaluation process.

Even with the language most compatible with the problem selected, there are still a number of issues which can give rise to problems. Pritsker[6] examines in some detail the trouble caused by the sometimes slow rate of convergence to steady state conditions. It is oftentimes difficult to determine when a model has reached its own steady state behavior, and until this point is defined the simulation results cannot be evaluated precisely. Another problem lies waiting in the significance to be attributed to sampling error. A simulation needs to be examined for a range of initial parameters to obtain more meaningful results. However, the repeating of simulation runs for a variety of conditions is quite often not feasible simply for economic reasons, as noted by Tocher[7]. He suggests a number of techniques to compensate for the problem. Another problem must be considered during the analysis of the output of a simulation. Even if the simulation correctly models the given system, the interpretation of the data is still a rather tricky proposition; Graybeal[8] considers some characteristics of this problem.

Finally, one potential problem remains that can easily be overlooked, yet is vital to decisions based on the outcome of the simulation. It is the separation of distinct possibilities before the constructing of the model begins. Chorafas[9] discusses the problem and sites examples of how its presence can totally invalidate any simulation which incorporates it. Essentially, some decisions concerning the problem must be recognized as outside the scope of the problem; thus the simulation will examine alternatives of a more compatible nature.

Up to this point in its history, simulation has been done with either a conventional programming language or one designed especially for simulation. In most instances a simulation resulted in tables of statistics concerning various entitites in the model. However, little support was made for the capability of user interaction. Crosbie and Hay[10] and Licklider[11] both observe the need for interactive facilities in a simulation programming environment. Indeed, without interaction the user must wait until program end to determine the results of the initial conditions; interaction allows the status of the model to be monitored and adjusted as it runs.

The interaction capability of PCModel extends past just allowing user access to the ongoing generation of statistics; it goes hand in hand with PCModel's most outstanding feature:

its graphic display. The user can watch on the screen before him what is taking place in the simulation. Jones[12] in referring to the graphic display of functions relating to simulations remarked that the display was a strong persuader in the argument concerning the correctness of the simulation. It was clear on the screen that the results being observed were indeed along anticipated lines within the range of possibility. This is even more so the case with PCModel. As will be seen, the various entities of the model will be displayed on the screen as they move around the work area defined by the model. This is indeed a powerful means of displaying the behavior of a given system.

B. ASPECTS OF PCMODEL

PCModel should probably be thought of as more a simulation programming environment than a simulation language. Simulating a given model requires first the creation of the software to represent it to the system. The software in turn is composed of two parts: the simulation instructions and the overlay. The overlay is the map or floor plan over which the entities of the simulation will travel. It is created as a rectangular area of screen rows and columns, the individual positions of which are filled with whatever characters will make the overlay the most meaningful. The accompanying instructions create and control the entities, or objects, of the model. Objects are created by PCModel jobs; a job may create one object for a simulation or thousands. Control of objects is then managed by means of routes; the various instructions for moving objects about the screen and so forth are contained in routes. Subroutine-like constructs are also supported; they are PCModel links and contain the same type of instructions as routes. The running of a simulation under PCModel is a two-step process: first, the object program is loaded and converted to its run-time equivalent; the actual simulation then takes place in the run-time step. The instructions described above are concerned with the run-time model; there are also those that deal with the load-time process. All of these topics will be examined in more detail in the next section, which categorizes the different parts of the PCModel simulation environment.

II. THE PCMODEL ENVIRONMENT

The purpose of this section is to acquaint the reader with the various aspects of PCModel. This is accomplished through examination of not only the simulation language elements, but also the capabilities of the interactive running process. The explanations given here are not intended to explain in technical detail; that is left to the software documentation. Rather, this chapter logically groups the various aspects of PCModel for presentation purposes; the last section defines the logical process for solving problems using the features of PCModel. The following sections will examine characteristic simulation problems, noting specific programming details as they come up.

A. LOAD-TIME DIRECTIVES

The purpose of the PCModel directives is to define certain parameters and so forth at load-time that will be used during the loading process to create the run-time model. They are presented here in the order recommended by the designer of PCModel.

M (Maximum objects) defines the maximum number of objects that will ever be allowed in the model at any one time. This value is used to reserve storage for the MCB's, or Movement Control Blocks; each active object will have one MCB allocated to it, and that MCB will contain all of the information pertaining to it.

W (set maximum Work-in-process) is used to initialize the variable number of objects that will be allowed in the model at any given moment. This number may vary from zero to the number specified by M above. The F5 and F6 keys are used to decrement and increment, respectively, this value during the simulation run.

S (Symbols) gives the number of symbols and labels in the program. This value is used to reserve storage for them. Since the program will not be able to load if insufficient storage is allocated, an exaggerated estimate may be used here. If it turns out to be insufficient, a larger value must be used. It is always a good idea to specify an extra large value. The reason for this is that the symbol table is generated using both the space defined here and the space defined for the MCB's. If a symbol in the program is overlooked or misspelled, the loader will produce an unresolved symbol error. Generally, examining the value screeen will identify the offending variable. However, it may be that the symbol was processed inside the MCB storage and thus will not show up on the value screen. Increasing the S directive amount will make room for it in the symbol storage space.

X (X dimension of logical screen) and Y (Y dimension of logical screen) specify the number of columns and rows, respectively, to be used in the overlay. At least a single screen of 80 x 25 spaces must be used; the maximum product of the X and Y directive values is 32,767.

V (Viewing-window location) is used to position the display screen on the overlay during the loading process. This is very useful, as the loader plots the various routes on the display as it loads the model. V can be used to change the portion of the overlay on the screen during the loading process; as many V's may be used as are deemed necessary.

D (Description definition) is used to define a screen of text which describes the problem being modeled. Once the model is loaded, this screen may be examined by pressing the D key. A maximum of 25 lines is permitted and the text is terminated with a '$'.

O (Overlay definition) gives the location of the overlay to be used with the model. The overlay may be included inline with the program, or it may be kept in a disk file. If the overlay is kept inline, it is typically created with whatever editor the user has at his disposal. If the overlay is a disk file, it may be created, modified, and saved using PCModel's built in attribute editor. This editor allows for the creation of colored overlays and has a generous selection of functions to aid in the process. The overlay directive provides for an overlay to be kept in a disk file with the same name as the simulation program, one with a different but constant name, or an arbitrary file which the loader will prompt the user for.

The next part of a PCModel program consists of symbol definitions. These follow from the S directive explained earlier. All of the various symbols required throughout the program should be grouped together and defined here. Further, all of the variable symbols should be placed before any constant or pointer symbols; this will conserve the amount of memory required for the model due to the internal workings of PCModel.

J (Job description) directives define each of the jobs to be active in the model. For each job, J specifies its job number, the character by which it can be identified on the overlay, the route it will take, the number of objects to be released by the job, and so forth.

U (Utilization definition) singles out specific positions on the overlay for which utilization statistics are to be generated. Basically, this results in a percent-of-time-occupied figure for the position. U allows the positions to be labeled so that when the statistics are viewed during the simulation using the U key, it is easy to associate positions with their significance.

R (Relative xy reference) is the last of the load-time directives. It is used to specify a reference position on which subsequent relative moves (MU, MD, ML, MR, and RM) can be based. An R directive has no effect at run time, as is the case for all of the directives listed here, but it is useful in the debugging phase of programming. As the instructions are being loaded, the path a route will take is charted on the overlay. If logical jumps take place without absolute moves, there will not be anything for the loader to reference succeeding relative moves on. The R directive solves this problem, allowing the route path to be verified. R directives may be use throughout the links and routes as needed.

This concludes the set of load-time directives. As stated at the beginning of this section, they have been presented in the order that they should occur in the simulation program.

The remainder of the program consists of the links and routes for the model. Of the two, the links should precede the routes.

B. DATA TYPES AND SYMBOLS

This section will discuss the various symbols that may be used in a PCModel program. Three basic data types are supported for use in modeling systems: variable values, constant values, and clock values. Two different types of labels are used, one type for labeling specific instructions and the other for labeling links. Finally, there is a symbol type for identifying overlay positions. Each of these will be examined in turn.

Variables are defined, as are all of the symbol types excepting labels, by including them in the symbol section of the program along with an initial value. They are of the form @VARNAME. The value of a variable may range from 0 to 65,535. The number of variables is limited only by available storage. Additionally, each object that is active in the model has six variable parameters, which are referenced as OBJ@n. They have the same restrictions as declared variables.

Constants are values that are fixed thoughout the program. They are referred to by #CONSNAME. Constants should be used for values that are fixed for a simulation, as opposed to coding the number itself; in the event that the value must be changed, editing the source program will still be necessary, but only in a single place. Constants, like variables, are valid for values from 0 to 65,535.

Clock values allow for a range of operations dealing with the passage of time in the simulation. First the current simulated time may be accessed by a reference to the system variable CLOCK. Second, clock variables may be declared in the symbol section as %CLCKNAME; these variables may be operated on arithmetically to determine times for various events to occur, elapsed times, and so forth. Additionally, each object has 2 parameters which are clock variables, thus making it easy to associate times with specific objects. Each object also has associated with it a system parameter, OBJ%ST, which is set to the creation time of the object. This parameter may be used as a third variable clock parameter. Clock values may be compared for branching purposes.

As noted above, each object has six variable parameters, two variable clock parameters, and one system parameter containing the time of its creation. Each object has three other system parameters as well. The first is the job number of the object, given by JOB@ID. This value is specified in the J load-time directive. The second is the serial number of the object, JOB@SN. This number indicates what position the object has in the sequence of objects created by its job. JOB@ID and JOB@SN can be used together to uniquely identify any object in the model. The last system parameter is OBJ@ID, which contains the ASCII code for the character used to represent the object on the overlay. This parameter yields essentially the same information as JOB@ID; the same decisions can be made on knowledge of either the job number or the character a job is represented by, provided a one-to-one correspondence exists. Each of the three parameters may be used just like variable symbols, if desired. Changing OBJ@ID will alter the object on the overlay.

Label symbols are used for reference to both specific instructions and to any links that may be defined. Instructions are labeled as :INSLABEL; every PCModel program will utilize some instruction labels as they are used as destinations for the various branching instructions. When links are used, they are labeled as !LINKNAME. Links are transferred to with the LK instruction. They are PCModel's version of procedures or subroutines; once an object has moved from the beginning to the end of a link, it is returned to the instruction following the LK in the route.

The final type of symbol is that used to indicate positions on the overlay. They are constant in nature, which tends to limit the flexibility of PCModel in some instances. A symbol for an overlay position is of the form *POSNAME.

C. THE INSTRUCTION SET

Due to the unique graphical approach of PCModel, the nature of the instruction set is different from that of a conventional simulation language. The instruction set includes instructions solely involved with the logic of the simulation, some concerned only with the display for the simulation, and still others that touch on both aspects of the model. This section will examine briefly the various instructions in logical groupings. Details concerning their use will be emphasized in example models later.

The route and link delimiters (BR, ER, BL, EL) are used to used to signify the beginning and end of each route and link in the program; specifically, BR and ER are the begin and end instructions for routes, while BL and EL are used with links. BR specifies a route number, overlay position, and initial delay for any object using its route. BL must give the label for the link and may have an optional overlay position and delay. Each BR must have one and only one corresponding ER; a BL may have one or more EL's, but this is a poor programming practice (one entry with more than one associated exit).

The object movement instructions (MU, MD, ML, MR, RM, MA), together with the arithmetic instructions, typically make up the main portion of sequential program flow. The movement instructions are used to move the object's display character around the overlay, taking specified amounts of time for the movement. Of the six instructions under this grouping, five are relative in nature; that is, they cause the object character to move a fixed number of spaces in the implied direction relative to the current position. There is one instruction for each of the four directions of movement on the overlay: MU (Up), MD (Down), ML (Left), and MR (Right). Each of these instructions specifies a constant value for the number of spaces to move and a delay period after moving each space. For example, MU(3,6) would cause the object character to move up on the screen 3 rows, delaying 6 seconds on each row. The RM instruction specifies a signed (+/-) movement for both the x and y directions, as well as a delay period once the object has moved. The last of the instructions is MA, or Move Absolute. It moves the character immediately from wherever it is to the location specified and then waits for the indicated period.

One thing to note is that in the loading process the loader will trace on the overlay exactly the moves that have been indicated in the program. At load time, there is no way to determine the current location for referencing subsequent relative moves when sequential program flow is broken. Therefore the loader must be told what to take as the current screen position after any 'transfer-of-execution' type statements (IF's, JP's, etc.); this is accomplished with the R (Relative reference) directive discussed earlier. The path of the route can then be correctly traced on the overlay, which in turn aids in the debugging process.

The object delay instructions (ST, DN) halt the object in the traversal of its route. The ST (Set Time) instruction is the primary means of delaying an object. The value used for the delay can be a constant, come from a global variable, or come from a parameter of the object itself. ST is typcially used for simulating the time involved for the object to undergo some process. The DN, or Do Nothing, instruction is essentially equivalent to ST(1). Its function is to hold the object at the position of the DN in the route until the next second of clock time. The typical use of DN is to prevent entry into an infinite loop. As many PCModel instructions take no simulation clock time to execute, it is very easy to inadvertently code the program so that it is vulnerable to an infinite loop situation. This is particularly easy to do where only mathematical calculations are required in a loop; this will be examined further later.

The arithmetic operations (AO, IV, DV, SV) provide for PCModel's manipulation of numbers. The first of the four instructions grouped here is the AO, or Arithmetic Operation, instruction. It is the heart of the mathematical manipulations that can be performed, as it encompasses the defined operations of addition (+), subtraction (-), multiplication (*), and division (/). It takes three operands, the second of which is one of the symbols for the listed operations. The first and third operands are both sources for the operation, with the first also being utilized as the destination of the result; its previous value is destroyed. A copy must be made if the first operand will be needed later. Note that the order of the operands for the instruction places them in infix notation for the operation to take place. The IV (Increment Value) and DV (Decrement Value) instructions do exactly what they imply. They perform the indicated change on their single operands. IV(OPERAND) and DV(OPERAND) are logically equivalent to AO(OPERAND,+,1) and AO(OPERAND,-,1), respectively.

The last instruction of the group is SV (Save Value), which simply copies the value of its second operand to its first operand. It is useful for saving the first operand of an AO instruction. It can thus be used to make copies of values that are required for arithmetic in more than one place or to prevent changing of a value until a certain point or time in the simulation.

The random number instructions (RS, RV) can accomodate those models requiring generation of a random number to determine inter arrival times of objects, which way to branch, and other such aspects. PCModel has built into it a random number generator for this purpose. RS (Random Seed) allows the user to choose a seed value for the sequence, while RV (Random Value) allows successive elements of the sequence to be obtained. It should be noted here that the random number sequence must be initialized with a seed by RS before any attempts are made to get a random number through RV. This in turn typically requires that programs using random numbers have a high priority job whose route will encounter RS before any other objects get to an RV.

The next group of instructions (JP, LK) are concerned with unconditional transfers. The JP (JumP) instruction breaks the sequential flow of the object through its route and transfers it to the label it specifies as its operand. The LK (LinK) instruction also transfers the object, but in this case it transfers it out of its route to the link specified as its operand. The object will be returned to the instruction following LK when it encounters an EL in the link. Thus, a link in PCModel is similar to a subroutine call in a conventional programming language.

The two conditional jumps (JB, JC) transfer the object to the label specified as their last operand depending upon the result of the test performed. JB (Jump if path Blocked) has as its first operand an integer specifying a number of positions;

this is followed by the screen positions to check. If one or more of the given positions are blocked, the object will be transferred. JC (Jump if path Clear) uses the same format for its operands; however it only transfers the object if all of the positions are clear.

The conditional jump (IF) instruction is considered separately as it has a number of different forms. All IF instructions utilize the same format for the condition they test. The check made concerns one of six possible relationships between two operands: equal (EQ), not equal (NE), greater than (GT), less than (LT), greater than or equal (GE), and less than or equal (LE). The relationship itself is the second operand of the instruction, while the two numerical values take the first and third positions. The remaining portion of the instruction details the action to take place based on the result of the tested relation. Essentially, the instruction allows branching for either of the true and false results. Both the true and false results may specify a label to branch to. Additionally, a label may be replaced with the keyword NEXT to indicate that the next sequential instruction is the destination of the branch. Finally, the keyword WAIT can be used in place of a destination label; if the condition occurs that would otherwise branch to the label in WAIT's place, the object is held at the IF statement until the next clock cycle when the condition will be checked again. This is similar to the manner in which the TP instruction operates.

The position posting and clearing instructions (PO, CL, TP) are generally used in combination with one another to synchronize object movement. PO (POst) takes as an operand a screen location which will be marked as occupied, or posted, when the instruction is encountered by an object. CL (CLear) reverses the effect of a PO. It marks as unoccupied, or clears, its operand which is also a screen location. Care should be taken to ensure that the position to be posted or cleared is not already occupied by an object. If it is, posting the location will only hide the object that is currently there. When the object's time at that position elapses, it will move on and the position will become clear. Clearing a location which is currently occupied will result in the deletion of that object from the simulation. The third instruction of this group, TP (Test Position) can be used to test a number of positions to see if any one of them is currently occupied. If so, the object at the TP instruction will be held there until the next clock period, at which time the check will once again take place. Thus, TP provides the capability of checking for a clear path before proceeding; this can be used to prevent a PO or CL instruction from affecting an occupied position.

The wait conditionals (WC, WK) halt the object when it encounters them and holds it until a condition is met. WC (Wait Clock) may be used to halt an object in its route until the clock time specified by WC's operand is reached. WC might be used to synchronize objects in the simulation. WK (Wait Keyboard), on the other hand, stops the entire simulation when it is encountered by an object. Execution will begin again as soon as a key is struck. WK could be used to halt the simulation at points when the display should be saved.

The overlay instructions (PV, PM, VW, XZ) constitute the remainder of PCModel's instruction set. PV (Print Value) prints at a given location the current value of the specified variable. This is useful in the output of intermediate calculations, as well as statistics that may be saved with the screen image. PM (Print Message) prints at a given location like PV, but it prints a constant character string. This has use in displaying messages concerning the occurrence of specific events. VW (Viewing Window) changes the portion of the overlay that is displayed on the 80 x 25 character screen. For logical screens greater than the minimum, it can be used to focus on a portion of the overlay when the program logic dictates that something of interest will be happening there. The upper left position of the desired overlay section is specified to VW. This instruction may be used in multiple places throughout the simulation routes. Lastly, SA (Set Attribute colors of object) is used to set the foreground and background overlay character colors of the object encountering it. This can be used where objects encounter a multiple branch. If the number of branches are relatively limited, the foreground color of the object could be set to reflect the branch taken or decision made, thus providing some useful feedback. The background color could be set by another scheme as well.

D. RUNNING THE SIMULATION

Once the PCModel program for simulation of a given system has been created, it is ready to be loaded and run. It will be the purpose of this section to examine this process and all of the options that are available throughout.

With the source file saved on disk, the computer is ready to run PCModel. Entering "PCModel" starts the program. This will bring up the copyright information and the message "Press G to Continue." Doing so brings up the main menu, or help screen, which lists all of the options available under PCModel. Each of the possibilities is identified by a single character associated with its purpose, such as 'L' for Loading a source program. This is the first step to take.

Once L has been pressed, PCModel asks for the filename of the source program to be loaded. The filename is typed in here; the extension should be left off as PCModel requires and looks for an extension of .MDL. When the filename has been entered, PCModel begins reading it from the disk drive and creating the run-time program. As it does this, it traces on the screen the path each route defines while simultaneously displaying the instructions responsible for the route at the bottom of the screen. This is useful in debugging the program; the loader will probably show here if a route is not going to behave as expected. Additionally, any undefined symbols will cause a message stating such to be printed at the end of the program load. The symbols could be any of the six types discussed earlier (variables, instruction labels, etc.).

The V (Values) command can be used to identify undefined variables. V displays all of the symbols defined in the program. For each variable that is undefined at the end of the loading process, a 'U' will appear next to it on the screen. If the route appears to have a flaw in it or one or more symbols are undefined, it will be necessary to exit PCModel and return to the text editor to correct the souce program. To exit, use the Q (Quit) command. When the problem is corrected, repeat the loading process.

Once the program has loaded without evidence of any errors, the simulation can be run using the G (Go) command. This starts the simulation at time 0000:00:00. While running, the simulation can be temporarily halted at any time by pressing the space bar. There are a host of user interactions available once a running simulation is obtained.

Four types of information are available both for viewing and for alteration. The most common one is accessible using the V (Values) command described earlier. The values displayed are not limited to being helpful in the debugging phase; they may also be viewed when the simulation is temporarily halted and any variable values may be changed. This allows greater flexibility in placing loads upon the simulation. The E (Event) command displays the event screen, which contains information pertaining to each object currently in the simulation; this information includes the character for the object, creation time, current screen location, and release serial number. The parameter screen can be accessed with the P

(Parameter) command; the six variable and two clock parameters for each active object are listed. Finally, data concerning each job for the model is displayed when the J (Job) command is invoked. This consists of each job's size, overlay character, priority, associated routing, etc. Like the value screen, the screens for events, parameters, and jobs can all be edited for certain values.

One other screen is available for viewing. It is the utilization screen and is called up with the U (Utilization) command. On this screen are kept the hourly utilization statistics for the locations defined in the source program. It is possible to define a maximum of 21 of these locations; PCModel then calculates the percent of time these positions are occupied and tabulates the information on this screen. It should be noted that the column of statistics for the current hour of simulation is meaningless until the end of the hour. Also, the screen has room for only ten hours of simulation. At the start of the eleventh hour, the column for the first hour will be used again, and so on. Invoking the U command causes the model to be temporarily halted while the utilization screen is displayed; using the G command will change the screen back to the overlay and continue the simulation. This is also the case for the V, P, E, and J commands. One notable difference between U and the previous four commands is that the screen has no alterable values on it.

There are two procedures that can be used with the utilization screen. The first is the O (Output) command. When invoked, it prompts the user for a filename under which to save the utilization data after each ten hour period; successive screens are appended to one another. The file thus created can be handed in turn to a program in a conventional programming language for further calculations or it may simply be used for inspection by the user. In either case, this option allows for long simulation periods without loss of information. The second procedure is defined through use of the function key 7. This key acts as a toggle for halting the simulation after every ten hours; when active this allows the user to inspect ten complete hours of utilization statistics and print them if desired before resuming simulation. It would also be useful if certain variables might require changing periodically. The status of the F7 toggle, H for Halt mode and G for Go mode, is displayed at the bottom of the overlay screen.

Another PCModel feature is the capability to save the entire status of the simulation to disk and then bring it back to start at the point where it was stopped. Also, some simulations may take some time to get to steady state. Once this steady state is reached, the simulation could be saved to disk using the S (Save) command, which prompts the user for a filename. Then it could be used to illustrate the system to others without the annoyance of having to wait while the initial simulation period takes place. The simulation environment is brought up from the disk using the R (Restore) command, which prompts for the saved filename. Further, R could be used on the same file more than once. The file extension for a saved simulation environment is required by PCModel to be .SIM.

As the simulation may be viewed by other than its author, the D (Description) command can be helpful. When invoked, it brings up the screen which consists of the input to the D directive in the source program. Useful comments here can make interpretation of the simulation more meaningful.

The last PCModel command that is directly involved with the running of the simulation is the I (Initialize) command. Utilizing I causes the simulation to be set back to the same position it was in when it was first loaded; thus the entire run can be started all over, perhaps with some different variable values, without having to reload the entire program. The only aspect of the model environment that I does not affect is that of the variable values which were defined in the symbol section of the program. These variables had their storage allocated at load time and they were initialized then as well; thus only reloading the program will initialize them again. To avoid this, the variables can be initialized in the run-time model by some dummy job whose priority allows it to affect these variables before any use is required of them.

At any time the simulation is running, the currently visible portion of the overlay may be sent to the printer. This is accomplished using the Shift-PrtSc key combination. Alternatively, the overlay image may be saved to a disk file using the F (File) command. The first time F is invoked, it will prompt for a filename; the extension is required to be

.SCR. Subsequent uses of F will concatenate the current screen to the .SCR file. The portion of the overlay that is currently being displayed can be changed in two ways. First, there is the PCModel instruction VW which changes the upper left corner of the viewing window to the specified position when executed. Secondly, the user may interactively move the screen over the overlay using the four cursor keys. To do this, the scroll lock key must be toggled so that it is the overlay which appears to move when the cursor keys are used. The other use of the scroll lock toggle will be discussed next.

A common happenstance in a simulated environment is that of tool failures and periodic maintenance. While this can be coded in the software, PCModel also provides an interactive method for simulating failures. As mentioned above, in one setting of the scroll lock toggle, the cursor keys move the display around the overlay. For the other, the cursor keys move PCModel's blocking character. Moving it to an arbitrary position and depositing a block there is equivalent to moving some job object to that position or posting the position in the software. Once a position has a block deposited on it, which is done by pressing the F8 key while the blocking character is over it, that position remains occupied until the block is removed, which is accomplished by moving the blocking character to the position and pressing F8 a second time. A maximum of 50 blocks may be deposited on the overlay at any one time.

Four of the function keys have functions relating to the time advancement for a model. Keys F1 and F2 decrement and increment, respectively, the pace at which the model proceeds. Using them, the model can be slowed from its original setting of the system's top pace to a pace which closely approximates real time clock advancement. The current factor for the pace, which ranges from 0 to 250, is displayed at the bottom of the overlay screen. F1 and F2 affect the pace for settings of 0 to 10 by intervals of 1 and then from 10 to 250 by intervals of 10. The pace can be slowed during crucial intervals and increased for those that are not of concern. F3 toggles the advancement mode for the simulation clock. In increment mode (indicated by I at the bottom of the overlay) the clock advances one second at a time; in look-ahead mode (indicated by L) the clock advances to the time of the next projected route movement of an object. Look-ahead mode will simulate a model more quickly, but PCModel's author states that it may not be as accurate as the increment mode for some simulation environments. Finally, F4 can be used to put the clock advancement into a single-step mode (indicated by an S next to the clock on the overlay). Pressing F4 again puts the model back in the regular go mode (indicated by a G). When in single-step mode, the clock advances 1 second for each press of the G key. Single-stepping makes possible close examination of events that would otherwise be difficult to follow.

Two other function keys have functions defined under PCModel. They are the F5 and F6 keys for respectively decrementing and incrementing the maximum number of objects allowed in the system at any given time. This is the Works-in-Process figure mentioned earlier in connection with the W load-time directive. W initializes the number and then F5 and F6 can be used to alter it. The current value for Maximum Works In Process (MWIP) is displayed at the bottom of the overlay screen, along with the other model status values examined thus far. The other two statistics displayed are the current number of objects in the model, or Works In Process (WIP) and the number of objects that have completed their routes and exited the model, or the Work Complete Count (WCC).

When it is desired to terminate PCModel, the Q (Quit) command is invoked. This command was mentioned earlier in connection with the loading process. Before quitting, any saving of screens or the current simulation environment must be completed because the Q command completely exits the PCModel program, returning the computer to the operating system.

For the sake of completeness, it should be noted that there are four other commands available under PCModel, although they have nothing to do with the actual running of a simulation program. The first of them is the A (Attribute editor) command, which invokes the overlay editor built in to PCModel. This editor is specially designed for creating colored overlay screens. The other three commands are all concerned with the display being used for the simulation session. They are B (Black and white), C (Color), and M (Monochrome).

E. SOLVING SIMULATION PROBLEMS USING PCMODEL

This section is devoted to the task of outlining the steps required for the development of the solution, or model, of a given simulation problem. This development will be broken down into five steps, each explained in the general terms of the PCModel programming environment. This development process will be used to develop two different kinds of simulation models in the following sections.

1. Define and Limit the Problem When a problem is presented for simulation, it inherently has a set of characteristics particular to it. It is this set of characteristics that must be precisely outlined during this step, not only of the given problem, but those of the desired solution.

In terms of PCModel, the types of things to identify are those that will be associated with jobs and those that will be incorporated into routes. For example, consider a simulation model to be set up for an automobile manufacturing plant. The model is to consist of an automibile assembly line and the stations at which they stop for assembly. Assume that all of the specific data, such as processing rates and required assembly line speed are available for use. Modeling of the assembly line still cannot begin until a thorough definition of the solution requirements has been stated. Those items of interest in the solution must be designated so that the model can incorporate generation of statistics for them. An example of this can be seen in the simulation output required for a particular tool on the assembly line. If the output for the tool is to consist of its utilization time, it may be easiest to incorporate the tool in the model as a job, rather than as part of a route. Simulation of failures may be simpler if the tool is a job, given the blocking capability of PCModel. If, on the other hand, it is only necessary to halt the automobile on the assembly line for some variable work time, the tool characteristics would probably just be incorporated into the route for automobiles. All of these types of decisions and their repercussions will be pointed out in the succeeding chapters dealing with specific examples.

Secondly, some aspects of the system must be regarded as outside the scope of the simulation. The problem must be limited to one which can be reasonably modeled. In order to keep the complexity of the software from becoming so great as to confuse the model behavior with programming tricks, some boundary must be drawn as the environment for the system. Consider the model of an automated warehouse. It must be decided where to begin modeling the behavior of the goods stored in the warehouse and which operations of the warehouse are to be included. Assume the goods arrive by truck at the warehouse door and are then processed through the receiving area. Also, goods must be processed by the shipping department before leaving the warehouse. It may be that including the behavior of the receiving and shipping departments would overly complicate the model; it might even be decided to simulate the shipping and receiving departments separately to determine their effectiveness. (The example of the automated warehouse will be examined in detail in the next section.)

Not only must the problem be limited, but in some cases it may be that PCModel is determined to be inappropriate for the given circumstances. As is the case in all simulation languages, PCModel inherently has some weaknesses which make it a poor choice to simulate some types of environments. For example, consider a model for a supermarket wherein customers enter the store and stay for various lengths of time before being checked out. GPSS, a standard simulation language, would typically make use of an ADVANCE block to delay for the time a customer spends in the store. The ADVANCE block creates neither a first-in first-out (FIFO) ordering of customers, nor is it a single customer facility. In order to simulate such a system with PCModel, its natural FIFO ordering of objects must be overcome. In fact, depending on the problem, it may be that PCModel is not the system to use. (The problem of the supermarket will be examined later.)

<u>2. Collect Data for the Problem</u> Once the problem and output requirements have been defined, the development of the solution moves to the second step, that of collecting data. This data will be in the form of specific numerical values for the various pieces of the simulation environment. Continuing with the example of the automobile assembly line, some representative examples would include the processing times of the various assembly stations on the line, the number of each type of tool at each of the assembly stations, the on-line time

for any given tool (or conversely its down time due to periodic malfunction and repair), and so on.

In addition to determining what the values for specific parts of the simulation are, it is also necessary to determine which values are fixed and which are variable for the simulation. If the simulation is to determine the results of possible changes in an already existing system, some numerical values will be constant due to the system's nature. For the assembly line, a particular painting apparatus that is going to be part of the new set up whatever the result of the simulation studies (due to economic factors, for example) must have a constant time to function. This value cannot change and may be entered into the simulation data as a constant.

In contrast to this are the variable values to be considered. These are quantities, rates, delay times, etc. which the simulation will be used to find optimum values for. On the assembly line, it would be possible to consider different numbers of assembly workers for upholstery fitting. Each of the workers will perform his task at a more or less constant rate, but the number of workers performing the same function will greatly affect the performance of the system. If there are too few workers, the entire assembly line has what amounts to a bottle neck; on the other hand, if there are too many workers, some of them remain idle and the assembly line may not prove to be economically sound. Besides quantities, there are also tools which can be set to run at variable rates and assembly times that may be varied. Essentially, no part of the assembly line can run so slow as to interfere with the performance of the other parts, yet neither can any part run so fast as to be economically unfeasible.

One other important part of the data collection step is to decide upon or obtain the specific measurements for the overlay, or "floor plan", of the simulation. Such information is needed by any simulation language, but it becomes especially important when the graphics nature of PCModel's output is considered. In order for the model solution to be realistic, it must obviously appear to be as close to the actual physical operating environment as possible. Fine points in the production floor layout that would either be disguised or overlooked completely by a conventional simulation language suddenly take on new relevance under PCModel.

3. Develop a Software Solution With the specifics of a system defined, the creation of the working simulation turns to the task of choosing and developing a method of solution. This entails the process of putting together the pieces of PCModel required to effectively simulate the model, once it has been thoroughly defined and the numerical information concerning it has been established.

Before the coding of the PCModel program can be attempted, each individual piece of the model should be assigned to a job and/or route. This furthers what was put in motion in step 1, with the emphasis shifted away from defining the problem requirements to the correlations to be made between the requirements and PCModel's specific configuration. With the pieces for each job and route known, coding can begin to take place. Now the data collected in Step 2 is applied to form the configuration for each of the routes required by the model.

For a given route, all of the information pertaining to it is reflected by PCModel in two ways. First, the path the object is to follow on the screen is mapped onto the overlay and plots, at the scale arrived at for the display, the foot by foot progression of the route's objects through the simulation. Second, everything that is to happen to an object during its course through the simulation is built into the route for that object.

The overlay path and the coded route are created in tandem with one another. The instructions must mimic the behavior of the object, as must its path on the overlay. At each place in the PCModel code where some event takes place other than a typical move, the overlay should reflect the nature of this event. Key processing points, cross-overs with other routes, points of object origin and exit, etc. should all be clearly labeled. Conversely, the overlay can be used to develop the code, making sure that each of the routes runs logically along the assembly floor. No interference should be caused by the route in question, nor should it receive any from other routes; that is, where routes logically overlap the software should anticipate and handle possible collisions beyond PCModel's built in collision-prevention mechanism. Also, the overlay will help to determine if routes are intersecting where they logically should not. The following sections examine the entire software development process for the two problems mentioned previously. Specific examples of programming details will be left until then to be explained.

4. Solve the Model With the system configuration and data embedded in a PCModel program and overlay, the simulation can be run to determine the effectiveness of the set up. It is in this step that the interactive nature of PCModel comes into play. By its unique method of operation, PCModel runs the simulation on the display of the computer in real time, at a rate which can be increased or decreased as desired throughout the simulation. The system can be adjusted to run at a rate approximating that of the real operation to provide close inspection of its operation. Solving the model entails running the program and interacting with it in whatever manner necessary to explore the variable aspects of the system. For example, at various points it may be of interest to temporarily halt the simulation and change the values of certain parameters or variables; this allows the system to be subjected to any loads or bottle necks that are deemed feasible. Simulation of tool failures, or route blockages, can also be created and removed with the used of the blocking character.

Besides user interaction taking place to examine the system, PCModel handles some generation of information by itself. Utilization data can be assimilated for various positions on the screen by defining these positions in the software. (This essentially amounts to the percentage of the time the position is occupied, a statistic similar to the utilization figures for a facility in GPSS.) The data screens can be saved to disk for later examination or for use by a program written in a conventional language such as BASIC or PASCAL. It might also be feasible to make use of PCModel's capability of checking the system clock and halting for user input to let the user known when to save overlay screens.

5. Evaluate the Solution Once the program has been run to its logical conclusion, it is time to evaluate what has been produced. First, it must be determined whether or not the model behaved in a realistic manner. If not, the model must be adjusted until it is deemed realistic by those familiar with the real world counterpart.

When it is decided that the simulation output can be trusted, it is time to examine what that output means. If the system modeled is already in existence, the simulation will hopefully have shown what courses of action can be taken to increase the performance and efficiency of the system. On the other hand, if the simulation was being done to determine the workings of a new or anticipated system, the output should be a strong indicator as to how many workers to hire and how to arrange them efficiently, for example.

Lastly, the simulation model can be continuously upgraded and expanded upon to examine other areas under scrutiny in the given system. Small changes can continue to be made to the software and overlay until PCModel creates an efficent and accurate graphical predictor of the system of interest.

III. THE AUTOMATED WAREHOUSE PROBLEM

In the text Simulation with GPSS and GPSSV by Bobillier, Kahan, and Probst[13], an automated warehouse is simulated using GPSS. An automated warehouse is one under computer control that handles all of the input and output assignments; thus the operation runs very efficiently. GPSS's output for the simulation is in the form of facility utilizations, queue statistics, and other numerical data determined upon the end of the simulation. In this section the same problem is examined; PCModel is used to solve it and the solution is compared with that of GPSS. In addition to showing the relative strengths and weaknesses of PCModel when compared to a standard simulation language, much insight into the workings of PCModel can be gained from such an example.

A. DEFINE AND LIMIT THE PROBLEM

As was indicated in the previous section, the first thing to be done is to determine the limitations to be placed on the problem. The same warehouse characteristics will be used for this example as are specified for the GPSS model. The problem as defined for the GPSS solution is to "simulate the operation of the warehouse to check if the whole system can operate satisfactorily, especially during peak hours."

The warehouse consists of corridors, each with its own automated crane. On each side of each corridor is a rack, so that each crane has access to two racks. Each rack is further divided into bins; each bin is capable of storing one pallet. Pallets, in turn, are defined to be the smallest unit of the warehouse and the only unit to be physically handled by the system.

Pallets of goods arrive at a receiving port and are placed on the lower level of a circular conveyor when an open position arrives. The conveyor is actually a twin pair of conveyors, one above the other, each moving opposite the other at the same speed. They are connected at their ends by a twin pair of lifts, running at the same speed so as not to disrupt the continuous path of the pallets. As a pallet travels across the upper level of the conveyor, it is transferred from the conveyor to the input buffer of its corridor by computer control, if space permits. The GPSS model serially assigns corridor numbers to pallets; for the PCModel simulation, this assignment will be done randomly, to exhibit this feature of the language. If the input buffer should be full, the pallet will go around the entire conveyor again.

Once a pallet enters the input buffer for its corridor, it waits for the crane to finish with any previous jobs and get to it for placement in the corridor. The corridor will be broken up into zones for placement of the pallet; this division will be according to the distribution of different types of pallets for that corridor.

Shipping requests will be handled in a similar manner. The requests will be considered to arrive at a central location and forwarded to the computer control of the crane of the corridor designated for the pallet to be shipped. As was done for the incoming pallets, the requests will be assigned randomly to the different corridors. The requests will wait in a buffer while the crane finishes prior jobs. When the crane is free, it will move to the position of the pallet to be shipped (determined again by the distribution of the pallets) and move the pallet from its stored location to the output buffer area to wait for a position on the main conveyor. When an open position arrives, the pallet is placed on the conveyor by computer control and moves to the lower level of the conveyor where it exits the system to be loaded onto a truck or freight car.

Thus, the problem is defined in general terms although no actual, specific data has been gathered concerning the system to be developed. The problem is implicitly limited to the areas of interest discussed in this section.

B. COLLECT DATA

In this phase of model development, numerical data concerning the particular problem to be simulated is gathered. The physical dimensions of the GPSS model will be used in the overlay screen of the PCModel simulation, as well as in defining the moves to be taken in the routes for the various jobs. Other constants, such as conveyor speeds, arrival rates, crane speeds, buffer sizes, etc. will be assimilated in order to be prepared for Step 3, that of putting together the software to solve the problem.

Data collection starts here with information pertaining to the smallest entity in the simulation, the pallet. Each pallet will be a square, 1 meter (m) on a side. Thus, each bin need be only this size. A rack will contain 10 bins vertically and 50 horizontally; it then measures 10 x 1m, or 10m, vertically and 50 x 1m, or 50m, horizontally. Each rack in turn will therefore contain 50 x 10, or 500, individual bins; with each corridor having a rack on either side, every crane will have access to the corridor's total of 1,000 bins. The simulation will model 10 such corridors, with a 1.5m width separating the racks for the respective cranes. For 2 racks of 1m width each and the 1.5m width of the crane space, the width of each corridor totals 3.5m; the total width of the warehouse of 10 corridors comes to 35m. The length of the warehouse will be 50m, for 50 horizontal bins at 1m length apiece.

Each corridor will initially be considered to consist of 4 zones. The quantity of both received pallets and shipping requests will breakdown as 40%, 30%, 20%, and 10% for each of the four zones A, B, C, and D, respectively. The GPSS model breaks the four zones into equal sizes; however the PCModel simulation will be built with slightly altered percentages. This comes from a decision to keep the logic of the program at a moderate level, as PCModel has only integer arithmetic. As will be seen in the calculations for crane movement, it is much simpler if each zone starts on a whole boundary. This cannot be resolved with a horizontal distance of 50m being split into 4 equal zones. The solution is to have the first zone be slightly larger than a fourth of the corridor at 13/50, the second slightly smaller at 12/50, and the third and fourth zones at 13/50 and 12/50. The distances for crane movement will be slightly different from those determined in the GPSS model, but the programming logic will remain manageable, as will be seen in the section on coding.

Each corridor will have an input buffer which will receive pallets from the conveyor and an output buffer where pallets will wait for an open space on the conveyor. The input buffer will hold 4 pallets, and the output buffer will have room for 2 pallets. The servicing of pallets in both buffers will follow the obvious first-in first-out ordering. The received pallets enter the input buffer and wait there for the crane to place them in the corridor; the pallets pulled from the corridor to be shipped are placed in the output buffer. If the input buffer is too small, some received pallets may have to travel around the conveyor more than once; if the output buffer is too small, the crane will be unable to get any more pallets requested for shipping because there is no place to put them. Thus, the values for the input and output buffer sizes are crucial to the effectiveness of the system. For both buffers of a corridor, an automatic mechanism independent of the crane is assumed to exist to take pallets to or from the conveyor.

The conveyor is defined to be 35m in length on both the upper and lower levels. The lifts between the two levels will be 3m tall. (A height of 2m was quoted as the distance for the GPSS model, but this conflicts with the conveyor capacity stated.) Thus, the length of the entire conveyor circuit will be (2 x 35m) + (2 x 3m), or 76m. Pallets will be placed 1 m apart on the conveyor, so it will support a maximum of 76m x (1 pallet/2m), or 38 pallets. The conveyor will move at a constant speed of 20 meters per minute.

The cranes will operate at 10m/min in the horizontal direction and 1 meter/min vertically. The behavior of the crane will be as follows: if a pallet has been placed in the input buffer, the crane gets it and places it in its bin. Next, the crane checks to see if a request to ship a pallet has been received. If so, the crane moves to the calculated position, if there is a pallet in storage, gets the pallet, and moves it to the output buffer. The crane then repeats the cycle by checking for a received pallet. (The determination of the position of placement for a received pallet or storage location of a pallet to be shipped will be considered in detail in the actual PCModel software coding of the calculation.)

The remaining physical part of the warehouse operation is the placement of the receiving and shipping ports. They are arbitrarily placed on the lower level of the conveyor. The receiving port is the place where pallets enter the simulation and wait to be placed on the conveyor. The shipping port is the place where the pallets exit from the simulation by leaving the conveyor. Their behavior is not part of the simulation itself; they are included for completeness of the problem. (This is actually part of the implicit limiting of the problem indicated in step 1.) From the left end of the lower level of the conveyor to the shipping port is 15m; the port itself is 1m wide; it is then 3m to the receiving port which is also 1m wide; the remaining distance is another 15m to the right end of the lower level of the conveyor to yield the conveyor's total length of 35m.

With the physical dimensions of the warehouse in hand, the scale to be used for the overlay can be obtained. As calculated, the warehouse is 35m wide; the divisions in that 35m are 1m for pallets, bins, etc. and 1.5m for the crane spaces. The greatest common fraction is 0.5m, so it would be convenient to use this in the model. At 0.5m per character space, 35m would require a minimum overlay width of 70 characters which is less than the required display screen width of 80 columns; in fact, an additional 10 columns, or 5 to either side of the warehouse, will be available at this scale. If 1m per character space were used (thus distorting the crane spaces), then the warehouse would only take up 35 of the required 80 columns. This arrangement would not be very attractive, so 0.5m per character column seems to be optimal.

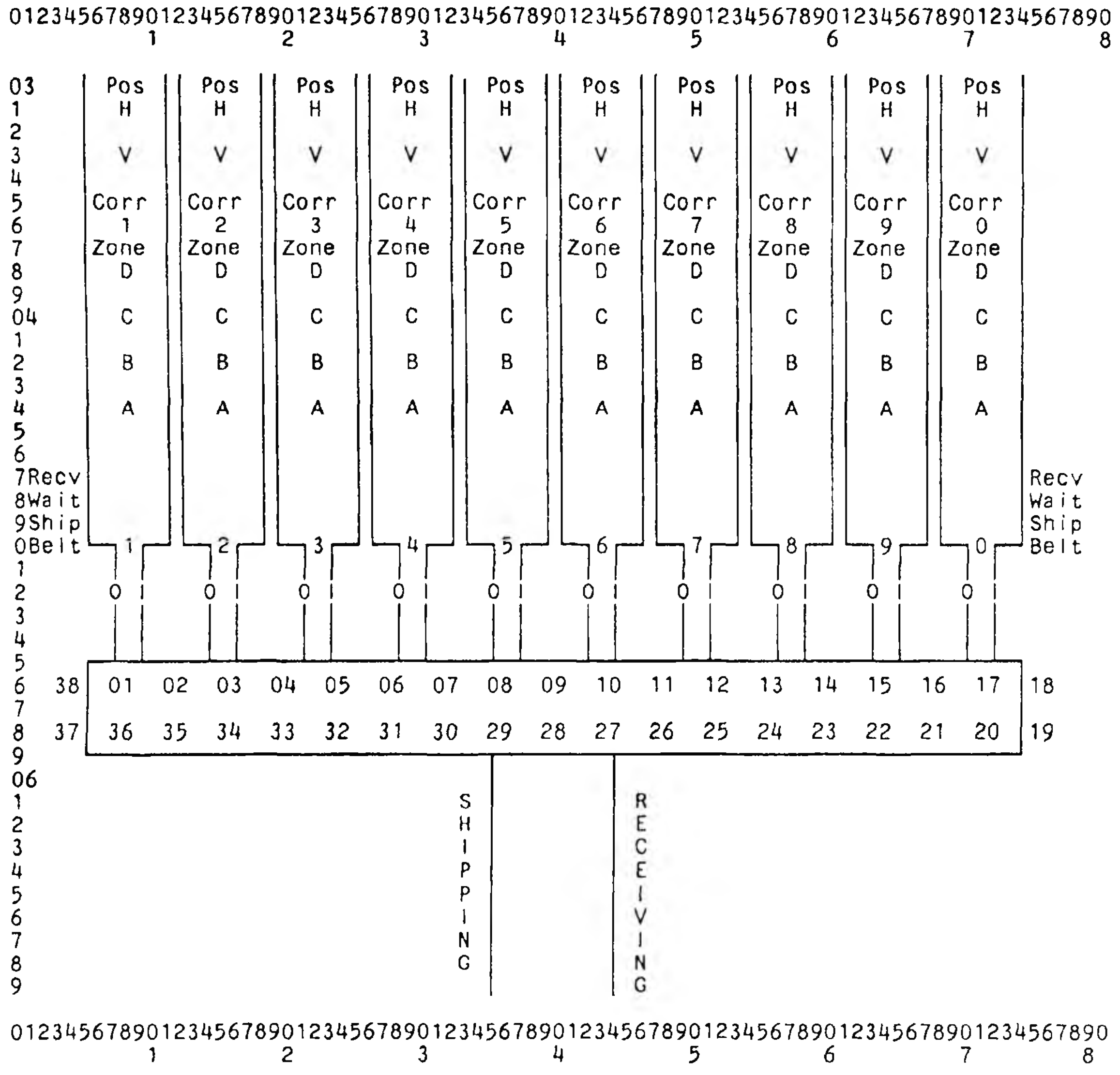
Determination of the scale of distance per display row proceeds in a similar fashion. The physical distance required would include the length of the warehouse at 50m, the length of the buffers at 4m, the distance to display the conveyor on the screen at 5m (1m for both the upper and lower levels and 3m for the lifts), and the length of the shipping/receiving port at 10m (arbitrarily chosen). The total comes to 69m. As 1m is used as an increment, it should be used as the scale per row. This would mean allocating a minimum of 69 rows for the overlay. This is well over the single screen figure of 25 rows. Thus even halving this scale would still not place it all on one screen. On the other hand, using 69 rows will allow the receiving/shipping ports, the conveyor, the input and output buffers, and the first few meters of the corridors to be displayed on a single screen. Thus 1m/row is an acceptable scale. The warehouse overlay can be seen in Figure 1.

Finally, the last numerical items required for the problem data collection are the arrival rates of received pallets and the requests for pallets to be shipped. The GPSS model is designed to allow for variable arrival rates, and thus the PCModel simulation will be set up in this manner as well. One capability of GPSS utilized in its model that PCModel does not have is that of generation of exponentially distributed variables. The arrival rates themselves are stated to be Poisson distributed. For the PCModel simulation, a constant mean interarrival time is used for the Poisson variable. The two sets of arrival rates to be simulated are (1) receiving at 2 pallets/min and shipping at 1 pallet/min and (2) receiving at 1 pallet/min and shipping at 3 pallets/min.

At this point in the development, the problem has been defined as to what is to be simulated. Further, the system specifics in the form of numerical data have been collected for the model to be constructed. The next step in the development process is to construct the PCModel software necessary to accurately reflect the problem as given.

C. SOLVING THE PROBLEM IN SOFTWARE

1. Job and Route Definitions To begin the writing of the software, the association of system pieces with jobs and



012345678901234567890123456789012345678901234567890123456789012345678901234567890
Pos H V Corr 1 Zone D C B A
Pos H V Corr 2 Zone D C B A
Pos H V Corr 3 Zone D C B A
Pos H V Corr 4 Zone D C B A
Pos H V Corr 5 Zone D C B A
Pos H V Corr 6 Zone D C B A
Pos H V Corr 7 Zone D C B A
Pos H V Corr 8 Zone D C B A
Pos H V Corr 9 Zone D C B A
Pos H V Corr 0 Zone D C B A
Recv
Wait
Ship
Belt
1 2 3 4 5 6 7 8 9 0
38 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18
37 36 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21 20 19
SHIPPING
RECEIVING

Figure 1. Warehouse Overlay

routes must be finalized. For the automated warehouse, the job associations will be stated and all other information will be incorporated into the routes taken by the jobs. First, the received pallets and requests for items to be shipped are designated as jobs, as they are the primary focus of the simulation. (An object of the received pallet job will typically be referred to as "R" throughout the software development, while an object of the shipping request job will be referred to as "S".) Second, each of the cranes will be modeled as a separate job, consisting of one object which loops through the system checking for the presence of R's and S's waiting to be processed in the corridor. Lastly, initialization of the simulation will be handled by one separate job, whose single object perfroms the required initializations along its route and then exits the model before any other objects enter. (This initialization at run time of the model, rather than only at load time, is a standard procedure when using PCModel; it allows the use of the initialize option of PCModel's interactive facility to restart the simulation without reloading it, as explained earlier.) In addition to the three types of jobs indicated, a fourth will be deemed necessary once programming begins, for synchronization of pallets on the conveyor belt.

With the jobs thus defined, the coding effort can be concentrated. One might be inclined to assume that the initialization job and its route would be the logical choice to define first. However, as it is the initialization job's purpose to set up the system for the others to follow, it actually makes more sense to leave it until the last. As noted previously, the R's and S's are the primary focus of the system, so they will be considered first. The discussion of the R job precedes that of the S job as a matter of human factors; it is normal to think of a received pallet as the first thing to enter the warehouse when it opens for business.

2. <u>Received Goods</u> The R's model the received pallets. They enter the system at the receiving port and wait for a spot on the conveyor. When an opening arrives, R is placed on the conveyor by implicit computer control and proceeds to its designated corridor. If a position in the corridor's input buffer is open, R moves into it and waits for the crane to process it; if not, it makes one full circuit of the conveyor and checks again. This process is repeated as many times as are necessary; this behavior will lead to the complete utilization of the conveyor capacity when the system becomes overloaded due to high arrival rates, low crane speeds, etc. Once R is accepted in the input buffer for its corridor, it moves up the input buffer as space permits.

When the crane gets to R, logically R would be removed from the input buffer and moved to the bin in the corridor where it would stay until a shipping request removed it. Practical considerations preclude this, however. For the capacity of the warehouse, there would have to be 10,000 character spaces allocated to this purpose. Although 10,000 characters is a considerable requirement, it is well within the 32,767 character overlay limit. The inclusion of this much space would make the overlay extremely unwieldy and certainly distort the physical dimensions of the warehouse. Further, PCModel does not allow for the direct movement of objects based on how many are already in storage. Even when this is overcome by coding tests to determine how much constant movement is required, the problem becomes one of having to code the tests and moves individually for each of the corridors. This is due to PCModel's limitation to constant reference locations. When the jobs and routes for the crane objects are discussed, this same problem arises and its resolution will be made there. For purposes of the R job, it is simpler to let the R's exit the system once the crane is known to be ready to take the R from the input buffer.

As can be seen in Figure 1, the R's will arrive at XY(44,69). The time between arrivals as discussed in the first section is taken to be a constant and will be stored in the variable @RECVRATE. (Using a variable instead of a constant allows the arrival rate to be altered using PCModel's value screen while the simulation is running.) Also, if the initialization and synchronization jobs are assumed to be numbered 1 and 2 respectively, then the R job is 3. Thus, to begin the R route,

```
BR(3,XY(44,69),@RECVRATE)
```

is coded. R moves up the receiving conveyor at an arbitrary speed of 6 sec/m to the point where it meets the main conveyor.

MU(9,6)

At this point, R waits for an open position to come by on the conveyor, which is chosen to travel past the waiting R's in a left-to-right fashion. The column-wise scale of the screen is 0.5m per column. This means that any given R will appear to only take up 0.5m left-to-right; it must be assumed that the R is actually occupying two horizontal character positions. Thus, the R's on the belt when it is full will occupy every fourth position (as the belt requirements state that 1m, or 2 columns, must separate pallets). Further, the conveyor is moving at a speed of 20 meters per minute, which is equivalent to 1/3 m/s. The given conveyor rate of 1/3 m/s is equivalent to 0.5m per 1.5 seconds. To keep the arithmetic reasonable, the conveyor speed needs to be in terms of 0.5m per n seconds, where n is an integer. This allows conformity with the PCModel restriction of integer operands. Cutting the given rate of 1/3 m/s in half yields 1/6 m/s, or 0.5m per 3 seconds. It is a simple matter to cut all other speeds in half, so that no part of the system is altered. The times indicated in the PCModel instructions will effectively be half-seconds, when the comparison with GPSS is made.

With the conveyor speed of 1/6 m/s, it takes 6 seconds to move 1m and thus 12 seconds to move 2m; therefore, a possible open spot on the conveyor will occur every 12 seconds. What is needed now is some mechanism to indicate, perhaps by means of a global variable, when a valid time to check the conveyor for an open spot occurs. This is accomplished by the synchronization job mentioned earlier. The sole task of this job will be to naintain a global variable, @GO, such that it increments on the closed interval (0,11) at the rate of 1 per second. Thus when @GO is zero, an R can "go" onto the conveyor (provided that the spot is not already taken, as will be checked next). Note the use of the DN (Do Nothing) instruction to prevent the occurrence of an infinite loop.

```
:WAIT     DN
          IF(@GO,EQ,0,:READY)
          JP(:WAIT)
```

Once it is known that a valid position has arrived, a check must be made to ensure that it is empty before an R attempts to move onto the space. Also, it must be remembered that an R is assumed to be occupying two horizontal character spaces, so both must be checked.

```
:READY    JB(2,XY(44,59),XY(45,58),:WAIT)
```

If R makes it past the JB statement without transferring, it is because it an open space was found. The R is assigned a corridor number before being placed on the conveyor because of the nature of the data structures available in PCModel. Since the only data type as of this writing is the scalar variable, all of the information pertaining to a specific corridor will have to be kept in global scalar variables. This in turn means that software for the simulation of each of the corridors must be coded separately with the particular set of variables built in. The global nature of the variables does make it easy to pass information to the cranes, though. The corridor number is generated from the random value sequence initialized by job 1.

```
RV(@RNDMCORR,1,10)
```

Also, the color foreground of R is changed so that it reflects the overlay color of the corridor input buffer it is designated for. The background is set as defined by @COLOR and the SA instruction.

```
SA(@COLOR,@RNDMCORR)
```

Now R is ready to branch to the set of instructions for its assigned corridor.

```
IF(@RNDMCORR,EQ,1:CORR1)
IF(@RNDMCORR,EQ,2:CORR2)
      .               .
      .               .
      .               .
IF(@RNDMCORR,EQ,9:CORR9)
JP(:CORR0)
```

It should be noted here that the tenth corridor is designated as corridor 0 and will be referred to as such throughout. This is solely for purposes of program readability; all of the other corridors have single digit numbers and by dropping the "1" from "10", the labels and variables for corridor 10 will line up with those of the other nine.

The next step is to examine the path an R takes, given its corridor number. The logic for the first corridor will be fully explained here; the remaining nine sets of corridor-dependent software behave in a parallel manner. Without any delay, the R moves onto the main conveyor under implicit computer control as soon as a valid open spot is recognized. (The branching procedure above contained no instructions requiring clock time.) As the next steps utilize instructions which will move R around the screen, it is necessary to use the

"R=" directive; this will give PCModel's loader a location as reference for the subsequent relative moves. There is no run-time value in this, but it is extremely handy to ensure during the loading phase that the route is indeed following the path intended. Once R is on the conveyor, it moves to the conveyor's right edge at the conveyor's constant rate of 3 sec per half meter (or 6 sec per meter).

```
:CORR1     R=(XY(44,60))
           MU(1,0)
           MR(30,3)
```

The reason the route above stops where it does (in the lower right hand corner of the conveyor) is that some R's will have to circle the conveyor one or more whole laps before the input buffer for an R's corridor has an opening. The right hand corner is a good place to loop because here no code will be duplicated; that is, the MR instruction above only moves R from the receiving position to the right edge. The next time around the R must be moved right from the left edge, not the receiving position. To get to corridor 1, R must move to the upper conveyor and then left to input buffer 1. The rate going up is 6 sec/m and the rate moving to the left is 3 sec/0.5m, or 6 sec/m also.

```
:CONT1     MU(4,6)
           ML(65,3)
```

Now, with R at the entrance to input buffer 1, the number already in the buffer, @INBUF1, must be compared with the buffer capacity @BUFCAP. If it is less than the capacity, there is room for at least one more R; thus R is transferred to the input sequence.

```
IF(@INBUF1,LT,@BUFCAP,:INPUT1)
```

Otherwise, R stays on the main conveyor belt and goes around again; R is moved to the left edge, down to the lower level, over to the right edge, and then repeats as explained before.

```
ML(3,3)
MD(4,6)
MR(68,3)
JP(:CONT1)
```

Once an R is accepted into the input buffer, it has only to wait for the crane to finish with its previous tasks to get to it. The entry of the R in the input buffer causes the number in the buffer to increase, and R is moved up the input buffer to wait. Note again the use of the R= directive to relocate the reference for relative moves after a branch.

```
:INPUT1   R=(XY(9,55))
          IV(@INBUF1)
          MU(4,6)
```

When the crane takes R from the input buffer, it exits from the system. The problem is how to have the crane know that there are received pallets waiting. This is accomplished by incorporating a pair of flags which are used by both the corridor sequence and the crane route. They are the @RECRQ1 and @OKRECRQ1 variables. The corridor sequence sets the RECeiving ReQuest flag by incrementing @RECRQ1 whenever an R gets to the entrance of the corridor (i.e., it is at the front of the input buffer).

```
IV(@RECRQ1)
```

The crane route will be looking for this to happen; when it does and the crane is free to replace the R, the crane route will increment @OKRECRQ1 to signal OKay on the RECeiving ReQest.

Until @OKRECRQ1 is set, R must wait at the front of the input buffer.

```
        :BACK1    DN
                  IF(@OKRECRQ1,EQ,0,:BACK1)
```

Thus, when the crane takes the received pallet for storage, R can exit the system. As it leaves, it resets both the @RECRQ1 and @OKRECRQ1 flags for the next R to use.

```
                  DV(@OKRECRQ1)
                  DV(@RECRQ1)
```

The count for the input buffer is also decremented to reflect the fact that the input buffer now has room for one more R.

```
                  DV(@INBUF1)
```

Lastly, R jumps to the end of the route used by all of the corridor receiving sequences.

```
                  JP(:INDONE)

        :INDONE   ER
```

The corridor sequences for the other 9 corridors follow exactly from that given here for the first; all of the variables and labels are changed to reflect the number of the corridor. The moves along the upper conveyor must be adjusted for the different corridors.

By allowing the received pallets to exit the system at this point, much effort is saved in the coding effort. The presence of the received pallets after the R's leave the system is kept track of in the variables for corridor quantities. This will be examined in full detail in the section covering the crane routes.

With the route for the received pallets constructed and the interrelationships among jobs defined, the job statement for the route can be given. The initialization job will be of highest priority (0) with the synchronization job following (1). Thus the receiving and shipping pallets, as well as the cranes, will all be of the next level priority (2). The job statement for the receiving pallets is

J=(3,R,3,0,0,2,5000)

indicating that job 3 will follow route 3, and job 3's objects will use the character "R" to represent them on the screen. Additionally, there will be 5000 pallets received before the job ends; this value was chosen arbitrarily.

3. Pallets to be Shipped The shipping requests for pallets are next. The shipping requests will be modeled by the objects of the shipping pallet job, denoted by "S". All of the request for pallets to be shipped enter the system at a common point, perhaps the office for such requests. An S is then forwarded to the computer control of the crane for the corresponding corridor. Once there, S waits in the buffer for crane shipping requests. The crane has to deal simultaneously with R's and S's; the procedure for handling both will become apparent in the development of code for a crane.

Once the crane does get to the S, there is the delay time for crane movement to consider. The position of the pallet to ship must be calculated and the crane delayed for the amount of time to move from its current location to that of the pallet and then to the output buffer. This arrangement for retrieval from the corridor parallels that for placement in the corridor described for received pallets. Once time has been allowed for the crane to finish its task, an S can move from the buffer for shipping requests to the output buffer for the corridor. It is here that S waits until an empty position on the warehouse conveyor is free; when a spot arrives, S is transferred to the conveyor by implicit computer control (as was the case for an R entering from the receiving area). It should be noted that both R's and S's will be vying for the positions on the belt. Once on the belt, the S moves to the shipping area and is deleted from the simulation. This deletion is the operation wherein the pallet would be placed on board a truck or freighter to be transported away.

Again referring to Figure 1, the position XY(34,69) is seen to have been selected as the shipping request center. The arrival rate of shipping requests will be maintained in the variable @SHIPRATE. As before, the use of a variable rather than a constant here makes possible the changing of the rate during the simulation run. As job numbers 1, 2, and 3 have been spoken for, the S job will be assigned job number 4. To begin the S route then,

```
BR(4,XY(34,69),@SHIPRATE)
```

is used. Immediately following the beginning of the route is placed a labeled instruction to be branched to when a shipping request buffer is full. This will be explained shortly.

```
:REPEAT   DN
```

The first sequence for an S is the determination of a corridor for S. For purposes of the simulation, this assignment of corridors to requests is done randomly; the GPSS model also makes a random selction.

```
RV(@RNDMCORR,1,10)
```

As was done for the R job objects, the S job objects have their foreground color set to the color used for their respective output buffers on the overlay; the random corridor number carries this information. This allows insight into the conveyor load; for instance, the simulation may be halted temporarily in order to observe exactly which corridors have R and S job objects on the conveyor and further, how many of each. @COLOR is used for the background as defined by the variable value.

```
SA(@COLOR,@RNDMCORR)
```

S can now transfer to the set of instructions coded explicitly for its corridor.

```
IF(@RNDMCORR,EQ,1,:EXIT1)
IF(@RNDMCORR,EQ,2,:EXIT2)
       .                .
       .                .
       .                .
IF(@RNDMCORR,EQ,9,:EXIT9)
JP(:EXIT0)
```

It should seem reasonable that the handling of an S job will parallel that used for an R job; one is coming in, the other going out of the warehouse. Just as the code for each of the corridor input sequences is similar to the others given a change of labels, distances, and relative references, so will be the code for the corridor exit sequences. An examination of the code required for exiting corridor 1 will serve to explain the sequences for the remaining corridors.

Given that S has been assigned a corridor, the first event to occur is a simple check to see if the buffer for shipping requests on that corridor is full. If it is, the S will simply be reassigned to a different corridor. This decision is made to ensure that the system can handle the shipping requests at the specified arrival rate. The other possiblility for an S that finds its buffer for shipping requests full is to have it branch out of the simulation altogether. This would only serve to lighten the load placed on the simulation; reassigning the S to a different corridor assures a conservative estimation of system capability.

The code to check the shipping request is

```
:EXIT1    R=(XY(7,47))
          JB(1,XY(7,47),:REPEAT)
```

where R= is used as before to allow the loader to accurately represent the job path. Note that a different approach to buffer capacity is used here, as compared to that for the corridor input buffers. As described previously, use was made of a counter to keep track of how many R's would be waiting on the crane at any given moment. Here, the Jump if path Blocked instruction is used to determine if the buffer is full; note that no explicit variable is defined or required. The buffer will be full if this space is blocked because the last shipping request to enter the buffer has not moved forward due to shipping requests occupying all of the preceding buffer positions. A trick of this sort could be helpful in large simulations where the physical storage for variables becomes a problem. It also eliminates the need for the program logic to maintain the buffer count. On the other side of the coin, implying the maximum size of a buffer by coding an instruction such as this makes the program less changeable. If it is desired to change the buffer capacity, even if only by one, then the program itself must be modified. These considerations suggest that such a contrivance should only be used in the case of constant, or unchangeable buffer sizes. The logic is coded here in this manner to illustrate an alternative to the explicit buffer variable concept used before.

The other point to note concerning the JB instruction is the location to which an object transfers if the location is indeed blocked. It is the instruction labeled :REPEAT given earlier. It might seem that the :REPEAT label could be attached directly to the RV instruction immediately following it in order to save a line of code and to save the time of the clock increment caused by the DN instruction. The reason for separating the JB destination with the DN instruction becomes apparent when one gives thought to the behavior of the system if it becomes saturated with shipping requests. It may well be that all of the shipping request buffers become filled at some point. Consider what would happen if an S were to enter the simulation at such a time. It would be assigned a random corridor, which would have a full buffer; it then immediately branches to another random corridor which would again have a full buffer. This sequence would be in essence an infinite loop. No hope of termination would exist because none of the instructions in the cycle require any clock time. The inclusion of the DN instruction prevents this from happening by allowing the S to attempt to enter only one shipping request buffer per clock second. Meanwhile, the cranes will be allowed to continue working, and hopefully, some corridor space will become available.

Assuming that the non-full shipping request buffer located is in corridor 1, the progress of S continues. Since a position for S exists, it is transferred without delay from the office for shipping requests (by implicit computer control) to the buffer for corridor 1 by

```
MA(XY(7,47),0)
```

where XY(7,47) is the first position in the buffer. Now, consider the behavior of the crane at this point. When the crane finishes its previous tasks and finds that an S is waiting in the buffer, its computer control will calculate the storage location of the pallet to be shipped and the crane will immediately begin the process of moving to the location and returning with the pallet. On first thought, this would probably appear to be all well and good. However, the possibility exists that the warehouse conveyor is currently saturated with R's and S's, and thus no room remains for additional S's to be transferred to the conveyor. If this is the case, then the output buffer for the corridor will become full, with its S's waiting for positions to open up. Unless the crane checks for this occurrence, it will bring back a pallet and have no place to release it. Therefore, a check is made as soon as the request reaches the end of the request lane but before the crane route is informed it is present.

```
MD(3,0)
DN
TP(1,XY(7,53))
```

The TP instruction is employed to check if the last position in the output buffer is blocked. Again, if this last position is occupied, it is because the position following it is occupied as well (the buffer capacity is 2) and the object in the foremost position cannot move ahead. This buffer position is XY(7,51), as can be seen in Figure 1. Note the use of the DN instruction to prevent the test from being performed before the previous shipped pallet has a chance to enter the output buffer. Without the DN instruction, it is possible for too many objects to move in the space of a single clock period to the output buffer.

Once it is known that space exists in the corridor's output buffer, the count of shipping requests for the corridor can be incremented. It is essential that this incrementation not be done until now in order to prevent the crane from retrieving a pallet before a space exists for it.

```
IV(@SHPRQ1)
```

With S at the bottom of the request buffer, there is nothing to do but wait until the crane signals it has brought a pallet to the output buffer. The crane does this by setting the flag @OKSHPRQ to 1, indicating that the SHiPping ReQuest is OKay to proceed. Until the flag becomes 1, the S waits here.

```
:HOLD1    DN
          IF(@OKSHPRQ1,EQ,0,:HOLD1)
```

Once the crane has signaled that a pallet has been brought up from its storage position to the corridor entrance, S is ready to move from the shipping request buffer to the corridor output buffer. Before this is done, however, the @OKSHPRQ1 flag is reset for use by the next S and @SHPRQ1 is decremented so the crane can know if there are any shipping requests pending.

```
          DV(@OKSHPRQ1)
          DV(@SHPRQ1)
```

It should be observed that the purpose of this pair of variables parallels the use of the @RECRQ1 and @OKRECRQ1 variables used for the R jobs.

R then moves down the output buffer immediately, so as to occupy the buffer positions before another fetch can be requested of the crane when no room actually exists.

```
          MD(4,0)
```

When S gets to the entrance onto the conveyor, it must wait here just as the R objects did for the two events that must occur in order: (1) S must be synchronized with the passage of pallets on the belt, and (2) the current position must be unoccupied in order for S to be able to be placed on the conveyor.

The synchronization problem has an added wrinkle from that considered for the R jobs. As the R jobs were placed on the belt at only one position, that position was the only one of interest for synchronization so it was decided to use the synchronization value, kept in the variable @GO, of 0 to determine valid times there. This same value of 0 is not necessarily the one to use for other conveyor positions. However, with one position and its @GO value defined, the @GO values for the corridor output buffer entries to the conveyor can be determined. For corridor 1 the output position is XY(7,55). For the receiving lanes entry the position is XY(44,59). (Both of these positions can be obtained from Figure 1.) As discussed before, an R placed on the conveyor at @GO = 0 travels to the right 30 spaces at 3 seconds per space, for a total of 90 seconds. Next, the R travels up 4 rows at 6 seconds per row, taking 24 seconds. Lastly, to reach corridor 1's output buffer position the R must travel left 67 spaces at 3 seconds per space, requiring 201 seconds. The total elapsed time is 90 + 24 + 201, or 315 seconds. Now remove the complete counts of 12 from 315; this is 315 mod 12, or 3. Thus the value of @GO when the R is in the position before the corridor output buffer is 3; consequently, this is the @GO value to check for before allowing a pallet to be placed on the conveyor.

```
:WAIT1    DN
          IF(@GO,EQ,3,:READY1)
          JP(:WAIT1)
```

The synchronization values for exits from the other nine output buffers can be arrived at similarly. It is actually much simpler after the initial value is obtained. Since the @GO value for output buffer 1 at XY(7,55) is known, the GO value for output buffer 2 at XY(14,55) comes from the determination of the time it takes an object to move from one buffer to the other, the distance being 14 - 7 = 7 spaces at 3 seconds per space, or 21 seconds. 21 mod 12 is 9 @GO units. Since XY(14,55) occurs prior to XY(7,55) in the direction of conveyor flow, 9 is the count to back up from 3; thus 3, 2, 1, 0, 11, 10, 9, 8, 7, 6 is the sequence with 6 being the @GO value for XY(14,55). Lastly, since all of the output positions are 7 spaces apart, the constant value of 9 may be used to back up from output position 2 to 3, 3 to 4, and so on.

Given that the conveyor is synchronized, the position must still be clear. As was explained earlier, each object is considered to be occupying two horizontal spaces due to the scale being employed, so two spaces must be checked on the conveyor to ensure that the current position is unoccupied.

```
:READY1   R=(XY(7,54))
          JB(2,XY(7,55),XY(6,55),:WAIT1)
```

Once the position is known to be present and unoccupied, the S object is placed on the conveyor without delay by the implicit computer control and proceeds around the conveyor to the shipping area of the simulation.

```
MD(1,0)
ML(1,3)
MD(4,6)
MR(29,3)
MD(10,6)
```

Note that since the shipping area is being simulated as requiring no time, no logic to check for "backing-up" in the shipping area is needed.

Finally, the S jumps to the ER used by all of the corridor shipping sequences.

```
                    JP(:OUTDONE)
          :OUTDONE  ER
```

This type of transfer was also done for the receiving sequences. It is necessary to use this sort of control flow because only one ER is allowed for each BR. Even if this were not the case, it is always preferable to develop code with only one entry and exit.

This section has developed the instructions necessary to generate a shipping request and then to model its passage through a single corridor. As was the case for the receiving section, the code outlined for the single corridor can be propagated to the other 9 by simply changing the label indices, the relative references, and some of the traveling distances (from the output buffer to the left edge of the upper conveyor). This serves to again emphasize PCModel's weakness due to its lack of both arrays and variable move capability. If the language had incorporated these features, much of the parallel behavior could be incorporated in links.

As was done for the receiving pallet job, the shipping pallet job is now coded last with all of the other information in place. The shipping job, like the receiving job, has priority 2, leaving priorities 0 and 1 for the initialization and synchronization jobs respectively. The job statement is coded as

```
J=(4,S,4,0,0,2,5000)
```

indicating that job 4's objects will be represented on the display by 'R' and that job 4 will follow route 4. Lastly, 5000 requests for pallets to be shipped will be placed before this job is exhausted. The 5000 figure was selected arbitrarily.

<u>4. Conveyor Synchronization</u> At this point, the routes for the receiving and shipping pallets have been coded. Both of these have made use of the synchronization variable @GO; it would be used to determine the appropriate times that a valid conveyor position would be present for entry onto the conveyor. In this section, the determination of the variable @GO is examined through the development of the synchronization job's code.

The synchronization job object enters the simulation during the first second without delay.

```
BR(2,XY(65,68),0)
```

Route number 2 was reserved for the synchronization job earlier and the position XY(65,68) was chosen entirely arbitrarily. Actually, no screen movement occurs for this job, so a position is not logically required here; however, it must be included as the BR instruction demands its presence.

As was described previously, @GO was assumed by both the receiving and shipping pallets to have a value on the closed range (0,11) which would change each second. This interval was derived from the conveyor speed. At 0.5m per 3 sec, the conveyor would travel 2m every 12 seconds. Thus a flag set to zero every twelfth second would indicate to the jobs when a valid conveyor position was present. Assuming that @GO will be set to 0 by the initialization job, its value would need to be incremented each clock second.

```
        :BACK      IV(@GO)
```

Now, the value for @GO can be checked to determine if it is still on the proper range (i.e., its value is no greater than 11, or equivalently, still less than 12).

```
                   IF(@GO,LT,12,:OVER)
```

If the value has incremented to 12, which will obviously occur every 12 clock seconds, then @GO must be reset for the current second.

```
                   SV(@GO,0)
```

In either case, @GO has now been set for the current clock second, so this value must now be preserved for the receiving and shipping jobs to reference during this second. This is done by delaying the synchronization job object for one second.

```
        :OVER      ST(1)
```

This value is made present for the complete second to the other jobs by giving the synchronization job a priority of 1; the receiving and shipping jobs, along with the cranes to be coded, will operate at priority 2. Thus the synchronization will be processed before any of the other jobs during any given second. (The priority 0 is being reserved for the initialization job, which will only run once before the first second of simulation.) Note that no other instruction for the synchronization job requires any clock time.

Once the @GO's value has been maintained for the current second, the job object is transferred to the beginning of the loop so that a new @GO value may be determined for the next second.

```
                    JP(:BACK)
```

Note that the job releases only the one object into the system. This same object will remain active for the duration of the simulation; thus it never reaches an ER, or End Route, instruction. Nevertheless, one must be included to indicate to the loader that the instruction sequence for the job route started with the preceding BR instruction is completed.

```
                    ER
```

This same technique of using a single job object throughout the simulation will be employed in the modeling of the cranes.

As was done for the prior jobs, coding of the job statement has been left for last. As explained above, the priority of the job will be 1 and the number of job objects is also 1. The job was assigned job number 2 before any coding began.

```
          J=(2,#,2,0,0,1,1)
```

Thus job number 2 will take route 2, and the screen image for the job is the '#' character, chosen arbitrarily as the job really has no logical display output.

<u>5. The Corridor Cranes</u> With the behavior of the receiving and shipping jobs completely determined, coding for each of the ten corridor cranes can begin. The decisions reached concerning the information the corridors would make

known to the cranes will now be utilized to create the crane behavior. The receiving objects passed the presence of jobs through @RECRQn and expected acknowledgments through @OKRECRQn (where n is the subscript of the corridor); the shipping objects likewise utilized @SHPRQn and @OKSHPRQn. The use of these variables will now be seen from a crane's point of view.

a. Crane Behavior As done for the corridor input and output sequences belonging to the receiving and shipping objects respectively, the behavior for only one crane will be examined here. As before, the changes necessary in the code to go from one crane to the next will consist basically of changes to label and variable indices and to screen positions.

Two facets of PCModel that have not been examined as of yet will be employed for the cranes. They are links and object parameters. Links will be used in the same manner as subroutines are used in typical programming languages; their inclusion will greatly relieve the coding and debugging efforts by making the same sequence of route instructions available to all ten corridors. Object parameters will take the role of subroutine parameters, carrying information to and from the links. The global nature of variables in PCModel forces this use of object parameters in links requiring clock time (as will be seen); otherwise, the links would be severely restricted in their capability to affect different corridor statistics.

Object parameters will also play another role in the modeling of the cranes. As noted in the section detailing the synchronization job, each crane will be modeled by a single non-terminating job object; in other words, each crane job will create only one object, which will remain in the simulation for its duration. This being the case, object parameters can be used to maintain some of the statistics concerning a crane and its corridor. This in turn reduces the number of variables which require storage to be allocated and it makes whatever information kept in the parameters known to any links a crane object, hereafter referred to as 'C', enters.

The information that must be maintained for each corridor includes the current crane position, both vertically and horizontally, and the number of pallets stored in each of the four zones of the corridor. This totals to six variables, exactly the number of parameters for an object. However, such an assignment does not leave any parameters for use in transferring intermediate information between links or for use during links as work variables. These types of uses will be seen during development of the links.

Before beginning the coding for crane 1, its behavior must be completely defined. It is present in the warehouse when the simulation starts and begins its work as soon as a pallet enters its input buffer. When this occurs, it takes the pallet from the input buffer and moves it to the bin assigned to it by computer control. It stays at that position and checks to see if a request has been made for a pallet to be shipped. If so ,it moves to the location of the pallet (again determined by computer), picks it from its bin, and moves it to the output buffer. Whether or not a shipping request sequence takes place, the crane next polls the receiving buffer to see if another pallet is waiting to be stored in the corridor. At this point, the cycle repeats and will do so until simulation end. Additionally, checks must be made so that no attempt is made to store a pallet in a zone that is already full or remove a pallet from a zone that is currently empty.

Now consider what pieces of this sequence of events can be grouped into links. Basically, whether a crane is storing a received pallet or bringing a pallet to the output buffer to satisfy a shipping request, two events must take place. First, the zone of the pallet must be determined; this would include checking the zone for the full (on receiving) or empty (on shipping) condition. Second, the crane must be moved from its current position to that of the pallet and then to the pallet's destination. Each of these two events is different enough for its receiving and shipping counterparts to warrant the development of a separate link for each. Thus, the utilization of four links is in order: determination of the zone for a received pallet, movement of the crane during receiving, determination of the zone of a pallet to be shipped, and movement of the crane during shipping. The development of these four links will be detailed first, in order that their requirements can be reflected in the crane route. As the determination of the zone for a received pallet is the first event to occur for a crane, the link for it will be developed first.

Before the software is decided upon, the utilization of the cranes six parameters must be defined. As noted previously, each corridor requires six variables: the four zone quantities and the crane's horizontal and vertical position. Now consider the nature of the links to be used. Global variables are naturally accessible inside each link; as long as the link causes no clock delay for the crane object, global variables may be used to pass information. However, if C encounters a time delay at any point in the link, this may allow one or more different C's inside the link and they will be referencing the same global variables while the first C is being delayed. Thus a new C would cause interference with the global variable values that the previous C was using when it entered the link. The solution to this problem is to ensure that one of two possible scenarios is followed: (1) no time delays are caused by the link, thus allowing the use of global variables, or (2) no global variables are specified (i.e. variables are passed solely as object parameters), so that time delays may be encountered by different objects without causing interference for others. Each of these scenarios will be used, the first by the link for determination of the zone and the second in the link for the movement of the crane, which will certainly require time delays.

Two factors come into the final definition of the crane object parameters. The first is that the four zone variables will be used only by the first link which will require no clock time. Thus, they may be passed globally. Secondly, the crane each corridor. Crane object parameters 3 and 4 are selected to contain the number of pallets currently in the selected zone and the offset to the zone, respectively. The determination of the bin from this information will be detailed in the link for receiving movement. Lastly, the link should set some sort of flag to indicate if the zone was already full so that no attempt is made to store yet another pallet there. Parameter 2 is set aside for this purpose. A '1' in this parameter will indicate a full zone to the crane route.

The coding of a link always begins with the Begin Link instruction, specifying the name by which routes will refer to it. For the link to determine the zone for received pallets,

```
BL(!RECVZONE)
```

is used. The next thing to do is insure that the flag for "fullness" is cleared upon each entry to the link.

```
SV(OBJ@2,0)
```

Now the physical zone can be selected for the received pallet. As detailed during the collection of data for the problem, the frequency for storage in the four zones A, B, C, and D breaks down as 40%, 30%, 20%, and 10%, respectively. To model such a breakdown, a random number is generated between 1 and 100.

```
RV(@RNDMZONE,1,100)
```

This RaNDoM ZONE value can then be used to branch based on the percentages. For zone A, at 40%, the range would be (1,40), so

```
IF(@RNDMZONE,LT,41,:RZONEA)
```

position will be required in the calculation of distance and thus time to move in the second link; thus, it would be immensely helpful if they could be maintained in the object parameters. This in turn will alow the link to encompass more logic, since the crane object will not have to return to its specific route for time delays. Tentatively then, two of the crane parameters will be used for its horizontal (arbitrarily chosen as parameter 5) and vertical (parameter 6) positions. The other four object parameters willl be open for use as needed in the links.

<u>b. Handling a Received Pallet</u> For the received pallet zone-determination link, the decision must be made as to what information needs to be returned to the crane route. Since the first four object parameters are open, they may as well be used for this purpose to reduce the storage required for variable allocation. It is the job of this link to acquire the information necessary for the receiving movement link to determine the specific bin the pallet is to be placed in. Knowing the bin, the link can then determine distances of movement and thus time required to move. To determine the bin, two values currently accessible to the object will suffice, specifically the offset from the input buffer at the head of the corridor to the zone and the number of pallets currently in the selected zone. The offset to the zone will be a different constant for each of the four zones, but it will be known on entry to the link. The current number in the zone can be obtained from the global variables that will be maintained for

is coded. Zones B and C are coded similarly for their percentages, keeping in mind that as an object passes an IF it is known that the random percentage must lie in the remaining interval.

```
IF(@RNDMZONE,LT,71,:RZONEB)
IF(@RNDMZONE,LT,91,:RZONEC)
```

The same type of instruction could be coded for zone D, but instead

```
JP(:RZONED)
```

is used. This probably better serves to aid in the detection of a coding error; if a number is incorrectly specified above, more or less items will be easiest to notice in zone D, the least frequent in traffic.

Now with the zone specified, information pertaining to it can be gathered. Assume at this point that the global variables containing the corridor zone quantities were placed in the variables @ZONEA, @ZONEB, @ZONEC, and @ZONED for use by the link. Consider first the treatment of zone A. As noted in the data collection process, zone A is defined to occupy 13/50 of the corridor, since equal percentages of volume were deemed inappropriately complicated. Since each corridor's capacity is 1000 pallets, zone A's capacity is 260, and this is the value used for identifying a zone-full condition.

```
:RZONEA    IF(@ZONEA,EQ,260,:ZONEFULL)
```

Assuming the zone is not full, crane parameter 3 is set to the number currently in the zone, and then this value is incremented for use in updating the zone quantities in the crane route.

```
                    SV(OBJ@3,@ZONEA)
                    IV(@ZONEA)
```

Note that parameter 3 contains the number before the current pallet is stored; it is important to keep this in mind when the determination of the bin location is made in the next link.

The other information required for the bin location is the offset to the zone, as explained before. As zone A starts at the head of the corridor, its offset is zero.

```
                    SV(OBJ@4,0)
```

With the information for zone A in place, the crane object can return to its route.

```
                    JP(:OKSTORE)
             :OKSTORE  EL
```

Had the zone been full, the object would have transferred to the instruction immediately prior to the EL instruction to set the flag parameter.

```
             :ZONEFULL SV(OBJ@2,1)
```

The handling of the other 3 zones is identical to that of zone A, in much the same manner as the treatment for the corridor input and output of the receiving and shipping routes. The different values for zones B, C, and D include the varying zone capacities: 240, 260, and 240; the zone offsets: 13, 25, and 38 meters; and the variables used for the zone quantities: @ZONEB, @ZONEC, and @ZONED. Note that the offsets were arrived by applying the corridor percentages to the corridor length of 50 meters: 13/50 for zone A implies 13m to zone B; 12/50 for zone B implies 12m plus the previous 13, or 25m, to zone C; and

13/50 for zone C implies 13m plus the previous 25, or 38m to zone D. Substitution of these changes into the code for zone A yields the code for zones B, C, and D.

Assuming that C has returned to its route just to update corridor 1's zone quantities from the link's global zone variables before another crane has time to interfere with them, its progress can now be charted through the link for receiving movement. Remember that time delays will occur here so it is crucial that all calculations be done using C's parameters; thus more than one crane can use the link at the same time. The link begins as described in the previous section.

BL(!RECVMOVE)

Assuming that the crane route did not affect the object parameters when the crane returned there from !RECVZONE, the values saved there are still in place. Parameter 3 contains the number in the selected zone and 4 has the offset to the zone. Parameter 1 was unused by the link so it is free for any use deemed necessary here. In addition to this, parameter 2, the zone full flag, fulfilled its purpose upon return to the route by indicating whether the zone was full or not; it is now free as well. Thus, C parameters 1 and 2 may be used as work variables where needed. This is the basis of the capability that allows other crane objects to use the same link simultaneously, without affecting one another.

The first thing the crane must do in order to store a pallet located in the input buffer is to move from its current location to the end of the corridor. Its current positions (horizontal and vertical) are located in object parameters 5 and 6. The end of the corridor is defined to be position (H=1, V=1); this is the same position as that for the first object to be placed in zone A. Consider first the horizontal move. Since the value for the horizontal position of the crane will be required for later work, it is copied to parameter 1 here.

```
SV(OBJ@1,OBJ@5)
```

This is done to provide the most accurate statistics possible on the object parameter display. The parameter screen will not be changed until the crane has had time to move.

The distance then is calculated by obtaining the difference between the desired position (constant at H=1, V=1) and the current position. In other words, subtract 1 from, or decrement, the horizontal position in OBJ@1.

```
DV(OBJ@1)
```

After decrementation, the parameter will contain a distance, not a position. Multiplying it by a rate of seconds per meter will give the time to move in the horizontal direction. The rate is kept in the global variable @HORZRATE.

```
AO(OBJ@1,*,@HORZRATE)
```

This being done, the parameter now contains a time, no longer a position or distance. This time can be used to delay the crane object for simulation of movement.

```
ST(OBJ@1)
```

Lastly, the crane's horizontal position is set to reflect the move.

```
SV(OBJ@5,1)
```

When the horizontal move is complete, the crane moves vertically. The logic and coding follow from that of the horizontal.

```
SV(OBJ@1,OBJ@6)
DV(OBJ@1)
AO(OBJ@1,*,@VERTRATE)
ST(OBJ@1)
SV(OBJ@6,1)
```

After these delays, the crane will be positioned at the head of the corridor, at the input buffer exit; once there, it removes a pallet from the buffer, an operation which is assumed to take negligible time. The crane is now ready to move to the pallet's position; thus the next step is the calculation of this position. Breaking the problem down further, consider only the distance to move in the horizontal direction. The crane is in horizontal position 1. Adding to 1 the offset for the zone (from OBJ@4) will yield the horizontal position of the first bin in the zone. The calculation from there is more involved.

First make a copy of the number in the zone in OBJ@1.

```
SV(OBJ@1,OBJ@3)
```

This provides a copy of the number which can be worked with, so as not to destroy the original; this value will be required later in the vertical calculation.

For each horizontal meter of corridor length there are 20 bins (10 high on either side); integer division (DIV) of the number in the zone by 20 thus yields the whole number of these sets of 20 filled. For instance, if there are 17 pallets in the zone, then 17 DIV 20 = 0, meaning no sets of 20 are filled, and the next pallet would be placed in this "zeroth" horizontal meter; if there are 46 pallets in the zone, then 46 DIV 20 = 2, two sets of 20 are filled, and the crane would have to move 2 meters further down the corridor, past the filled "zeroth" and first meters, to place the next pallet. Therefore dividing by 20 yields the distance into the corridor the crane must move.

```
AO(OBJ@1,/,20)
```

Adding in the offset to the zone gives the distance the crane must move from position 1 to get to the new position.

```
AO(OBJ@1,+,OBJ@4)
```

OBJ@1 will be used in the delay for horizontal movement.

Lastly, the new horizontal position is determined and saved in OBJ@2 for placement in OBJ@5 once the time has elapsed for the crane to get there. The new position is obtained from the fact that the crane is in position 1 and OBJ@1 is currently the distance; thus saving a copy of OBJ@1 in OBJ@2 and then incrementing OBJ@2 puts the position in OBJ@2 while leaving the distance in OBJ@1.

```
SV(OBJ@2,OBJ@1)
AO(OBJ@2,+,1)
```

This allows OBJ@1 to be worked with again and it prevents the position parameter of the crane from being set until the crane gets there.

Knowing the required horizontal distance, the required movement time can be determined as was done for the movement to position 1. Multiply the horizontal distance by seconds per meter to get the time and then delay for that time.

```
AO(OBJ@1,*,@HORZRATE)
ST(OBJ@1)
```

Now the horizontal position of the crane can be updated from the copy saved earlier.

```
SV(OBJ@5,OBJ@2)
```

Now in the proper horizontal position, the crane is ready for vertical movement. Basically, the determination of the next vertical bin position to be filled is an extended application of the logic used for the horizontal position. As before, get a copy of the number in the zone to work with.

```
SV(OBJ@1,OBJ@3)
```

To determine how many bins are occupied of the 20 vertically accessible to the crane in its current horizontal position, remove the whole sets of 20 from the number in the zone. In other words, take the number in the zone modulo 20. Doing this with PCModel's four arithmetic operations requires several steps and two variables. First get the number of whole 20's by dividing the number in the zone by 20.

```
AO(OBJ@1,/,20)
```

Second, get the greatest multiple of 20 contained in the zone by multiplying by 20.

```
AO(OBJ@1,*,20)
```

Now, subtracting this value from the number in the zone will produce the desired modulo. A copy of the zone quantity is made in OBJ@2 (to preserve OBJ@3 for possible future use), and then the subtraction is performed.

```
SV(OBJ@2,OBJ@3)
AO(OBJ@2,-,OBJ@1)
```

OBJ@2 now contains the number of filled bins in the current vertical 20. In fact, if the bins were arranged vertically on one wall, OBJ@2 + 1 would be the vertical position of the bin for the next pallet. However, this is not the case; there are 10 bins arranged vertically on either side of the corridor. Therefore the vertical distance to the first empty bin is OBJ@2 DIV 2. For example, 0 or 1 filled bins means the distance is 0, 2 or 3 filled bins yields a distance of 1, 4 or 5 filled bins yields a distance of 2. Thus

```
AO(OBJ@2,/,2)
```

puts the vertical distance in OBJ@2. A copy of this distance is placed in OBJ@1 for use in the movement delay.

```
SV(OBJ@1,OBJ@2)
```

Lastly, OBJ@2 is changed from a distance to a vertical position by adding one.

```
AO(OBJ@2,+,1)
```

Again, this is due to the fact that the crane is already in vertical position 1, so 1 plus the distance to move will give the new position. As done horizontally, the vertical position parameter will not be updated until the time has passed for the crane to get there.

Moving the crane vertically consists of the same steps as moving horizontally: multiply distance by seconds per meter, delay for the required time, and then set the new vertical position.

```
AO(OBJ@1,*,@VERTRATE)
ST(OBJ@1)
SV(OBJ@6,OBJ@2)
```

At this point, the crane can place the received pallet, an operation which is assumed to take no appreciable time. The crane's next logical procedure is to check for more pallets to be moved. It has been decided to let the crane wait wherever it placed the pallet; this decision will be explored further in the crane route, which is where the crane object returns at this point.

```
EL
```

c. Handling a Shipping Request Now the focus of attention is turned to the link dedicated to determining the zone to obtain a pallet from to satisfy a shipping request. It is an exact counterpart to the link for determining the zone information for a received pallet. As such, it will not be explained in the detail that !RECVZONE was. As for every link, this one begins with the BL instruction declaring the name it will be referred to by.

```
BL(!SHIPZONE)
```

This time, a flag in the value of OBJ@2 will be used to determine the "emptiness" condition for the selected zone. It is set to zero to initialize it for the current C.

```
SV(OBJ@2,0)
```

Again, the zone itself is selected on a percentage basis, 40%, 30%, 20%, and 10% for zones A, B, C, and D respectively. The random number and the zone branch is handled as before.

```
RV(@RNDMZONE,1,100)
IF(@RNDMZONE,LT,40,:SZONEA)
IF(@RNDMZONE,LT,71,:SZONEB)
IF(@RNDMZONE,LT,91,:SZONEC)
JP(:SZONED)
```

Considering only zone A for the moment, the first thing to check is the possibility that the zone could be empty.

```
:SZONEA  IF (@ZONEA,EQ,0,:ZONEMPTY)
```

Next, the number currently in the zone is saved in OBJ@3 and then the number is decremented to reflect the retrieval of a pallet upon return to the crane route.

```
SV(OBJ@3,@ZONEA)
DV(@ZONEA)
```

The zone offset is saved and the crane object is ready to transfer back to the crane route.

```
SV(OBJ@4,0)
JP(:OKGET)
```

```
:OKGET    EL
```

To handle the zone empty condition, OBJ@2 would have been set to 1 immediately before encountering the EL return.

```
:ZONEMPTY SV(OBJ@2,1)
```

To generate the code for the three remaining zones, proceed as in the case for the received zone link. Change the label and variable indices and the offsets to the different zones (13, 25, and 38 for zones B, C, and D respectively); note that no change need be made for the zone empty check, as the same value (zero) indicates a zone empty condition for all four zones.

The last link required for use by the ten cranes is the one to model the behavior of the crane as it moves both from its current position to that the link calculates for the pallet to be shipped and then to the output buffer to release the pallet into the system. Observe that a fundamental difference exists between this behavior and that modeled by the link which determined the movements for receiving a pallet. In the receiving situation, the crane had to move from wherever it was currently to the input buffer, and then back into the corridor again. For it, the direction of movement is implied by the behavior; thus, distances can always be calculated to come out as positive. For the shipping case, the crane moves from wherever it is currently to the position determined for the pallet to be shipped; the direction of motion is not implied here, so special checks will have to be made to assure that the subtraction is performed in the proper direction. (As noted in the definition of the AO instruction, a negative result causes the destination to be set to zero.)

In keeping with the established pattern for link nomenclature, the name of the link for determining movement behavior during shipping is !SHIPMOVE.

BL(!SHIPMOVE)

As in the receiving movement modeling, the first thing the crane must accomplish is movement from its current location to that of the pallet to be shipped. Proceed as in the receiving case, first considering the horizontal movement and then the vertical. To get the horizontal position of the pallet, get its distance into its zone first. This is accomplished by copying the number in the zone (OBJ@3) to a work variable

(OBJ@1) and then dividing by the number of bins per horizontal meter of corridor.

```
SV(OBJ@1,OBJ@3)
AO(OBJ@1,/,20)
```

OBJ@1 is thus the offset into the zone; adding the offset to the zone from the corridor (OBJ@4) yields the distance from position 1 to the pallet location.

```
AO(OBJ@1,+,OBJ@4)
```

Finally, adding 1 to this distance produces the actual horizontal location.

```
AO(OBJ@1,+,1)
```

Note that OBJ@1 is no longer a distance, but a position.

Again as before, OBJ@1 is copied to OBJ@2 so that C's parameter may be set to reflect the new position of the crane once the time has elapsed for the move.

```
SV(OBJ@2,OBJ@1)
```

Now comes the task of determining the distance from the current horizontal crane position (OBJ@5) to that of the pallet (OBJ@1). A check is made here to determine in which direction the crane is to move. If the crane will be moving toward the entrance to the corridor (OBJ@5 is greater than OBJ@1), the crane object transfers to the code for that situation.

```
IF(OBJ@5,GT,OBJ@1,:DOWN1)
```

Otherwise, the crane will be moving away from the entrance (OBJ@5 is less than OBJ@1). Thus the distance may be obtained by subtracting the smaller location from the larger.

```
                    AO(OBJ@1,-,OBJ@5)
```

The crane object then transfers to the code which assumes the distance is in OBJ@1.

```
                    JP(:DOWN2)
```

In the event the crane will be moving toward the exit, OBJ@5 is greater than OBJ@1. Again the distance is calculated by subtracting smaller from larger. An extra temporary variable is used to allow the preservation of OBJ@5 until the move has been completed.

```
          :DOWN1    SV(@TEMP,OBJ@5)
                    AO(@TEMP,-,OBJ@1)
```

Since the following code assumes that the distance for the move will be in OBJ@1, the value is transferred.

```
                    SV(OBJ@1,@TEMP)
```

Observe that since no clock time elapses during the scope of @TEMP's use, no other object will be able to affect @TEMP, even though it is a global variable.

Now for either direction of motion, OBJ@1 contains the distance. The time for movement is again obtained by multiplying by the rate of seconds per meter (@HORZRATE) and the delay is implemented with ST.

```
          :DOWN2    AO(OBJ@1,*,@HORZRATE)
                    ST(OBJ@1)
```

Now the horizontal position of the crane can be updated.

```
                    SV(OBJ@5,OBJ@2)
```

With the horizontal move completed, the vertical move takes place in much the same manner, combining the pattern of !RECVMOVE with the direction logic employed above. Copying the number in the zone (OBJ@3) to OBJ@1 and then successively dividing and multiplying that value by 20 produces the greatest multiple of 20 contained in the zone.

```
SV(OBJ@1,OBJ@3)
AO(OBJ@1,/,20)
AO(OBJ@1,*,20)
```

Again get a copy (OBJ@2) of the number in the zone (OBJ@3); subtracting the greatest multiple of 20 (OBJ@1) from this leaves the number of bins filled vertically in the current horizontal meter of corridor.

```
SV(OBJ@2,OBJ@3)
AO(OBJ@2,-,OBJ@1)
```

Dividing the number of filled bins by 2 gives the distance from position 1 to the next empty bin; adding 1 gives its location.

```
AO(OBJ@2,/,2)
AO(OBJ@2,+,1)
```

The location is copied to OBJ@1 so that OBJ@2 may be used to update the vertical position after the move.

```
SV(OBJ@1,OBJ@2)
```

Now the check is made to transfer if the crane's current position is greater than that desired.

```
IF(OBJ@6,GT,OBJ@1,:DOWN3)
```

If the pallet's location is larger, the distance is obtained by performing the subtraction with it being subtracted from and the object transferring to the code expecting the distance in OBJ@1.

```
AO(OBJ@1,-,OBJ@6)
JP(:DOWN4)
```

Otherwise, the subtraction will be performed in the opposite direction, again employing the use of the variable @TEMP to preserve the vertical location until after the move.

```
:DOWN3    SV(@TEMP,OBJ@6)
          AO(@TEMP,-,OBJ@1)
          SV(OBJ@1,@TEMP)
```

With the distance computed in OBJ@1, the simulated travel time comes from the multiplication by the rate and the object is delayed appropriately; when the crane has "moved", its location (OBJ@6) is set.

```
:DOWN4    AO(OBJ@1,*,@VERTRATE)
          ST(OBJ@1)
          SV(OBJ@6,OBJ@2)
```

The crane is now in the same position as it was for the first movement of !RECVMOVE (i.e., it is somewhere out in the corridor and must move to the entrance). For !RECVMOVE, the crane traveled to the entrance to pick up a pallet for storing; here, the crane is taking a pallet to the entrance for shipping. Since the weight of the pallet is assumed to have no effect on the crane's speed, the coding is identical. The first step is to get a copy of the current horizontal position and decrement it to convert it to the horizontal distance to the corridor entrance.

```
          SV(OBJ@1,OBJ@5)
          DV(OBJ@1)
```

Secondly, multiply by the horizontal rate and use this for the delay.

```
          AO(OBJ@1,*,@HORZRATE)
          ST(OBJ@1)
```

Lastly, update the horizontal position to reflect the crane's movement to the entrance.

```
SV(OBJ@5,1)
```

The vertical movement is handled in a parallel fashion.

```
SV(OBJ@1,OBJ@6)
DV(OBJ@1)
AO(OBJ@1,*,@VERTRATE)
ST(OBJ@1)
SV(OBJ@6,1)
```

This completes the modeling of the crane's shipping movement. Therefore C is returned to its route to proceed with the processing of pallets for the corridor.

```
EL
```

d. Controlling the Crane In the next section, the route which sends the crane to the previous four links will be examined. It is there that the decisions are made concerning when to utilize the previously defined links.

As was done for the programming of the specific corridor input and output procedures, the examination of a single crane route here will suffice to explain the structure used for all ten routes. This section will look at the coding process for crane 1's route.

The object for crane 1 ('C') enters the simulation at the head of corridor 1, XY(8,50), in the first second of simulation.

```
BR(11,XY(8,50),0)
```

As the route numbers 1 through 4 have already been used, the crane routes will be numbered consecutively from 11 to 20. Note that this implies no PCModel restrictions exist on the numbering of routes, as 5 through 10 are completely unused.

During the coding of the links, it was specified that C's parameters would each be designated to contain a variable value; these values would be assumed present by the links upon C's entry into them. Scrutiny of the !RECVZONE and !SHIPZONE links shows that they use no incoming values, but set parameters 2, 3, and 4 before exit. The !RECVMOVE and !SHIPMOVE links use the information that enters in parameters 3, 4, 5, and 6. From this information alone, it should be obvious that parameter 2 is something passed back to the crane route for use there, while parameters 5 and 6 must be set by the route itself before entry into the MOVE links.

Now remembering that parameters 5 and 6 were defined to be the horizontal and vertical positions of the crane respectively, they are initialized here; it is assumed that the crane will be positioned at the entrance to the corridor, ready to take a pallet from the input buffer.

```
SV(OBJ@5,1)
SV(OBJ@6,1)
```

The next five PCModel instructions define the basis for the crane behavior. The first is simply a position to transfer to. It is required by the nature of the route. Only one C will be generated; this C will remain in the simulation for its duration. Thus C must be in an endless loop. The following instruction is the start of each cycle.

```
:CRANE1   DN
```

The purpose of the DN instruction is to prevent C from entering an infinite loop between clock seconds. This would occur because the following behavior may well take no time at all during periods of the simulation, specifically at simulation start.

The first thing the crane does, as explained before, is to check for any received pallets waiting in the input buffer. Their presence would be indicated by a value of 1 for @RECRQ1 (which is set by the receiving route).

```
:RECVING1 IF(@RECRQ1,GT,0,:REC1)
```

This is followed by a DN instuction to prevent entry into a different infinite loop which could occur for reasons to be explained shortly.

```
DN
```

If there were no pallets waiting, C proceeds to the next instruction where it checks for a shipping request.

```
:SHPPING1 IF(@SHPRQ1,GT,0,:SHP1)
```

If, on the other hand, there was indeed a pallet to be stored in the corridor, C transfers to RECeiving1, after which it returns to the shipping check.

Shipping is handled in the same way. If there were no requests waiting, as indicated by the global @SHPRQ1 variable, C proceeds to the next instruction where it transfers to the top of the loop, to be followed by the receiving check.

```
JP(:CRANE1)
```

If there was a request ready to be filled, C transfers to SHiPping1, at the end of which it transfers directly to :RECVING1. This is the reason the receiving check is followed by a DN instruction; it is here that an infinite loop could be entered. Consider the case where there are shipping requests but only empty zones (e.g. at the start of the simulation). First C would transfer immediately to the receiving check. If, at that instant, there were no pallets in the input buffer, C would immediately fall through to the shipping check again and transfer immediately to the receiving check again. This behavior constitutes an infinite loop, thus the DN directly following the receiving check.

It should be noted here that the prevention of an infinite loop can be handled in any number of ways. One alternative might be to have the shipping code transfer to the top of the loop rather than the receiving check. The trade off here is to alter the logic of the shipping code to reflect an inherent problem with programming languages. It was decided to leave the shipping code alone and reflect the problem in the relatively simple master loop, where it would not complicate matters. However, where to do so is definitely a matter of choice and style. The bottom line is that prevention of an infinite loop must be accomplished in some fashion.

Next, consider the behavior of the crane once it has been determined that a pallet is waiting to be placed in the corridor. The first thing to do is determine the information for the zone to place the pallet. As noted in !RECVZONE, the zone is determined randomly, and C's parameters are set to the number in the zone selected (OBJ@3), the offset to the zone (OBJ@4), and as a flag indicating if the zone is already full (OBJ@2). !RECVZONE also assumes that the global variables

@ZONEA, @ZONEB, @ZONEC, and @ZONED will contain the zone quantities for the C which enters the link. Outside of the link, the zone quantities will be kept in subscripted variables, such as @ZONE1A for corridor 1's zone A, @ZONE1B for corridor 1's zone B, and so forth.

Passage by C to and from the link then is accomplished by first setting the zone quantities for corridor 1.

```
:REC1     SV(@ZONEA,@ZONE1A)
          SV(@ZONEB,@ZONE1B)
          SV(@ZONEC,@ZONE1C)
          SV(@ZONED,@ZONE1D)
```

Next, C transfers to the link by the LK instruction.

```
LK(!RECVZONE)
```

Finally, as RECVZONE required no clock time, the values in @ZONEA-D will not have been altered by any other objects, so they may be used to update corridor 1's zone quantities.

```
SV(@ZONE1A,@ZONEA)
SV(@ZONE1B,@ZONEB)
SV(@ZONE1C,@ZONEC)
SV(@ZONE1D,@ZONED)
```

This completes the "subroutine call" procedure as performed for PCModel.

As mentioned above, C's parameters 2, 3, and 4 were set inside !RECVZONE; parameter 2 in particular is used by the route to indicate the "full" condition for the zone. If the zone was full, no attempt should be made to store it. Thus, the code for moving the crane with a pallet is skipped.

```
IF(OBJ@2,EQ,1:FULL1)
```

:FULL1 is a label on the last instruction of the sequence, which transfers C to the check for shipping requests. Note that the pallet in the input buffer is simply ignored; it will be there the next time the crane checks the input buffer. However this time, one of two things could have happened: (1) a space may have opened up in the full zone due to the shipping request that may have just been filled, or (2) a different zone may be determined by the random number process for the pallet. Note that this is a simplifying assumption; actually a pallet would be designated for a specfic zone and have to be handled accordingly, even if the zone were full. This behavior was deemed to be beyond the bounds of this simulation in that it would only serve to complicate the logic.

Now, assuming that the zone is not full, the receive request is acknowledged by incrementing the number of pallets.

```
IV(@OKRECRQ1)
```

Actually, this should not be performed until the crane has reached the entrance to the corridor and picked up the pallet, but the structure of the !RECVMOVE precludes this.

Next, C is immediately moved to position XY(8,47) on the overlay to signal that it is in the process of moving to store a received pallet.

```
MA(XY(8,47),0)
```

The crane and pallet movement takes place by transferring C to the appropriate link.

```
LK(!RECVMOVE)
```

After the crane has placed the pallet in the corridor, the zone quantities are printed on the overlay to display the status of the corridor.

```
                    PV(XY(4,39),@ZONE1D)
                    PV(XY(4,41),@ZONE1C)
                    PV(XY(4,43),@ZONE1B)
                    PV(XY(4,45),@ZONE1A)
```

Additionally, the crane's position in the corridor is updated.

```
                    PV(XY(4,32),OBJ@5)
                    PV(XY(4,34),OBJ@6)
```

Lastly, C is moved to position XY(8,48) to represent that the crane has completed the receiving movement and is waiting out in the corridor for another task.

```
                    MA(XY(8,48),0)
```

With the receiving behavior complete, C transfers to the check for shipping requests. Note that this is also the instruction transferred to in the occurrence of an attempt to store a pallet in a full zone.

```
        :FULL1      JP(:SHIPPING1)
```

The programming for the crane when a shipping request is similar to that used for the received pallets waiting in the input buffers, as has been noted. These similarities occur in much the same fashion as those noted in the parallels between shipping and receiving links for zone determination and movement.

Given the presence of a shipping request, the global variables for the corridor zone quantities are established and C transfers to the link for determination of the zone to pull the pallet from.

```
        :SHP1       SV(@ZONEA,@ZONE1A)
                    SV(@ZONEB,@ZONE1B)
                    SV(@ZONEC,@ZONE1C)
```

```
SV(@ZONED,@ZONE1D)
LK(!SHIPZONE)
```

The zone quantities are then updated upon the return of C.

```
SV(@ZONE1A,@ZONEA)
SV(@ZONE1B,@ZONEB)
SV(@ZONE1C,@ZONEC)
SV(@ZONE1D,@ZONED)
```

The check for an empty zone is then made; a parameter 2 value of '1' indicates the occurence here as it did for the receiving procedure.

```
IF(OBJ@2,EQ,1,:EMPTY1)
```

One point should be noted here in addition to the remarks made previously about the behavior of the model during either the full or empty zone conditions. As stated before, the pallet or request is simply ignored on the current pass and will still be in the buffer next time around. One might consider simply generating a different zone inside the link when one is found full or empty so that the pallet could be handled during the current pass. The problem with that reasoning is that the entire corridor could be either full or empty, respectively. In that case, an infinite loop would again result. It was decided that the simplest procedure would be to proceed as presented, rather than complicate the logic in favor of a relatively small point.

To reflect that the crane is going to get a pallet for shipping, C is moved to XY(8,49) on the overlay. This is followed by entry into the appropriate link.

```
MA(XY(8,49),0)
LK(!SHIPMOVE)
```

The global variable @OKSHPRQ1 is then incremented to signal okay to the shipping route's request for a pallet.

```
IV(@OKSHPRQ1)
```

In other words, C has had time to bring a pallet up from the corridor to the output buffer, so one of the shipping requests can change from being a request to the pallet which will fill that request; it will then enter the output buffer.

After the move is complete, the values for the corridor zone quantities are printed to reflect the loss of one pallet to satisfy the shipping request.

```
PV(XY(4,39),@ZONE1D)
PV(XY(4,41),@ZONE1C)
PV(XY(4,43),@ZONE1B)
PV(XY(4,45),@ZONE1A)
```

Also, the crane's new postion, which will be (H=1, V=1) after a shipping process, is printed on the overlay.

```
PV(XY(4,32),OBJ@5)
PV(XY(4,34),OBJ@6)
```

Finally, C is moved on the overlay to XY(8,50), the position used to indicate that the crane is at the entrance to the corridor and is waiting for a task to perform.

```
MA(XY(8,50),0)
```

This completes the procedure for shipping, so C next transfers to the check for a received pallet.

```
:EMPTY1    JP(:RECEIVING)
```

Note that this is also the location transferred to in the event that the zone determined for shipping was found to be empty.

As explained previously, the crane job will only produce one object, which will loop through the crane route for the duration of the simulation. This implies that C will never encounter an End Route instruction, which is indeed the case. However, as observed for the synchronization route, one must be included to signal to the loader where the instructions for the crane route stop.

```
ER
```

This then completes the coding for the route of crane 1. The instruction sequences for the other nine routes are exactly the same except for the usual changes of screen positions and variable and label subscripts. It is interesting to note that no relative references are required for the crane routes; this is because the movement instructions are absolute.

Finally, the job statements for the crane jobs can be assembled. As noted at the beginning of this section, the routes for the ten cranes will be numbered consecutively from 11 to 20. Thus, the crane jobs will be numbered identically to provide a 1-to-1 correspondence between job and route. Also as noted above, the jobs will only produce one object each. Lastly, the priority for the objects during the simulation will be 2, as established at the beginning of the coding for the corridors.

```
J=(11,1,11,0,0,2,1)
```

The job statements for the other nine jobs are identical with a change of job and route number.

6. Initialization The last job required for the simulation is the initialization job. As noted previously, it may seem at first that this job should be considered first.

However, with the rest of the simulation built first, the variables required by them can now be included during the building of the initialization route.

The initialization object enters the model at the first second of the simulation. The screen position is chosen arbitrarily to be XY(60,68), as there is no logical display purpose for it.

```
BR(1,(XY(60,68),0)
```

Job number '1' was reserved for use here earlier.

The first task of the route is to initialize the random number sequence. This must be done before any other object tries to obtain a number from the sequence.

```
RS(#SEED)
```

The fact that the job will have the highest priority allows this to be accomplished.

Second, the route initializes the conveyor synchronization variable @GO.

```
SV(@GO,0)
```

This sets it up for use by the receiving and shipping jobs later.

Third, the position of the overlay on the screen is set so as to focus on the center of the overlay action.

```
VW(XY(1,46))
```

The positioning can be adjusted later with the scrolling feature of PCModel.

The initialization object then sets the value of each of the global variables used by the system to zero. For the following subscripted variables, there are ten of each (n = 0, 1, . . . , 9):

```
SV(@INBUFn,0)
SV(@RECRQn,0)
SV(@OKRECRQn,0)
SV(@SHPRQn,0)
SV(@OKSHPRQn,0)
```

These are used in the shipping and receiving routes, as well as by the cranes.

For the corridor zone quantity variables, there are also ten of each.

```
SV(@ZONEnA,0)
SV(@ZONEnB,0)
SV(@ZONEnC,0)
SV(@ZONEnD,0)
```

The purpose of initializing these 90 variables, as well as @GO, may not be clear at first. Certainly, when the model is loaded, their values are set to zero by their symbol definitions. Thus, when the simulation is started after loading, most of the work done by the initialization route is only repeating during run time what was accomplished during load time. The value of having these initializations done during run time becomes obvious when the second running of the simulation is considered. It may be decided after a few minutes of simulation that one of the variable values, perhaps a conveyor speed or buffer capacity, needs to be adjusted before the simulation will provide any meaningful results. PCModel allows the simulation to be reset using its I (Initialize) command. However, this initialization does not reset all of the user-defined variables for the model. To reset them, the value for each one must be edited on the values screen or the model must reload them itself during execution, which is what is being done here.

This completes the work of the initialization route, so its object may now leave the simulation.

```
                    ER
```

Note that none of the instructions specified in the route require any clock time. Thus, the object will complete its route before the first second of the simulation is complete. Further, due to the priority of 0 reserved for the job, it will be done before any other job is considered. Thus, the initializations will be effective for all variables.

The job statement will specify the zero priority and the fact that there is to be only one object produced.

```
          J=(1,X,1,0,0,0,1)
```

The character 'X' is arbitrarily chosen to represent the object during its instantaneous period on the display. As noted earlier, this will be job number 1 and follow route 1.

7. <u>Load-Time Directives</u> At this point, the coding for each of the routes is complete. All that remains to do to have a working PCModel program is to define the load-time directives for the system constructed thus far. It is important to note here again that the directives are implemented during the model loading; they have no effect during the running of the simulation. However, they are vitally important to the system in that they define the storage for variables, the overlay to be used, and other such features. They are presented here for

the automated warehouse in the order suggested by the designer of PCModel. Variations in this ordering may cause unpredictable results.

The first directive to specify is the M, or Maximum objects, directive. The value specified here is used to reserve storage for the maximum number of movement control blocks. This maximum may be determined by assuming that the simulation is completely saturated with received objects and shipping requests. On the conveyor there is room for 38 objects. Each of the corridors have 1 crane, for a total of 10 objects. The corridors each have an input buffer, for which the capacity is initially set at 4; thus 40 objects can wait here. As to the shipping objects, they appear both as requests, at a buffer of 4 to a corridor, and as pallets in the output buffer, here at 2 to a corridor; this yields a maximum of 60 objects. Additionally, a maximum of 10 objects can fit in the receiving area. The only other object is that used for the synchronization job. All of these objects total to 159. 200 spaces were reserved. It is better to allocate too much storage here than too little. In the former case the system will recognize the problem up front; in the latter, the simulation may run for an extended period before the shortage causes a problem.

```
M=(200)
```

The W, or maximum Works in process, directive is directly related to the M directive. W's value at any give time specifies how many objects may be active in the model at that time. It may be altered during the simulation run interactively. The W value specified here is used to initialize the maximum. This feature allows variation in the simulation load. It is set here to the maximum possible to allow the simulation to run under a full load.

```
W=(200)
```

Storage is then reserved for symbols and labels using the S directive. The minimum value for S is the number of variable symbols used. In this simulation, each corridor had 5 different crane-information variables and 4 zone quantity variables, for a total of 90. Additionally, there were 2 variables for arrival rates, 2 for crane speeds, 2 for random numbers, 1 for color, 1 for buffer capacity, 1 for synchronizing conveyor movement, and 5 used by the links, for a total of 14. The total required is then 14 + 90, or 104. A value of 600 is specified here, which will allow room for additions to be made.

```
S=(600)
```

Next come the numbers of screen columns and rows, as given by the X and Y directives, respectively. This defines the size of the overlay. From the data collection process a 100 column by 70 row screen, as shown in Figure 1, will be adequate.

```
X=(100)
Y=(70)
```

Recommended next is the description definition, given by the D directive. The description used for the warehouse simulation is derived from the information gathered in step 2;

it may be seen in the program listing in Appendix 1. Note that the description is limited to 25 lines and is terminated with a '$'.

The source of the overlay is next in the sequence of load-time directives. O defines the overlay file, whether it is inline with the program or is in a separate overlay file, as is the case for this simulation.

```
O=(=)
```

The '=' option specifies that the overlay is kept in a file with the same filename as the model's and an extension of .OLY.

Immediately following the overlay is the V directive for Viewing window location. The XY position given here will be used as the upper-left hand corner of the portion of the overlay displayed during the loading process. More than one could be specified throughout the program to move around the overlay image to focus on areas of interest during loading. As all object movement takes place on one 80x25 section of the overlay in this simulation, the one is all that is utilized.

```
V=(XY(1,46))
```

The next section consists of the symbol definitions. For the warehouse model, the symbols are those that were listed during the calculation for the symbol storage directive. Their initial values at load-time are specified here. For the rates of receiving and shipping:

```
@RECVRATE=(60)
@SHIPRATE=(120)
```

Crane rates horizontally and vertically:

```
@HORZRATE=(12)
@VERTRATE=(120)
```

Random number variables:

```
@RNDMCORR=(0)
@RNDMZONE=(0)
```

Overlay character background color:

```
@COLOR=(7)
```

Input buffer capacity:

```
@BUFCAP=(4)
```

Synchronization variable:

```
@GO=(0)
```

Link temporary variable:

```
@TEMP=(0)
```

Corridor input buffer quantity:

```
@INBUFN=(0)
```

Pallets waiting to be stored:

```
@RECRQN=(0)
```

Stored pallets okay to exit:

```
@OKRECRQN=(0)
```

Shipping requests for pallets:

```
@SHPRQN=(0)
```

Retrieved pallets okay to go to buffer:

```
@OKSHPRQN=(0)
```

Zone quantities used by links:

```
@ZONEA=(0)
@ZONEB=(0)
@ZONEC=(0)
@ZONED=(0)
```

Corridor zone quantities:

```
@ZONEnA=(0)
@ZONEnB=(0)
@ZONEnC=(0)
@ZONEnD=(0)
```

Random number seed:

```
#SEED=(9997)
```

Note that the variable symbols should be placed before any constants (#SEED) or screen pointers, due to the manner in which PCModel allocates storage.

Next, all of the job statements follow in a single group. The derivations for the individual statements were explained at the end of the route for each.

```
J=(1,X,1,0,0,0,1)
J=(2,#,2,0,0,1,1)
J=(3,R,3,0,0,2,5000)
J=(4,S,4,0,0,2,5000)

J=(11,1,11,0,0,2,1)
J=(12,2,12,0,0,2,1)
    .        .
    .        .
    .        .
J=(20,0,20,0,0,2,1)
```

The last of the load-time directives is that for Utilization definitions, the U directive. It specifies XY locations on the screen for which a utilization figure is maintained. The crane routes were arranged so that the crane objects would be on one of two overlay rows whenever they were at work. The 47th row was moved to when the crane was receiving, while the 49th row was used to indicate shipping movement. Specifying the two locations for each crane will yield statistics concerning the crane's busy time.

```
U=(1,REC 1,XY(8,47))
U=(2,SHP 1,XY(8,49))
U=(3,REC 2,XY(15,47))
U=(4,SHP 2,XY(15,49))
    .        .
    .        .
    .        .
U=(19,REC 0,XY(71,47))
U=(20,SHP 0,XY(71,49))
```

Note that the U directive allows a meaningful label to be associated with each position.

This completes the sequence of load-time directives. The rest of the simulation should be organized so that the links are next, followed finally by the routes. See Appendix A for a complete listing of the program.

D. RUN THE SIMULATION

It is at this point that the power and uniqueness of PCModel come into play. For a GPSS program, this step would consist of nothing more than submitting the program for execution and then receiving the tabulated statistics at simulation end. PCModel's approach to simulation allows the user to have a hand in the operation of the system being modeled. As listed earlier, there are a host of interactive capabilities supported.

Several options needed to be decided on at the start of the simulation period. The O (Output) command was used to cause PCModel to automatically save the utilization data generated every 10 hours of simulation time. This created useful data for the warehouse as the two busy positions of each crane have been defined as 20 of the 21 available utilization positions. The F3 key was used to toggle PCModel's execution to the increment mode; as mentioned previously, increment mode steps the system clock one second at a time while the look-ahead mode sets the clock to the time of the next event to occur. The warehouse simulation was run under the incremental mode to obtain as much accuracy as possible in the operation of the model. It was also decided to place the simulation in the halt mode with the F7 key; under halt (as opposed to the normal go mode) the simulation stopped at the end of every 10 hour period and waited for a user entry before resuming simulation. While the O command saved the utilization figures every ten hours without interaction, the halt mode permitted the corresponding overlays to be saved as well. The task of saving overlay screens was accomplished by use of the F (File) command; F saves successive screens to a user specified file.

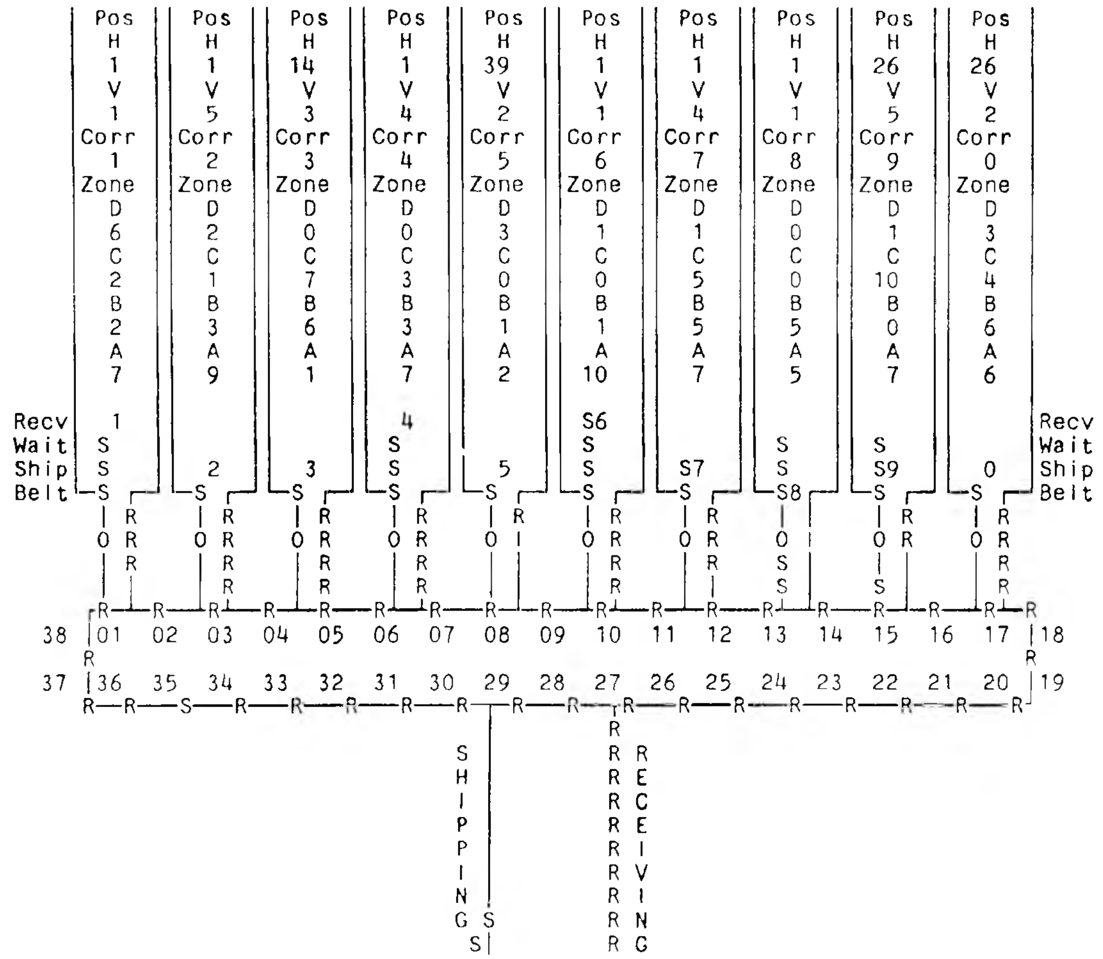
The debugging phase consisted of a set of simulation runs for which various loads were placed on the system. The receiving and shipping rates were varied to insure that the model would behave as anticipated; the crane speeds were also altered. The blocking character facility of PCModel lent itself especially well to the debugging process. Blocking characters were deposited at various positions to verify that the model would react appropriately. For example, blocking characters were used to stop up the corridor output buffers; it was then possible to observe that the cranes did not attempt to fetch pallets for shipping while no space remained in their output buffers. Without the blocking capability, the trial runs would have to have been allowed to run until the output buffers filled up as a result of the load on the system. This could have taken an unreasonable amount of time to occur. Combinations of variable alterations and blocking characters were used to test the program for a wide variety of circumstances.

One run was made for each of the two sets of initial conditions used for the GPSS model. The parameters that vary for the two sets of conditions are the arrival rate for received goods and the arrival rate of requests for pallets to be shipped. These values are stored in the variables @RECVRATE and @SHIPRATE, respectively. For the first case (Run 1), the receiving rate is 2 pallets per min and the shipping rate 1 pallet per minute. This is the peak case wherein the receiving rate is larger than the shipping rate. These values are equivalent to 30 seconds per pallet on receiving and 60 seconds per pallet on shipping. Seconds are required as that is the time unit PCModel works with. Finally, recalling that the entire model was adjusted to work at half the rate of the GPSS model (in order to maintain integer relationships), these rates of seconds per pallet were also doubled. Thus the values used in the program are 60 seconds per pallet on receiving and 120 seconds per pallet on shipping. The second case (Run 2), wherein shipping outpaces receiving, puts receiving at 1 pallet per minute and shipping at 3 pallets per minute. These values are equivalent to respective rates of 60 and 20 seconds per pallet; these values are in turn doubled as above to yield a receiving rate of 120 seconds per pallet and a shipping rate of

40 seconds per pallet. The model was run and statistics generated over a 20 hour period for both sets of initial conditions. The I (Initialize) command allowed the simulation to be reset for the second run without having to reload the source program. The value screen was accessed to edit the receiving and shipping rates. The simulation for the first case above created the overlay of Figure 2 at 10 hours into the simulation and that of Figure 3 20 hours in. The corresponding utilization statistic screens are presented in Tables I and II. For the second case described above, the overlay at 10 hours into the simulation is given in Figure 4 and that at 20 hours in Figure 5. Tables III and IV contain the accompanying utilization statistics. Interpretation of the output will be covered in the next section.

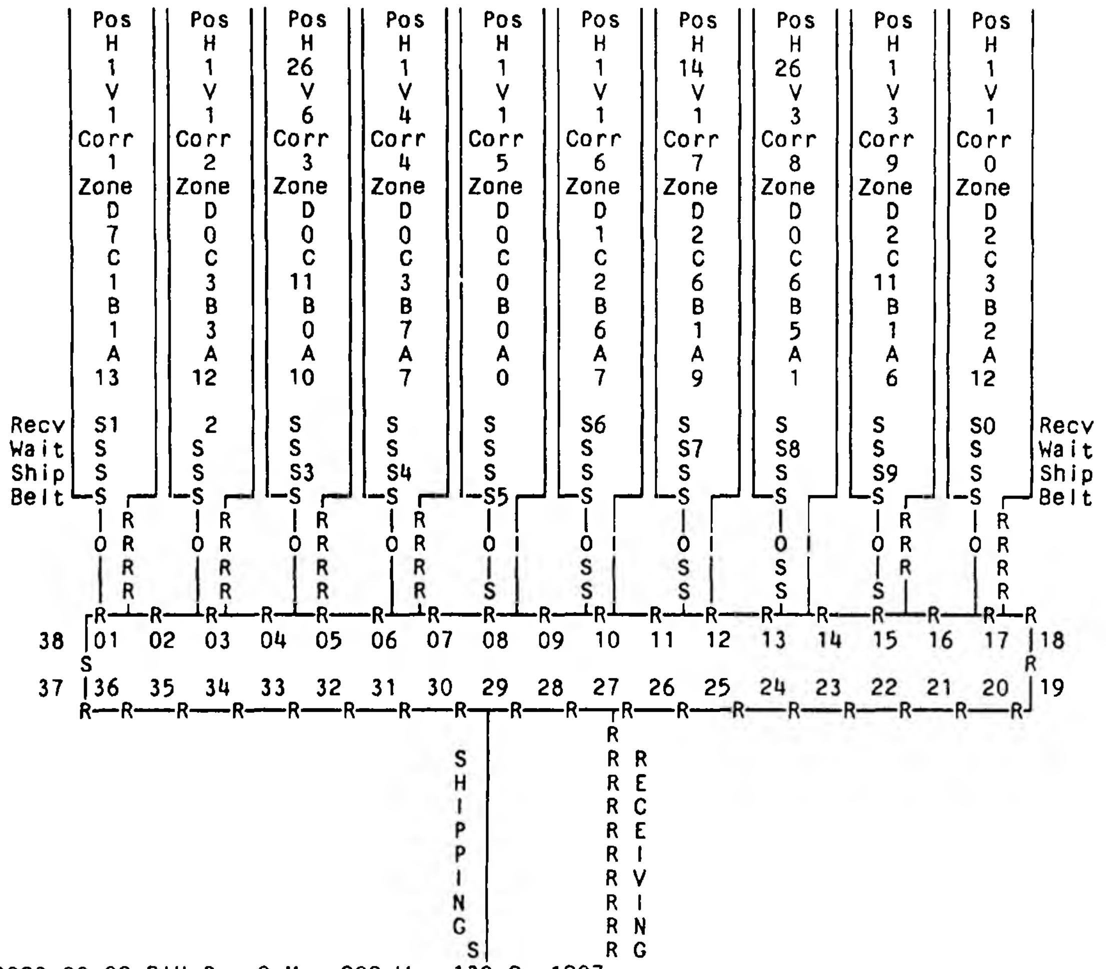
E. EVALUATE THE RESULTS

First, consider the results for the peak input situation (Run 1). It is clear from Figure 2 that after ten hours of simulation the conveyor was totally congested, causing incoming pallets to back up in the receiving bay. Figure 3 indicates that the problem is still evident 10 hours later. The utilization statistics of Tables I and II for the cranes indicate that enough of the cranes were busy all of the time to keep the conveyor belt congested; that is, the cranes never got caught up so that the pallets waiting for them on the belt could get to them. This suggests that the behavior of the system as defined should be examined. Since the problem requires that the system be capable of handling the peak loads given, some


Pos H 1 V 1 Corr 1 Zone D 6 C 2 B 2 A 7
Pos H 1 V 5 Corr 2 Zone D 2 C 1 B 3 A 9
Pos H 14 V 3 Corr 3 Zone D 0 C 7 B 6 A 1
Pos H 1 V 4 Corr 4 Zone D 0 C 3 B 3 A 7
Pos H 39 V 2 Corr 5 Zone D 3 C 0 B 1 A 2
Pos H 1 V 1 Corr 6 Zone D 1 C 0 B 1 A 10
Pos H 1 V 4 Corr 7 Zone D 1 C 5 B 5 A 7
Pos H 1 V 1 Corr 8 Zone D 0 C 0 B 5 A 5
Pos H 26 V 5 Corr 9 Zone D 1 C 10 B 0 A 7
Pos H 26 V 2 Corr 0 Zone D 3 C 4 B 6 A 6
Recv
Wait
Ship
Belt
SHIPPING
RECEIVING

```
0010:00:00 RIH P=   0 M=   200 W=   115 C=   695
```

Figure 2. Warehouse Overlay: Run 1, Hour 10


Pos H 1 V 1 Corr 1 Zone D 7 C 1 B 1 A 13
Pos H 1 V 1 Corr 2 Zone D 0 C 3 B 3 A 12
Pos H 26 V 6 Corr 3 Zone D 0 C 11 B 0 A 10
Pos H 1 V 4 Corr 4 Zone D 0 C 3 B 7 A 7
Pos H 1 V 1 Corr 5 Zone D 0 C 0 B 0 A 0
Pos H 1 V 1 Corr 6 Zone D 1 C 2 B 6 A 7
Pos H 14 V 1 Corr 7 Zone D 2 C 6 B 1 A 9
Pos H 26 V 3 Corr 8 Zone D 0 C 6 B 5 A 1
Pos H 1 V 3 Corr 9 Zone D 2 C 11 B 1 A 6
Pos H 1 V 1 Corr 0 Zone D 2 C 3 B 2 A 12
Recv Wait Ship Belt
SHIPPING
RECEIVING

0020:00:00 RIH P= 0 M= 200 W= 130 C= 1207

Figure 3. Warehouse Overlay: Run 1, Hour 20

TABLE I

WAREHOUSE STATISTICS: RUN 1, HOUR 10

HOURLY UTILIZATION FIGURES

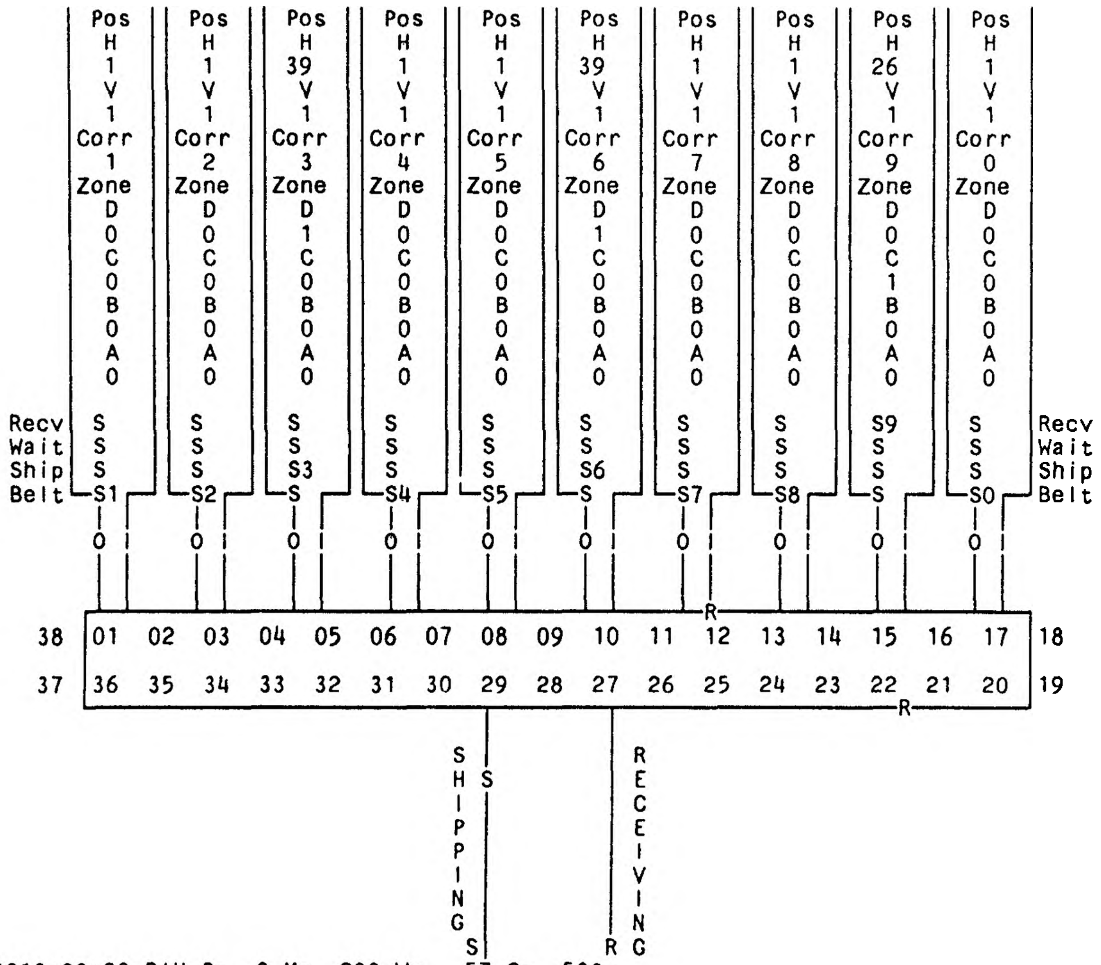
--TOOL--	HOUR1	HOUR2	HOUR3	HOUR4	HOUR5	HOUR6	HOUR7	HOUR8	HOUR9	HOUR10
REC 1	45.00	60.44	57.33	85.00	52.33	44.33	78.22	17.66	44.33	29.77
SHP 1	20.58	39.41	22.00	15.00	47.66	55.66	21.77	82.33	55.66	70.22
REC 2	46.33	28.02	25.66	92.97	51.33	16.69	56.33	45.97	29.33	33.33
SHP 2	12.66	27.66	27.66	07.02	48.66	83.30	43.66	54.02	70.66	66.66
REC 3	21.00	76.25	43.41	48.66	70.11	36.22	68.61	39.05	31.00	52.66
SHP 3	05.91	23.75	56.33	51.30	29.69	63.61	31.38	60.94	69.00	47.33
REC 4	22.86	69.47	66.00	22.27	38.72	34.66	38.33	23.66	28.66	40.27
SHP 4	00.00	17.63	33.97	77.72	61.27	65.33	61.66	76.33	71.33	59.72
REC 5	43.44	64.33	30.55	16.00	26.61	43.50	30.16	10.72	16.00	54.00
SHP 5	12.66	35.66	50.66	80.58	53.41	35.33	61.33	73.00	56.33	33.11
REC 6	65.69	29.66	59.63	46.00	31.33	16.66	32.66	40.00	30.33	37.36
SHP 6	21.33	25.33	32.69	54.00	68.66	83.33	67.33	60.00	69.66	62.63
REC 7	33.33	38.00	82.33	26.02	91.63	44.33	43.69	44.33	51.66	41.30
SHP 7	27.55	47.47	17.66	73.63	08.36	55.66	56.30	55.66	48.33	58.69
REC 8	13.00	41.33	14.33	61.52	34.80	29.52	26.80	51.00	33.00	24.33
SHP 8	00.00	23.66	36.61	05.72	65.19	70.47	66.02	49.00	67.00	66.63
REC 9	17.00	16.36	40.08	49.66	56.66	70.88	60.00	67.44	50.66	56.22
SHP 9	17.00	12.66	17.33	27.00	25.00	29.11	40.00	32.55	49.33	43.77
REC 0	38.00	80.69	69.63	48.69	62.66	50.97	52.33	28.33	36.02	54.63
SHP 0	27.30	19.02	26.69	51.30	37.33	49.02	47.66	71.66	63.97	45.36
not used	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
THRUPUT:	67	85	76	74	73	73	61	60	58	68

```
0009:59:59 RIH P=   0 M=   150 W=   115 C=   695
```

TABLE II

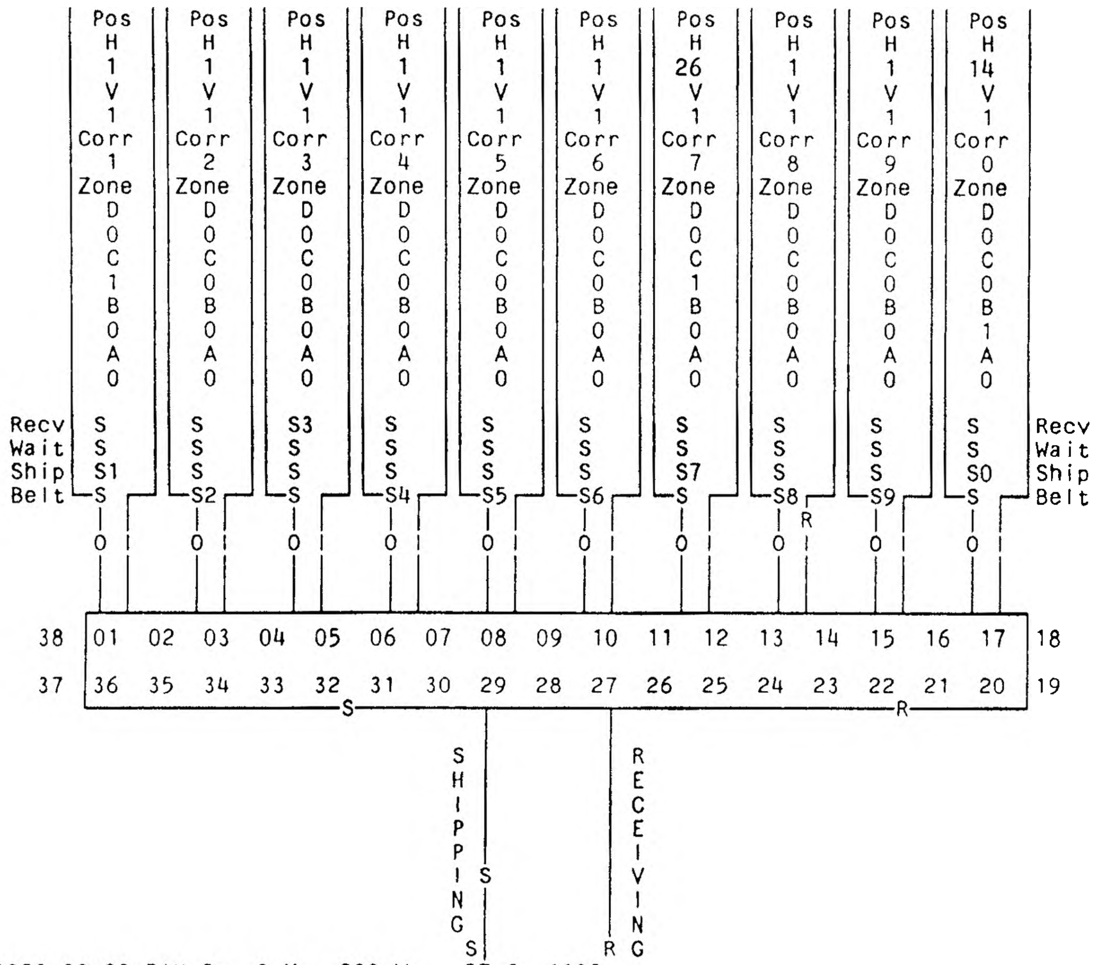
WAREHOUSE STATISTICS: RUN 1, HOUR 20

```
                                    HOURLY UTILIZATION FIGURES
--TOOL--   HOUR1  HOUR2  HOUR3  HOUR4  HOUR5  HOUR6  HOUR7  HOUR8  HOUR9  HOUR10
REC 1      30.55  32.11  68.55  79.44  38.66  33.00  25.22  37.66  53.33  34.44
SHP 1      69.44  67.88  31.44  20.55  61.33  67.00  74.77  62.33  46.66  65.55
REC 2      63.02  80.00  41.63  08.66  28.33  21.86  38.47  43.66  31.66  33.52
SHP 2      36.97  20.00  58.36  75.16  71.66  78.13  61.52  56.33  68.33  66.47
REC 3      50.94  53.38  45.61  59.00  41.38  38.27  80.00  44.38  34.33  46.66
SHP 3      49.05  46.61  54.38  41.00  58.61  61.72  20.00  55.61  65.66  53.33
REC 4      55.00  29.05  38.27  44.66  23.38  27.00  28.27  45.66  52.05  36.00
SHP 4      45.00  70.94  61.72  55.33  76.61  73.00  71.72  54.33  47.94  64.00
REC 5      03.33  07.66  04.33  00.00  01.77  36.22  21.00  04.33  21.33  12.66
SHP 5      25.50  70.91  34.13  00.00  00.00  28.41  34.36  09.22  21.33  12.66
REC 6      51.33  47.30  29.33  21.00  28.00  37.02  56.63  46.33  22.00  04.86
SHP 6      48.66  52.69  70.66  79.00  72.00  62.97  17.66  49.86  37.47  24.33
REC 7      71.36  30.00  28.63  31.66  36.00  16.66  36.66  12.66  03.44  26.88
SHP 7      28.63  70.00  71.36  65.75  62.22  11.00  57.00  25.00  46.00  69.33
REC 8      08.33  34.66  54.00  48.66  18.33  22.66  00.00  11.66  37.66  27.66
SHP 8      51.36  65.33  45.83  47.13  50.33  34.33  10.02  14.97  51.66  30.33
REC 9      38.33  60.00  24.33  45.11  31.88  20.00  35.11  42.33  25.88  15.33
SHP 9      61.66  40.00  75.66  54.88  68.11  80.00  64.88  57.66  74.11  84.66
REC 0      46.36  33.66  30.97  37.00  27.66  52.02  18.30  25.33  34.33  44.02
SHP 0      53.63  66.33  69.02  63.00  72.33  47.97  81.69  74.66  65.66  55.97
not used  100.00 100.00 100.00 100.00 100.00 100.00 100.00 100.00 100.00 100.00
THRUPUT:      56     54     63     51     44     52     48     48     51     45
0019:59:59 RIH P=  0 M=  150 W=  130 C= 1207
```


Pos H 1 V 1 Corr 1 Zone D 0 C 0 B 0 A 0
Pos H 1 V 1 Corr 2 Zone D 0 C 0 B 0 A 0
Pos H 39 V 1 Corr 3 Zone D 1 C 0 B 0 A 0
Pos H 1 V 1 Corr 4 Zone D 0 C 0 B 0 A 0
Pos H 1 V 1 Corr 5 Zone D 0 C 0 B 0 A 0
Pos H 39 V 1 Corr 6 Zone D 1 C 0 B 0 A 0
Pos H 1 V 1 Corr 7 Zone D 0 C 0 B 0 A 0
Pos H 1 V 1 Corr 8 Zone D 0 C 0 B 0 A 0
Pos H 26 V 1 Corr 9 Zone D 0 C 1 B 0 A 0
Pos H 1 V 1 Corr 0 Zone D 0 C 0 B 0 A 0
Recv
Wait
Ship
Belt
S S S S1
S S S S2
S S S3 S
S S S S4
S S S S5
S S S6 S
S S S S7
S S S S8
S9 S S S
S S S S0
R
38 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18
37 36 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21 20 19
SHIPPING
RECEIVING
S
R

```
0010:00:00 RIH P=  0 M=  200 W=   57 C=  588
```

Figure 4. Warehouse Overlay: Run 2, Hour 10



Pos H 1 V 1 Corr 1 Zone D 0 C 1 B 0 A 0
Pos H 1 V 1 Corr 2 Zone D 0 C 0 B 0 A 0
Pos H 1 V 1 Corr 3 Zone D 0 C 0 B 0 A 0
Pos H 1 V 1 Corr 4 Zone D 0 C 0 B 0 A 0
Pos H 1 V 1 Corr 5 Zone D 0 C 0 B 0 A 0
Pos H 1 V 1 Corr 6 Zone D 0 C 0 B 0 A 0
Pos H 26 V 1 Corr 7 Zone D 0 C 1 B 0 A 0
Pos H 1 V 1 Corr 8 Zone D 0 C 0 B 0 A 0
Pos H 1 V 1 Corr 9 Zone D 0 C 0 B 0 A 0
Pos H 14 V 1 Corr 0 Zone D 0 C 0 B 1 A 0
Recv
Wait
Ship
Belt
S1 S2 S3 S4 S5 S6 S7 S8 S9 S0
R
38 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18
37 36 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21 20 19
S
R
SHIPPING
RECEIVING

```
0020:00:00 RIH P=  0 M=  200 W=    57 C= 1189
```

Figure 5. Warehouse Overlay: Run 2, Hour 20

TABLE III

WAREHOUSE STATISTICS: RUN 2, HOUR 10

```
                                        HOURLY UTILIZATION FIGURES
--TOOL--    HOUR1   HOUR2   HOUR3   HOUR4   HOUR5   HOUR6   HOUR7   HOUR8   HOUR9   HOUR10
REC 1       46.02   21.63   04.33   25.33   29.33   38.33   33.66   04.33   04.33   08.66
SHP 1       29.66   38.00   04.33   25.33   28.41   41.66   39.91   04.33   04.33   08.66
REC 2       13.00   08.33   17.00   04.33   33.33   25.33   12.66   17.00   29.66   17.00
SHP 2       13.19   14.80   17.00   04.33   16.16   65.83   12.66   17.00   36.33   17.00
REC 3       08.33   17.00   00.00   16.66   04.33   33.33   08.33   16.66   24.94   17.72
SHP 3       08.33   17.00   00.00   16.66   04.33   40.00   08.33   13.58   24.41   19.55
REC 4       12.52   25.13   20.19   34.66   08.47   08.66   04.33   21.33   04.33   12.66
SHP 4       08.33   26.83   15.16   38.00   12.66   08.66   04.33   20.19   05.47   12.66
REC 5       17.33   20.77   04.55   16.66   00.00   00.00   04.33   21.00   13.00   00.00
SHP 5       17.33   12.66   12.66   16.66   00.00   00.00   04.33   21.00   13.00   00.00
REC 6       13.00   04.33   00.00   04.33   04.33   08.33   17.00   04.33   21.00   29.66
SHP 6       07.69   09.63   00.00   04.33   04.33   08.33   13.69   07.63   21.00   23.02
REC 7       00.00   29.66   16.66   08.33   15.27   10.38   04.33   13.00   21.00   17.00
SHP 7       00.00   45.00   15.94   09.05   12.66   10.27   07.05   13.00   21.00   17.00
REC 8       00.00   00.00   00.00   04.33   17.00   08.33   00.00   00.00   00.00   00.00
SHP 8       00.00   00.00   00.00   04.33   17.00   08.33   00.00   00.00   00.00   00.00
REC 9       00.00   04.33   12.66   00.00   08.33   34.00   08.66   00.00   04.33   10.44
SHP 9       00.00   04.33   12.66   00.00   08.33   20.41   30.25   00.00   04.33   00.00
REC 0       12.66   17.00   34.02   25.30   29.66   12.66   63.33   17.00   04.33   00.00
SHP 0       08.25   21.41   29.66   29.66   27.36   12.58   29.72   69.36   09.97   00.00
not used   100.00  100.00  100.00  100.00  100.00  100.00  100.00  100.00  100.00  100.00
THRUPUT:       44      66      60      62      52      62      59      62      63      58
0009:59:59 RIH P=   0 M=   150 W=    57 C=   588
```

TABLE IV

WAREHOUSE STATISTICS: RUN 2, HOUR 20

```
                              HOURLY UTILIZATION FIGURES
--TOOL--   HOUR1  HOUR2  HOUR3  HOUR4  HOUR5  HOUR6  HOUR7  HOUR8  HOUR9  HOUR10
REC 1      25.33  12.66  04.33  24.77  04.88  04.33  00.00  00.00  08.33  46.66
SHP 1      07.11  30.00  11.88  21.00  08.66  04.33  00.00  00.00  08.33  34.36
REC 2      08.66  37.66  17.00  00.00  25.33  15.69  20.97  31.00  08.33  08.33
SHP 2      08.66  38.08  25.25  00.00  25.33  12.66  21.00  34.00  08.33  08.33
REC 3      12.66  08.94  12.38  24.94  13.05  00.00  00.00  25.66  12.66  20.94
SHP 3      14.44  08.66  12.66  21.33  16.66  00.00  00.00  49.00  12.66  12.66
REC 4      08.66  08.33  17.00  04.33  12.52  29.80  32.86  26.13  00.00  08.66
SHP 4      08.66  08.33  17.00  04.33  08.33  40.66  08.66  57.00  00.00  08.66
REC 5      04.33  25.66  25.33  04.33  12.66  00.00  38.00  30.77  15.55  08.33
SHP 5      04.33  32.33  25.33  04.33  12.66  00.00  25.44  65.22  17.00  08.33
REC 6      42.66  04.33  21.36  16.63  04.33  14.69  06.63  04.33  29.33  08.66
SHP 6      33.30  27.00  12.66  33.33  04.33  12.66  08.66  04.33  22.52  15.47
REC 7      00.00  29.66  04.33  17.00  42.33  29.66  04.33  16.66  12.61  25.38
SHP 7      00.00  09.61  39.72  17.00  38.75  33.25  01.61  19.38  00.00  30.52
REC 8      00.00  08.66  04.33  08.33  08.66  04.33  12.66  38.00  54.77  08.55
SHP 8      00.00  08.66  04.33  08.33  08.66  04.33  12.66  27.16  45.16  57.00
REC 9      14.55  13.11  03.88  00.00  29.66  29.33  04.33  04.33  12.44  13.22
SHP 9      25.00  12.66  04.33  00.00  12.69  43.41  07.22  04.33  08.66  17.00
REC 0      40.69  01.63  04.33  08.33  00.00  13.00  00.00  00.00  00.00  04.33
SHP 0      29.66  12.66  04.33  08.33  00.00  13.00  00.00  00.00  00.00  00.02
not used  100.00 100.00 100.00 100.00 100.00 100.00 100.00 100.00 100.00 100.00
THRUPUT:      58     58     66     59     60     60     56     62     59     63
0019:59:59 RIH P=   0 M=   150 W=    57 C= 1189
```

characteristic of the model must be altered. Examples of things to consider are the crane speeds and conveyor pace. Certainly if the cranes moved faster the system would not be as congested; this is suggested by the fact that in both Figures 2 and 3 fully half of the corridor input buffers are full of pallets waiting on cranes to handle them. The conveyor pace might also be increased; this would allow the pallets to be shipped that do make it to the conveyor to be cleared off more quickly. This in turn would make room for more of the received pallets, which might be headed for some of the not-so-busy cranes. Alternatively, the crane behavior might be changed to decide whether to store an incoming pallet or place it in the output buffer if a shipping request exists. As it stands now, the cranes always store received pallets regardless of the presence of shipping requests for the corridor.

For the peak output situation (Run 2), the system ran smoothly. As would be expected with the number of shipping requests exceeding the number of incoming pallets, the number of pallets in the bins of the corridors is practically zero. Shipping requests are of course backed up, but that is only to be expected. If it were decided to discard requests that could not be filled, this problem would be eliminated as well. Figures 4 and 5 show the status of the system at the 10 hour and 20 hour points. The zone quantities and waiting states of the cranes are clear from them.

The GPSS model did not experience the backing up of pallets as the PCModel simulation did for the first case.

Subsequent runs of the PCModel program with higher crane rates of movement eased this problem somewhat. Both the GPSS and PCModel programs are fairly involved, which allows some room for variance. In the next section, a much simpler GPSS model is examined, thus allowing a closer comparison with PCModel's results.

IV. THE SUPERMARKET PROBLEM

In the previous section, the simulation of an automated warehouse was dealt with. For this problem, the floor plan of the warehouse was instrumental in determining the solution of the problem; the overlay directly reflected areas where objects were backed up. This relationship of the physical layout of the warehouse to the information gathered from the overlay of the running model is due to the inherent first-in, first-out (FIFO) ordering of the problem. This is the natural way that goods and shipping requests would be handled. This is also inherent in the design of PCModel. As there are objects moving about the screen, they must move in a follow-the-leader fashion; there is no way for objects to move at different speeds along the same route without interfering with one another's progress. Behavior of this different type is easily modeled with GPSS's ADVANCE block, as has been noted. This section examines how such a problem can be dealt with using PCModel and also looks at how statistics might be collected for its simulation.

A. DEFINE AND LIMIT THE PROBLEM

A good example of the type of problem to be considered here can be found in the simulation of a supermarket. Shoppers will enter such a store and shop for a period of time that is dependent upon circumstances outside the realm of the model, such as the size of the shopper's family, the interval between shopping trips for the family, and so on. At any rate, it is impossible to model the shoppers in a FIFO fashion as is typical of PCModel. In this section and the next the behavior of such a system and its numeric parameters will be defined; the section on coding the problem will then examine ways to model the set up. The text GPSS V: AN INTRODUCTION$_{14}$ defines exactly such a problem; using the same parameters for a PCModel simulation will allow comparison of the results obtained under the two languages.

Behavior of each customer upon entering the store begins with an attempt to obtain a shopping cart. It is assumed that the customer will wait until a cart is available before beginning shopping (i.e. each shopper will need a cart) and that no customer will leave the store due to the required wait for a cart. Once a cart is obtained, the customer shops for an amount of time which is determined in two parts. First, a certain percentage of the customers entering the store are assumed to be there for general shopping needs; these regular customers will all take a minimum amount of time for shopping. The remaining customers will have entered the store just to pick up a limited number of items; these are the express customers, none of which will require over a given maximum amount of time. This breakdown of customers sorts them into two categories. The other part of determining the shopping time follows from the assumption that one average time is required for express shoppers and another for regular shoppers. All remaining details concerning these times will be stipulated in the data collection process.

The remaining part of customer behavior is concerned with the checkout process. For purposes of this model, there will be two registers available for checkout. One will be available for express checkouts only, with the other reserved for regular customers. To simplify matters, it is assumed that regular customers will use only the regular checkout line, and the same will apply for express shoppers. This will hold true even if customers must wait in the line for one checkout register while the other line is empty. Once the checkout process has been completed for a customer, his cart is assumed to be free for use immediately. The customer then leaves the store, bag in arms.

B. COLLECT DATA

1. Data to Build the Model With For the supermarket as defined, there are five general entities about which to gather information: express shoppers, regular shoppers, the express checkout counter, the regular checkout counter, and the carts present in the store. Each of these will be examined in turn in this section.

One customer, either express or regular, will be entering the store on the average of every four minutes; the distribution of interarrival times is uniform and discrete, varying from 0 to 8 minutes. The percentage of customers entering the store as express shoppers will be 30%, leaving 70% as regular shoppers. Once an express customer has obtained a shopping cart, he will require from 1 to 11 (6 +/- 5) minutes in the store; a regular shopper will take from 15 to 45 (30 +/- 15) minutes to select his items. These times are according to a uniform discrete distribution. At the end of the shopping time, both types of customers enter the lines for their respective checkout counters.

The checkout for express shoppers will take from 1 to 3 (2 +/- 1) minutes once the customer reaches the counter. In the regular case the process takes from 3 to 7 (5 +/- 2) minutes. Again, these times are according to a uniform discrete distribution. The remaining model entity is that of the carts. The store will have 20 of them available for use. Each customer will take one as soon as one is available and keep it throughout the shopping interval, time in the checkout line, and checkout period. Once checkout has been completed, the cart will be assumed ready for use by another customer. This concludes the parameters required for the model.

It should be noted here that none of the above data in any way details the layout of the store. In fact, this absence indicates that the design of the store should in no way influence the statistics generated. This is in direct contrast to the automated warehouse problem discussed earlier. For the warehouse a good part of the data concerning the simulation detailed the distances from one point to another, which in turn helped define how much time would be necessary for movement from one position to another. The dimensional information directly defined the overlay to be used with the model. Here, it is just the opposite; it does not matter how many screen units a customer moves, so long as the various times are preserved. This reasoning permits great flexibility in the overlay, allowing it to be created in such a fashion as to facilitate the collection of statistics.

2. Data Desired as Output The purpose of this step thus far has been the collection of data for the system to be modeled. Now, consider for a moment the collection of data to be output for the model. The automated warehouse output consisted in large part of the action taking place on the overlay. This was necessitated by the complexity of the problem itself. Here the coding of the problem solution will be somewhat simpler, so more emphasis can be placed on the collection of data. The GPSS model is particularly concerned with statistics concerning the carts (a GPSS 'storage'), the checkout and cart waiting lines ('queues'), and the checkout counters ('facilities'). While the model itself is being constructed, additional software will be incorporated with the generation of statistics concerning these GPSS entities in mind.

C. SOLVING THE PROBLEM IN SOFTWARE

1. Job and Route Definitions Now comes the process by which the problem definition and data are used to define the jobs and routes of the PCModel program; once complete, the coding process itself can begin. As will be the case for almost any simulation program, a job and route will need to be reserved for initialization purposes. As noted in the warehouse problem, the coding of this job (job number 1) is best left until last. Now, for the behavior of the model proper, it appears that a single job and associated route for

the customers will be sufficient. The carts can be modeled using a global variable to keep track of the number in use; certainly an entire job and route are not warranted. For the checkout counters, there is an argument for modeling them as separate jobs, as was done for the cranes in the warehouse problem. This would necessitate a similar handshaking pattern between customer and checkout routes, but on the other hand it would allow the checkout counters to have several different modes as their objects could move to different positions to indicate different statuses. This is what was done with the crane jobs. However, examination of the checkout counter behavior indicates that a counter is either occupied or unoccupied by a customer. These two states can easily be represented by the presence of a customer object at the checkout position on the screen. Further, statistics may be gathered concerning this overlay position using the U= directive and the utilization screen to determine how much of the time it is occupied, and thus how much of the time the counter is in use. With these facts in mind, it seems simplest to just incorporate the checkout counters as part of the customer route. Thus one job and route will be sufficient to model the supermarket system itself.

In addition to the prior two jobs, one more will be needed for the purpose of collecting statistics. Consider the statistics generated by a GPSS simulation. These include maximum and average contents for queues and storages, number of entries for queues, storages, and facilities, as well as average time per customer in each, to name a few. Clearly, the limited arithmetic capabilities of PCModel would make it extremely difficult to generate such information exactly. For instance, the average contents of a storage would logically be found by maintaining a running total consisting of the number of customers in the storage for each clock interval and then dividing that total by the number of clock intervals that have passed. The update of displayed statistics could be done at any given interval down to each clock interval. PCModel's variable value limitation of 65,535 makes the maintenance of such a running total unfeasible by itself; for instance, if on average 10 of the carts are being used each second, in 6554 seconds (less than 2 hours) the variable for the running total would overflow. Even if this were not a problem, PCModel's division is limited to truncated integer results, which would not be very precise. One solution to the problem of variable overflow is to only add to the running total at regular intervals, each of which consists of a fixed number of clock intervals. Thus the inevitable overflow is delayed to a reasonable time for simulation purposes. A separate job can thus be used to generate an object at these regular intervals for the purpose of collecting variable information. The other associated problems, such as the limitation to integer division, will be examined during the creation of the route. As this job is not directly concerned with the overlay simulation itself, it is designated job 2. The job for customers then becomes job 3. This is a programming convention which puts the actual simulation routings last in the program.

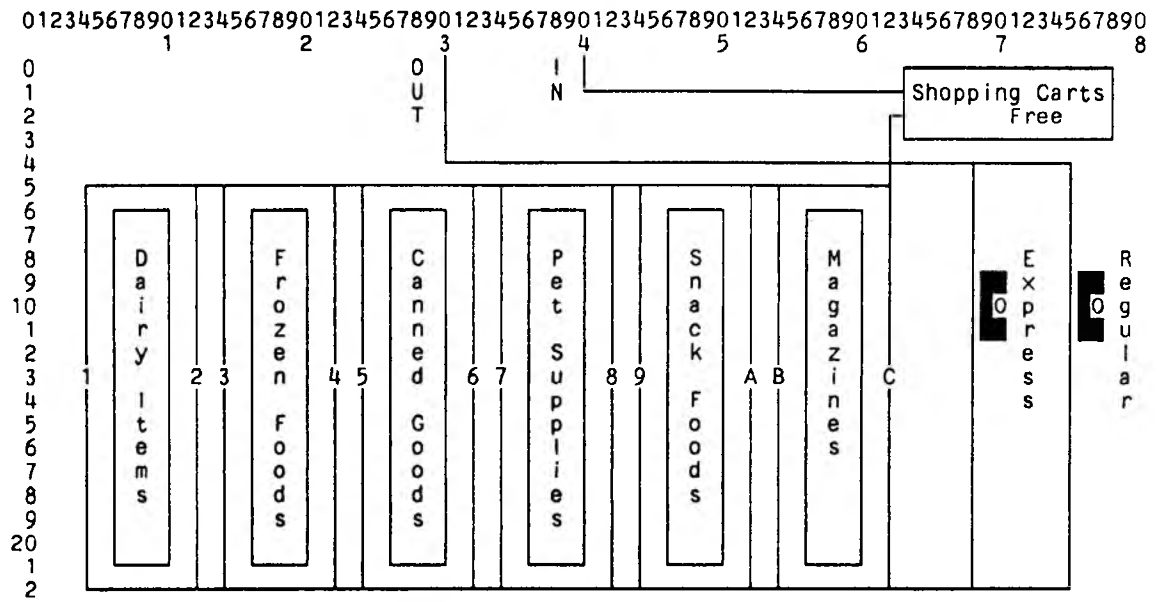
The focus of attention is now turned to the coding of the route for the customers. (It should be noted here that coding will not be as straight forward as possible in places so as to employ miscellaneous PCModel instructions that would otherwise not be included.) As observed previously, the particulars of the overlay are left to the descretion of the programmer. This allows the freedom necessary to solve the problem using PCModel. As the customers will be in the store for different periods of time, it is clearly impossible to have all shoppers follow the same path around the screen. Clearly, express shoppers will be held up by the regular shoppers simply because the program logic would not make it possible for customers to pass one another in the aisles. Indeed, there is no straight forward way to incorporate this behavior in the model, due in part to the restriction to constant move distances with PCModel. Separating the express and regular shoppers will not entirely solve the problem either. For instance, even though regular shoppers will take on the average thirty minutes each, there is nothing that says two successive customers entering the store will not both be regular shoppers, and further that the second of the pair will be in the store for a much shorter time than the first. The whole problem can be solved by dividing the store up into a number of aisles, say twelve. Reserving three for express shoppers and the remaining nine for regular shoppers solves the problem of separating the two widely separated (time-wise) groups. Additionally, within the aisles reserved for one type of shopper, each successive shopper can be sent to a currently unoccupied aisle. This separates the individuals from one another. Inside the aisle shoppers can move at different speeds, while they will all move at the same pace on common ground. The transferring process will be examined in detail shortly. The supermarket overlay as decided upon with its twelve aisles can be seen in Figure 6.

2. Customers Now coding can begin for the customer route; the various movements will be defined by the selected overlay. First, as noted in the data collection process, customers will arrive according to a uniform discrete distribution from 0 to 480 seconds (0 to 8 minutes). The mean is 240 seconds (4 minutes), and the variable @ARRIVE will have been initialized to it. Further, the arrival of the next customer can be determined using PCModel's random number sequence.

```
BR(3,*ENTRY,@ARRIVE)
RV(@ARRIVE,0,480)
```

As remarked earlier, the customer job is numbered 3, and thus the customer route is numbered likewise. The entry position into the store, *ENTRY, is defined in the program listing in Appendix B.

Now the customer object, hereafter referred to as 'C', moves into the store at an arbitrary speed to the area reserved for the cart waiting line. Here the total number of entries into the cart queue, @TTLCRTQ, and the current number of entries in the cart queue, @CRTQUEUE, are updated.



```
                          Supermarket Simulation

 Number of Counts Taken:              Time of Last Update:

          Storages
  Storage   Capacity    Average    Entries    Total Time    Current    Maximum
                        Contents              in Storage    Contents   Contents
  Carts



          Queues
  Queue     Maximum     Average    Total     Total Time    Current
            Contents    Contents   Entries   in Queue      Contents
  RegCk
  ExpCk
  Cart



       Statistics are updated every four minutes of simulation time.
```

Figure 6. Supermarket Overlay

```
MD(1,5)
IV(@TTLCRTQ)
IV(@CRTQUEUE)
```

Additionally, C's first clock parameter is set to the current time. This will be used shortly in the determination of C's time in the cart queue. This save is followed by movement of C through the cart queue to its end.

```
SV(OBJ%1,CLOCK)
MR(22,5)
```

Note the use of the system variable CLOCK and the fact that since C's movement is causing a delay itself, C will take some time (110 seconds) in the queue even if its progression to the queue end is not blocked by C's waiting for carts.

Now is a good time to update the maximum number in the cart queue, @MAXCRTQ. Note that this is accomplished with instructions which require no clock time themselves.

```
IF(@CRTQUEUE,LT,@MAXCRTQ,THEN,:PASS)
SV(@MAXCRTQ,@CRTQUEUE)
```

Once C reaches the end of the cart queue, it can leave the queue if there is a cart available. A check is made against the current number of carts available, @NUMCARTS, to see if one is available with the IF statement. Note the use of the IF keywords THEN and NEXT denoting transferrence to the next sequential instruction if a cart is available, while ELSE and WAIT hold C at this point, trying the IF condition on each successive clock interval until it is satisfied before letting C move on.

```
:PASS      IF(@NUMCARTS,GT,0,THEN,NEXT,ELSE,WAIT)
```

Once a cart is determined available, C leaves the cart queue and decrements the number in the queue accordingly, as well as updating the total of the times spent in the cart queue, %TTLCRTQT.

```
DV(@CRTQUEUE)
SV(%DIFFRNCE,CLOCK)
AO(%DIFFRNCE,-,OBJ%1)
AO(DIFFRNCE,-,110)
AO(%TTLCRTQT,+,%DIFFRNCE)
```

Several things should be noted here. The first is that the elapsed time is logically the difference in the clock times upon entering and exiting the queue, less the required time in the queue. At both points CLOCK is copied so that is may be worked with. Secondly, a global variable %DIFFRNCE is used here for obtaining the difference in clock times, rather than C's second clock parameter which is currently unoccupied. Trial simulation runs indicated that clock parameters can not be specified as the destination of an arithmetic operation. Doing so causes unpredictable results. Finally, it should be observed that clock variables can be operated on with the addition and subtraction operators in the same manner as standard variables.

At the same time C leaves the cart queue it enters the cart storage. A storage as defined by GPSS is an entity with multiple unit capacity. For the PCModel simulation, the single global variable @CRTINUSE will suffice to keep track of the number of carts currently in use. Upon entry into the cart storage, the total number of storage entries made, @TTLCRTSU, and the current number of objects in the storage can be updated; the second object clock parameter is also set here for later determination of the time in the storage.

```
                    IV(TTLCRTSU)
                    IV(@CRTINUSE)
                    SV(OBJ%2,CLOCK)
```

Further, the maximum number of carts ever in use at any one time, @MAXCRTSU, can be updated here if necessary.

```
                    IF(@CRTINUSE,LE,@MAXCRTSU,THEN,:OVER)
                    SV(@MAXCRTSU,@CRTINUSE)
```

Thus, C has now left the waiting line for a cart and is ready to start shopping. This is reflected on the screen by moving the customer to the position at which to take a cart, *CARTS, and decrementing the number of carts available. The number of carts left is then displayed on the overlay.

```
             :OVER  MA(*CARTS,0)
                    DV(@NUMCARTS)
                    PV(*CRTSFREE,@NUMCARTS)
```

Next, C moves to the position at which it will branch once it is determined what type of shopping will be done (express or regular) and which aisles are open for that type. At that position a random value is generated for the differentiation between express and regular shopping times.

```
                    MD(3,1)
                    RV(@CHANCE,1,100)
```

Up to this point every C has been treated identically. Now comes the first transfer based on the results of the random value generated.

```
                    IF(@CHANCE,LT,#PERCENT,THEN,:EXPLOOP)
```

The constant #PERCENT will thus be defined as the percentage of express customers. The data collection process indicated this to be 30%. C's fall through the IF to :REGLOOP if the condition is not met.

Now consider the route taken by regular customers. As discussed earlier and consequently incorporated into the overlay of Figure 6, the C's will be kept from interfering with one another by sending them successively to unoccupied aisles. More specifically, each aisle will be considered occupied if the position at the head of the aisle is occupied. While moving to the aisle, moving down it, and then moving to the checkout line, every regular customer will travel at the same rate over the corresponding areas. The occupation of the single position at the head of the aisle is the only instance where varying times of delay will be employed; the varying times themselves will be calculated based upon the shopping time computed for a C. This chain of reasoning can employ PCModel's Post (PO) and Clear (CL) instructions to good advantage. While the C is at the branching position, *READY, tests can be performed to find an aisle for which the head position is clear; then, before C even starts moving toward that aisle (a process which will require measurable time to mean anything on the overlay), the position can be reserved by posting it until C gets there. Thus no subsequent C will start out for the same aisle while the first C is moving to it.

Thus, the logic for routing C's is defined. Before it is actually employed, the C's need to branch to the routing for the particular lane. The lane is decided upon by utilizing the Jump if path Clear instruction to determine the first lane available for posting.

```
:REGLOOP  JC(1,*LANE1,:SHPLANE1)
          JC(1,*LANE2,:SHPLANE2)
              .          .
              .          .
              .          .
          JC(1,*LANE9,:SHPLANE9)
          DN
          JP(:REGLOOP)
```

Note that in the event all nine of the allocated lanes are either posted or occupied, C will pause and check again on the next clock interval. Logically there should never be a delay here because the software construct above is artificial and does not model the actual system; rather, it is being used to solve the problem of interference among objects. If the nine lanes prove insufficient in handling C's without delay, more lanes should be allocated in order to keep from distorting the simulation results. One simple way to monitor the use of the various posting locations is to define them as utilization locations. This will be explored further later.

Once a clear lane has been found, C can move to it to perform the shopping task. In the event that the reserve position for lane 1, *LANE1, is currently clear, transfer will occur to the label :SHPLANE1 to "shop" in lane 1. The first thing to do is establish a reference for the subsequent relative moves and post the reserve location.

```
:SHPLANE1 R=(*READY)
          PO(*LANE1)
```

Now C can move over to the lane or aisle. An arbitrary rate of 5 seconds per screen position will be used for the typical movement of a regular shopper. Any time used while moving around the screen will be subtracted off the total shopping time determined for the customer. Note in Figure 6 however that regular and express shoppers share a common screen route for the first ten spaces to the left of *READY. With the screen laid out as it is, it is impossible for express customers to also move at 5 seconds per position because of their much shorter shopping durations. A rate of 1 second per position will be needed to move a C through the store in under the minimum time for an express shopper. This is why the "express" lanes (A, B, and C) are located so close to both the cart storage area and the express checkout counter. If there were any more separation, it would not be possible for an express customer to traverse the required route in less than the minimum time without utilizing some immediate moves (no delay imposed). All of this will be seen in more detail in the routing of the express lanes, but it needs to be brought out here to explain the movement of the regular shopper. For any route throughout the store that is used by both regular and express shoppers, the regular shopper must move at the higher rate of the express shopper so as not to form a possible bottleneck for them. This explains the pair of moves for a regular shopper moving to lane 1.

```
ML(10,1)
ML(48,5)
```

Once at the head of the lane, the reserve position may be cleared so that the C itself can then occupy it.

```
CL(*LANE1)
MD(1,5)
```

Now C is at the position where any time other than that used up moving around the overlay must be consumed. The shopping time itself is determined as a random number from 900 to 2700 seconds (30 +/- 15 minutes).

```
RV(@SHOPTIME,900,2700)
```

From this time must then be subtracted all of the times expended from the cart cage to the checkout counter. These times include: 10 spaces at 1 second per and 48 seconds at 5 seconds per getting to the lane (250 seconds); 1 space in 5 seconds moving to the reserve position (5 seconds); 15 spaces at 10 seconds per moving down the lane (150 seconds); 1 space in 5 seconds moving to the back wall (5 seconds); and finally 40 spaces at 5 seconds per moving along the back wall to the position common for all nine lanes of regular shoppers (200 seconds). These various times total to 610 seconds and constitute the movement time from the position beneath the cart cage to the common position for regular shoppers at the back wall behind the ninth aisle. Outside of this are times common to all regular shoppers: 3 spaces at 1 second per moving from the cart cage to the branching position (3 seconds); 6 spaces at 8 seconds per and 1 space at 9 seconds (to balance the timing) moving to the point common with express customers moving toward checkout (57 seconds); 18 spaces right at 1 second per through the express-common route (18 seconds); 6 more spaces at 9 seconds each moving to the aisle of the regular checkout (54 seconds); and finally a move of 12 spaces at 9 seconds each moving to the checkout counter (108 seconds). The times here total to 240 seconds. Note that this figure will be common to all nine lanes, while the sum of 610 seconds is peculiar to lane 1 only. For each of the other eight, a different value must be determined, due to the varying distances of the lanes from the common starting and end points. All of the moves described here will be seen in the routing shortly. Also it should be mentioned once again that the combination of times used is arbitrary so long as they balance out. At present, the constant delays for lane 1 are subtracted from the total shopping time and then C is delayed for the remaining shopping time.

```
AO(@SHOPTIME,-,610)
AO(@SHOPTIME,-,240)
ST(@SHOPTIME)
```

Observe that the sum of the required times for a customer sent to lane 1 is 850 seconds, 50 less than the minimum shopping time of 900 seconds for a regular customer.

When the time has elapsed, C moves down the aisle, to the back wall, right to the common point for regular shoppers, all as described above, and then transfers to the routing common for the shoppers at that point.

```
MD(15,10)
MD(1,5)
MR(40,5)
JP(:REGCHK)
```

As for the other eight lanes, the routings for them are parallel to that of lane 1. The differences consist of changing labels for the lanes, substituting in correct move distances to and from the lanes, and altering the shopping time constant peculiar to the lane. All of these values change in a regular pattern from aisle to aisle as the overlay is laid out in a regular pattern itself. The distances to lanes 2 through 8 are 50, 48, 40, 38, 30, 28, 20, and 18 spaces respectively; the distances from the lanes to the common point *REGLINE are 32, 30, 22, 20, 12, 10, 2, and 0 spaces. Finally, the corresponding times for each lane are 530, 510, 430, 410, 330, 310, 230, and 210 seconds; these values can be obtained in the same manner as the 610 second figure for lane 1.

One other point should be made again here before preceding on. The seemingly arbitrary rates and distances for the movements discussed in the time calculations are just that: arbitrary. Again, the purpose of the route is to create some displayable image. Any combination of movements and rates could be used so long as they do not interfere with the simulation logic of a customer or reflect on the display a misinterpretation of what is really transpiring.

At this point, the behavior for each regular customer through the store has been mapped to the common point *REGLINE, where they all meet on their way to the regular checkout. The route continues here as they enter the queue, or waiting line, for the regular checkout counter. Entry into this queue signals the time at which to update both the total number of entries into the regular queue, @TTLREGQ, and the current number in the queue, @REGQUEUE, as well as setting C's first clock parameter to the current simulation time.

```
:REGCHK   R=(*REGLINE)
          IV(@TTLREGQ)
          IV(@REGQUEUE)
          SV(OBJ%1,CLOCK)
```

OBJ%1 will be used shortly to determine the time in the queue for C; remember also that OBJ%2 still contains the entry time for C into the cart storage.

Additionally, the maximum contents for the queue, @MAXREGQ, can be revised at this point.

```
          IF(@REGQUEUE,LT,@MAXREGQ,THEN,:DOWN1)
          SV(@MAXREGQ,@REGQUEUE)
```

Now C moves around the screen to the checkout counter as described previously.

```
:DOWN1    MR(6,8)
          MR(1,9)
          MR(18,1)
          MR(6,9)
          MU(12,9)
```

Again note the increase in speed through the route common with the express customers.

When C completes the move up of 12 units indicated above, it is at the checkout counter and thus ready to exit the waiting line, or queue, for checkout. To accomplish this, the number in the queue, @REGQUEUE, is decremented and the elapsed time of C in the queue, %DIFFRNCE, is added to the running total for times in the queue, %TTLREGQT.

```
          DV(@REGQUEUE)
          SV(%DIFFRNCE,CLOCK)
          AO(%DIFFRNCE,-,OBJ%1)
          AO(%DIFFRNCE,-,237)
          AO(%TTLREGQT,+,%DIFFRNCE)
```

In this case the elapsed times for C's in the queue will consist of the fixed move delays above, which total to 237 seconds, as well as any delays caused by C's backing up at the checkout. The figure of 237 seconds is subtracted out of the elapsed time so that the obtained average may be compared with 0 instead of 237.

Once at the register, the checkout time, @REGCHK, is determined randomly in the range of 180 to 420 seconds (5 +/- 2 minutes); C is then held for the period of checkout.

```
RV(@REGCHK,180,420)
ST(@REGCHK)
```

When the checkout procedure is finished, the customer has no more use for his cart; thus it may be returned to the cart storage. C leaves the storage by updating the number of carts in use, @CRTINUSE, and adding the elapsed time in the storage, %DIFFRNCE, to the running total, %TTLCRTST.

```
DV(@CRTINUSE)
SV(%DIFFRNCE,CLOCK)
AO(%DIFFRNCE,-,OBJ%2)
AO(%TTLCRTST,+,%DIFFRNCE)
```

Additionally, the number of carts free for use is updated and displayed in the cart storage area.

```
IV(@NUMCARTS)
PV(*CRTSFREE,@NUMCARTS)
```

Note that the variables @CRTINUSE and @NUMCARTS are simply complements of one another. @CRTINUSE is sufficient to hold the necessary information, but an extra variable would be required for determining the number free for output anyway; thus it is maintained as a separate value.

At this point, C has effectively completed the simulation and could be removed immediately. However, in order to reflect on the overlay that the customer would now leave the store, C is routed across the overlay to the screen position which is common to all departing C's, *EXIT. A transfer then takes place to the common routing all C's take upon exiting.

```
           MU(6,5)
           ML(7,5)
           JP(:EXITSTOR)

:EXITSTOR  R=(*EXIT)
           ML(5,5)
           RM(-2,+0,5)
           ML(31,5)
           MU(4,5)
```

Observe the use of the Relative Move (RM) instruction to cause C to jump over the path crossing that of incoming customers. Using RM saves the determination of the exact screen locations required by the Move Absolute instruction. It would be convenient if these two paths did not have to cross at all. The problem here relates to that explained for the express customers. They must be able to move from the cart cage, down their aisle, and then back to their checkout before the minimum time expires; thus the overlapping routes.

One final inclusion is the use of a termination count. The GPSS language provides for specification of a value at which to stop the simulation and display statistics. PCModel can do the same thing by keeping a running count of objects that have completed the customer route. C leaves the simulation after this update.

```
IV(@TERMCNT)
ER
```

The count might better be taken as C's leave the register, for that is the point the simulation is done with them. Placing it here makes it a duplicate of PCModel's Work Complete Count (WCC) and should not create any appreciable distortion of statistics gathered. After initialization, the initialization job object can check for the occurrence of this variable reaching the specified value using an IF statement. Further, the simulation need only be halted temporarily when it occurs. After the statistics are noted, simulation can resume.

The routing for those C's determined to be express shoppers proceeds in the same manner as that in the regular case. Transfer first occurs to :EXPLOOP where an unoccupied reserve position is located.

```
:EXPLOOP  JC(1,*LANEA,:SHPLANEA)
          JC(1,*LANEB,:SHPLANEB)
          JC(1,*LANEC,:SHPLANEC)
          DN
          JP(:EXPLOOP)
```

If these three lanes do not prove to be sufficient, another should be established.

Now assume that the first unoccupied reserve position happens to be *LANEA. In this case, C will shop on lane A. The lane is posted to reserve it, C moves to it, the reserve position is cleared, and C assumes position *LANEA for the delay of shopping time above moving around the overlay. The shopping time is obtained as a random number from 60 to 660 seconds (6 +/- 5 minutes).

```
:SHPLANEA R=(*READY)
          PO(*LANEA)
          ML(10,1)
```

```
CL(*LANEA)
MD(1,1)
RV(@SHOPTIME,60,660)
```

From this time must then be subtracted all of the movement delays associated with lane A. These include moving to the aisle (10 seconds), assuming the reserve position (1 second), moving down the aisle (15 seconds), moving to the back wall (1 second), and finally moving to the point common to all express shoppers, *EXPLINE (10 seconds). The sum of these times is 37 seconds. This is the minimum time necessary to move from the position beneath the cart cage to *EXPLINE, where all express shoppers meet. Delays outside of this requirement include the 3 seconds to move from the cart cage to the lane branching position, 6 seconds moving to the lane of the express counter from the common position *EXPLINE, and 12 seconds needed to move up the lane to the checkout counter. These times total to 21 seconds; this sum will be common to all express customers. Thus, the 37 and 21 second delays may be subtracted out of the shopping time, as they are expended throughout the route; C then delays for the remainder.

```
AO(@SHOPTIME,-,37)
AO(@SHOPTIME,-,21)
ST(@SHOPTIME)
```

The total delay is 58 seconds, 2 less than the minimum shopping time for an express customer.

After the specified period elapses, C moves down the aisle, to the back wall, and to the common point for all express shoppers.

```
          MD(15,1)
          MD(1,1)
          MR(10,1)
          JP(:EXPCHK)
```

The other two lanes, B and C, are handled similarly. The distances to them are 8 and 0, while the distances moving from them to *EXPLINE are 8 and 0 also. The delays associated with lanes B and C are 33 and 17 seconds, respectively. The only other changes necessary are in the labels used.

At the common position *EXPLINE, all express customers enter the waiting line, or queue, for the express checkout.

```
:EXPCHK   R=(*EXPLINE)
          IV(@TTLEXPQ)
          IV(@EXPQUEUE)
          SV(OBJ%1,CLOCK)
          IF(@EXPQUEUE,LT,@MAXEXPQ,THEN,:DOWN2)
          SV(@MAXEXPQ,@EXPQUEUE)
```

C's then move to the counter and exit the queue. Note the removal of the constant time in the queue before updating is done on the running total.

```
:DOWN2    MR(6,1)
          MU(12,1)
          DV(@EXPQUEUE)
          SV(%DIFFRNCE,CLOCK)
          AO(%DIFFRNCE,-,OBJ%1)
          AO(%DIFFRNCE,-,18)
          AO(%TTLEXPQT,+,%DIFFRNCE)
```

The checkout procedure occurs next, taking from 60 to 180 seconds (2 +/- 1 minutes).

```
          RV(@EXPCHK,60,180)
          ST(@EXPCHK)
```

The customer then returns his cart immediately to the storage area, at which time the storage variables are updated, as is the displayed number of carts available.

```
DV(@CRTINUSE)
SV(%DIFFRNCE,CLOCK)
AO(%DIFFRNCE,-,OBJ%2)
AO(%TTLCRTST,+,%DIFFRNCE)
IV(@NUMCARTS)
PV(*CRTSFREE,@NUMCARTS)
```

C then moves to the common point for all customers and transfers to the same routing explained for the regular customers earlier.

```
MU(6,5)
JP(:EXITSTOR)
```

This completes the routing for customer objects. The job statement to create them reflects the fact that it is job 3 and will take route 3, as well as identifying the character 'C' as that to be used to represent the objects on the overlay.

```
J=(3,C,3,0,0,2,1000)
```

Priority 2 is assigned to the job in order to leave priorities 0 and 1 for the initialization and statistics jobs, respectively. Finally, the number of C's to be created by the job is arbitrarily chosen as 1000, well over the 500 customers modeled in the GPSS program.

3. Statistics While the customer job of the previous section is running, PCModel can generate utilization statistics for up to 21 different screen locations and it indicates areas where customers must wait by way of the overlay generated. However, to generate figures of a more statistical nature, such as average lengths of waiting lines, arithmetic operations must be employed. Further, all of these types of statistics will be generated by a single job and its route; their creation will be covered now.

As discussed earlier, the method of generating average queue lengths and storage quantities will be to sample the current quantities in these structures at regular intervals. Again, this can not be performed each second because the variables for the running total would overflow in an unacceptably short period. In order to determine the interval, the maximum size of the variable to be sampled must be known. Additionally, since PCModel uses integer arithmetic, generation of each decimal place will require multiplication of the running total involved by a factor of 10. Assuming that one decimal place is deemed sufficient for averages, this limits the maximum value of the figure to be divided to 65535 DIV 10, or 6553. A single addition to this value will make it 6554, and that multiplied by 10 is 65540, larger than the PCModel variable maximum of 65535. In addition to this maximum sum of 6553, the maximum value to be added must be known. Averages will be determined for the number in the cart storage and the quantities in the cart, regular checkout, and express queues. The cart storage itself has as its maximum number of entries 20. Assuming that none of the queues will have more than 20 objects in them at any one time seems reasonable, so 20 is chosen as the maximum. If the simulation shows a maximum for any queue greater than 20, the calculations to follow must be adjusted.

The maximum sum was determined to be 6553; if the maximum increment for any sample is 20, the 6553 capacity would allow 6553 DIV 20, or 327 such samplings to be made. Now the decision must be made as to how long the simulation should be able to run without possibility of statistic overflow; this decision is the converse of how often to take samples. At 1 second between samples, overflow could not occur for 327 seconds (5.45 minutes), a clearly unacceptable period. Separating the samples by 240 seconds (4 minutes) will put off a possible overflow until 78,480 (240 * 327) seconds into the simulation; this is 21 hours and 48 minutes, a much more reasonable figure. This will be used as the value for the possible overflow time, %OVERFLOW. Also, this has defined the interval between successive passes through the statistics route, that is the 240 second figure.

Now can begin the coding for the route. As established, one pass needs to be made every 240 seconds through the route. In order to utilize a different combination of PCModel instructions, only one object will be created by the job; it will then loop through the route. The first object will enter the route for the first pass 240 seconds into the simulation.

```
BR(2,*DUMMY2,240)
```

This will be route 2 corresponding to the statistics job number of 2; as the object has no overlay usage, it is created at a dummy location and will stay there.

Concerning the loop times, the clock variable %LOOPTIME will be utilized to control the object transfer. It is not possible to use one of the object's clock parameters here for reasons that will be made clear at the bottom of the loop. The time of the first pass is taken from the previous BR instruction.

```
                    SV(%LOOPTIME,240)
```

Consider now what is to be accomplished by the statistics route. For the most part, the output to the overlay will consist of printing values for the statistics concerning the queues and storage: the current quantity of customers contained, the maximum contents at any one time, the total number of entries. The updating of all of these variables is handled in the customer route explained previously. The one exception to this is the computation of average values, for instance the average contents of a queue. Now review the sequence of operations of the averaging process. As this program has been developed, the contents of a queue will be added periodically to a running sum of such periodic checks. To determine the average value is then a simple operation of division of the running sum by the number of samplings, or counts, made. This number will be kept in the variable @NUMBCTS and updated on each iteration.

```
        :TOP        IV(@NUMBCTS)
                    PV(*COUNTS,@NUMBCTS)
```

Additionally, PCModel's limitation to integer division should be compensated for; left alone, integer division will result in truncated quotients. In order to achieve a rounding effect on the quotient, half of the divisor needs to be added to the dividend before division. The dividend will be the various running totals, but the divisor will always be @NUMBCTS. Thus the rounding factor, @ROUND, can be calculated now, just one time for all divisions to be performed for the current pass.

```
SV(@ROUND,@NUMBCTS)
AO(@ROUND,/,2)
```

The value in @ROUND will thus be added to any number to be divided by @NUMBCTS to obtain a rounded result.

Now the statistics for a particular entity, say the cart storage, may be focused on. The first GPSS statistic to be output will be the total number of carts available, @TTLCARTS.

```
PV(*CRTSTG1,@TTLCARTS)
```

There are a couple of points to be observed here. First is the use of another variable concerning the cart storage besides @NUMBCARTS and @CRTINUSE. @CRTINUSE is the number of carts currently in use, or the number of entries in the cart storage currently; it always starts out as zero. @NUMCARTS is the converse of @CRTINUSE; it represents the number of carts free (for display purposes). @NUMCARTS will be initialized from @TTLCARTS. It might seem at first that @TTLCARTS should be declared as a constant or even coded as a numerical value in the program; however, declaring it as a variable value allows it to be changed from the variable screen along with @NUMCARTS without altering the source program. If the program is initialized, not even @NUMCARTS need be edited on the value screen, as the initialization job will handle it. Additionally, the correct value for the maximum number of entries possible in the storage will always be reflected in the statistics on the overlay as it is printed straight from the variable.

One other thing to note is the pattern of the overlay position labels. The various fields for each entity statistics are collected for will be numbered sequentially. Also, the label will indicate whether it is dealing with the carts, regular checkout or express checkout by including the string 'CRT', 'REG', or 'EXP', respectively; the association with a storage or queue will be made by 'STG' or 'QUE'.

Next comes the actual determination of the average number in the storage. The current number is in @CRTINUSE; this will be added to the running total @CKCRTUSE for checking cart usage. This sum is in turn copied for computation of the average.

```
AO(@CKCRTUSE,+,@CRTINUSE)
SV(@AVERAGE,@CKCRTUSE)
```

Now as decided at the beginning of this section the average will be computed to one decimal place. Multiplying by 10, adding in the rounding factor, and dividing will produce the average value with an implied decimal place between the ten's and one's position of the number.

```
AO(@AVERAGE,*,10)
AO(@AVERAGE,+,@ROUND)
AO(@AVERAGE,/,@NUMBCTS)
```

It is desired to get the value printed out with the decimal point inserted in the correct position. Advantage can be made of the fact that the Print Value instruction prints at the location specified a 5-space field, the leftmost space of which is printed at the given location. The value always prints 5 spaces, with any leading zeroes suppressed as blanks;

the fact that 5 spaces are used follows from the variable value limit of 65535 which requires 5 spaces.

Now print the average with the implied decimal point on the overlay.

```
PV(*CRTSTG2C,@AVERAGE)
```

Next print a character string consisting of a single decimal point at the screen position which is occupied by the ten's digit, or second rightmost position of the value just printed.

```
PM(*CRTSTG2B,.)
```

This creates the decimal portion of the average. All that remains now is to eliminate the one's digit from the average and print it a a position that will place its rightmost digit immediately to the left of the decimal point. Elimination of the one's digit is accomplished by simply dividing the value by 10.

```
AO(@AVERAGE,/,10)
PV(*CRTSTG2A,@AVERAGE)
```

Thus the average accurate to one decimal place is obtained and displayed.

The third value to be output will be the total number of entries into the storage, @TTLCRTSU, which was updated by the customer route.

```
PV(*CRTSTG3,@TTLCRTSU)
```

Another common statistic for output is the average time spent in the storage by customers. The customer route maintained a running total of such times, %TTLCRTST; the number of customers that contributed to this sum can be found by subtracting the number of customers currently in the storage, @CRTINUSE, from the total number of entries, @TTLCRTSU. However, trial runs of the simulation indicate that the division operation, unlike addition and subtraction, is either in error or not supported for the clock variables of PCModel. Thus the best that can be accomplished is to print out the accumulated sum so that the computation may at least be done by hand.

```
PV(*CRTSTG4,%TTLCRTST)
```

The final two statistics to be displayed are the number of customers who currently have carts, @CRTINUSE, and the maximum number of carts ever in use at any one time, @MAXCRTSU. Both of these values are maintained by the customer route and need be simply displayed here.

```
PV(*CRTSTG5,@CRTINUSE)
PV(*CRTSTG6,@MAXCRTSU)
```

Generation of statistics for each of the three queues is identical for each; thus the generation concerning one queue, say that for the regular checkout, will be sufficient to explain the others. Further, the queue statistics are for the most part handled in the same manner as those of the storages. They are reordered here to reflect the typical output of GPSS results.

The first value is the maximum in the queue, @MAXREGQ, which is updated in the customer route.

```
PV(*REGQUE1,@MAXREGQ)
```

The average queue length is achieved by the same means as those used for the average number in the storage. Here @CKREGQ is the running total and @REGQUEUE is the current number in the queue.

```
AO(@CKREGQ,+,@REGQUEUE)
SV(@AVERAGE,@CKREGQ)
AO(@AVERAGE,*,10)
AO(@AVERAGE,+,@ROUND)
AO(@AVERAGE,/,@NUMBCTS)
PV(*REGQUE2C,@AVERAGE)
PM(*REGQUE2B,.)
AO(@AVERAGE,/,10)
PV(*REGQUE2A,@AVERAGE)
```

The remaining values to be displayed are the total entries into the queue, @TTLREGQ, the total time spent in the queue by customers, %TTLREGQT, and the current contents of the queue, @REGQUEUE.

```
PV(*REGQUE3,@TTLREGQ)
PV(*REGQUE4,%TTLREGQT)
PV(*REGQUE5,@REGQUEUE)
```

The remaining statistics concern the express checkout queue and the cart waiting line queue. The routing for them is identical to that for the regular checkout queue above, with the substitution of their respective variables and labels.

After generation of the statistics on the current loop, the overflow situation will be examined. Actually, this would probably best be handled using a comparison between the variable values in question and their overflow values. However the problem will be managed using clock values to display their use. The time of the first possible occurrence of overflow will be stored in %OVERFLOW; the actual time was derived previously as 21 hours, 48 minutes. Thus the system clock may be compared against this time to determine if overflow may be imminent.

```
IF(CLOCK,LT,%OVERFLOW,:SKIP)
```

If overflow is indeed impending, a message will be displayed and the simulation halted until the user strikes a key; at that point the message is erased and simulation continues.

```
PM(*MESSAGE,STATISTICS CAN OVERFLOW)
WK
PM(*MESSAGE,xxxxxxxxxxxxxxxxxxxxxxx)
```

Note that this arrangement will cause the simulation to halt on each subsequent pass of the statistics route unless the value of %OVERFLOW is altered. One could edit the value screen the first time the message occurs, examine the values of the various running totals, and increase %OVERFLOW to a conservative estimate of the next time to check for overflow if it has not yet occurred. Again, this whole arrangement would probably best be handled by making comparisons on the running totals themselves; the employed method is used for focusing attention on the possible uses of clock values.

All that remains for embodiment in the route is the looping mechanism for the single job object. This is simply a matter of delaying the object until the time of the next statistics update to be made. The determination of the next time can be calculated by adding the delay to the current time; prior to this, the time of the current update is displayed.

```
:SKIP     PV(*TIME,%LOOPTIME)
          AO(%LOOPTIME,+,240)
```

The original purpose of utilizing the %LOOPTIME variable was to be able to use it at this point to delay the object until the next loop time with the Wait Clock instruction.

```
WC(%LOOPTIME)
```

However, for an undetermined reason, the execution of this instruction fails to take place properly some eighteen hours into the simulation; it simply does not wait until the specified clock time before releasing the object. Thus it is replaced in the final version of the program with a Set Time instruction to accomplish the same goal.

```
ST(240)
```

The object then loops to the top of the routing instructions for the statistics generation; an End Route instruction follows to complete the statistics route.

```
JP(:TOP)
ER
```

The job statement of job 2 will generate the single object for route 2 with the priority of 1 established for it.

```
J=(2,#,2,0,0,1,1)
```

The '#' character is used arbitrarily as no display function is associated with this job.

4. Initialization The last job required by almost any PCModel simulation is that dedicated to initialization of the model. The job object enters the simulation before any other objects.

```
BR(1,*DUMMY1,0)
```

Like the statistics route, there is no display purpose associated with this route and the object is thus placed on the overlay at an arbitrary location.

The first task is initialization of the random number sequence; this is followed by the time of arrival of the first customer and the count of customers who have completed the simulation.

```
RS(#SEED)
SV(@ARRIVE,240)
SV(@TERMCNT,0)
```

The termination count,@TERMCNT, for halting the simulation is initialized here as well.

Next, the number of carts free for use, @NUMCARTS, is set from the original total of carts, @TTLCARTS. @TTLCARTS will be unaltered while @NUMCARTS will be incremented and decremented throughout the simulation; as @TTLCARTS is output on the display, editing its value on the value screen is the only work necessary to alter the simulation parameter for the number of carts.

```
SV(@NUMCARTS,@TTLCARTS)
```

The remaining variables to be initialized are concerned with the statistics of the simulation. The number of counts made, @NUMBCTS, is zeroed here first.

```
SV(@NUMBCTS,0)
```

This is followed by the variables for the cart storage.

```
SV(@CRTINUSE,0)
SV(@TTLCRTSU,0)
SV(@MAXCRTSU,0)
SV(@CKCRTUSE,0)
SV(%TTLCRTST,0)
```

The variables associated with the regular checkout queue, the express checkout queue, and the cart waiting line queue follow in similar fashion.

```
SV(@REGQUEUE,0)
SV(@MAXREGQ,0)
SV(@TTLREGQ,0)
SV(@CKREGQ,0)
SV(%TTLREGQT,0)

SV(@EXPQUEUE,0)
SV(@MAXEXPQ,0)
SV(@TTLEXPQ,0)
SV(@CKEXPQ,0)
SV(%TTLEXPQT,0)

SV(@CRTQUEUE,0)
SV(@MAXCRTQ,0)
SV(@TTLCRTQ,0)
SV(@CKCRTQ,0)
SV(%TTLCRTQT,0)
```

As far as the initialization process goes, the route is complete and the object can exit the model. To save defining yet another job and route, the logic necessary for the determination of the time to note statistics for comparison with those of the GPSS model is incorporated here. The object is held at this point until the specified count is reached.

```
IF(@TERMCNT,LT,@GPSSTERM,THEN,WAIT)
```

When the count is reached, a message indicating such will be displayed and the simulation halted until the a key is struck.

```
PM(*MESSAGE,500 OBJECTS COMPLETED)
WK
```

This allows the user to copy the current statistics screen before another update occurs.

The message is then erased and the object allowed to exit the simulation.

```
PM(*MESSAGE,xxxxxxxxxxxxxxxxxxxxx)
ER
```

The job statement associated with the route will create the single object of the route. The arbitrary character 'X' is used as no significance should be attached to the overlay output of the route.

```
J=(1,X,1,0,0,0,1)
```

Finally, the zero priority of the job is identified, along with the association of job 1 with route 1.

5. Load-Time Directives The remaining portion of the PCModel program for the supermarket simulation consists of the load-time directives used by the loader in creating the run-time program. The first pair of directives, M and W, specify the maximum number of objects that will ever be allowed in the simulation at any one time and the initial value for the number that will be allowed in the model during the current clock interval, respectively.

```
M=(100)
W=(100)
```

The values use here are outside estimates of the values required. This is also the case for the number of symbols to reserve storage for.

```
S=(250)
```

Painstaking efforts were made in the warehouse simulation to illustrate how these values can be arrived at in a conservative fashion; limited memory may require that this be done, but this is not the case here.

Next in the sequence are the parameters concerning the overlay. X and Y give its column and row dimensions, respectively, while V determines the position of the overlay to coincide with the upper left corner of the display during loading.

```
X=(81)
Y=(47)
V=(XY(1,0))
```

Note that while Y is larger than the 23 line limit of a single screen in order to allow space for the statistics, X is only one larger than the default width of 80 columns; this in combination with the V directive allows the first column of the overlay to be numbered sequentially and used as a reference without distracting from the overlay of the running simulation. This can be seen more clearly in the overlay in Figure 6.

The D (Description definition) and O (Overlay) directives are supplied next.

```
D=  (SEE APPENDIX 2)
O=  (SEE FIGURE 6)
```

The next section consists of symbol definitions. As this section is the first occurrence of each symbol in the program, they should be defined in the accompanying comments . The fact that all symbols must occur here works to the programmer's advantage during both the commenting and debugging processes. The variables of the program are the first symbols to be listed, along with the specification of their initial values after the loading process. For the customer arrival rate and chance of being an express versus a regular customer:

```
@ARRIVE=(240)
@CHANCE=(0)
```

The generated shopping, express checkout, and regular checkout times:

```
@SHOPTIME=(0)
@EXPCHK=(0)
@REGCHK=(0)
```

The count toward termination and the value to be reached:

```
@TERMCNT=(0)
@GPSSTERM=(500)
```

The number of statistics counts made, the rounding factor, and the variable used for averaging:

```
@NUMBCTS=(0)
@ROUND=(0)
@AVERAGE=(0)
```

The number of carts in the store and the number currently available:

```
@TTLCARTS=(20)
@NUMCARTS=(0)
```

The variables pertaining directly to the cart storage, as explained in the customer route:

```
@CRTINUSE=(0)
@TTLCRTSU=(0)
@MAXCRTSU=(0)
@CKCRTUSE=(0)
%TTLCRTST=(0000:00:00)
```

The variables concerned with the regular checkout, express, checkout, and cart waiting line queues successively:

```
@REGQUEUE=(0)
@MAXREGQ=(0)
@TTLREGQ=(0)
@CKREGQ=(0)
%TTLREGQT=(0000:00:00)

@EXPQUEUE=(0)
@MAXEXPQ=(0)
@TTLEXPQ=(0)
@CKEXPQ=(0)
%TTLEXPQT=(0000:00:00)

@CRTQUEUE=(0)
@MAXCRTQ=(0)
@TTLCRTQ=(0)
@CKCRTQ=(0)
%TTLCRTQT=(0000:00:00)
```

The remaining clock variables for identifying the time of overflow, the current statistics loop, and the time between object events:

```
%OVERFLOW=(0021:48:00)
%LOOPTIME=(0000:00:00)
%DIFFRNCE=(0000:00:00)
```

Next come the declarations of constants used in the program. For the supermarket, a random number seed and percent of express shoppers (versus regular shoppers) were used.

```
#SEED=(9997)
#PERCENT=(30)
```

The last group of symbols consist of labels for overlay positions. For the dummy locations of routes 1 and 2:

```
*DUMMY1=(XY(2,2))
*DUMMY2=(XY(1,39))
```

The entry and exit positions of shoppers:

```
*ENTRY=(XY(40,0))
*EXIT=(XY(68,4))
```

The positions to wait for a cart and then an open aisle:

```
*CARTS=(XY(62,2))
*READY=(XY(62,5))
```

The common positions all regular or express customers meet when headed for their respective checkouts:

```
*REGLINE=(XY(44,22))
*EXPLINE=(XY(62,22))
```

The locations occupied during regular and express checkout:

```
*REGLANE=(XY(75,10))
*EXPLANE=(XY(68,10))
```

The reserve positions at the head of each lane:

```
*LANE1=(XY(4,6))
*LANE2=(XY(12,6))
   .         .
   .         .
   .         .
*LANE9=(XY(44,6))
*LANEA=(XY(52,6))
*LANEB=(XY(54,6))
*LANEC=(XY(62,6))
```

The output areas for the number of carts available and the overflow and termination messages:

```
*CRTSFREE=(XY(65,2))
*MESSAGE=(XY(2,1))
```

The sites for the number of statistic samples made and the time of the last update:

```
*COUNTS=(XY(28,25))
*TIME=(XY(60,25))
```

The fields used for cart storage statisticss, particularly the storage capacity (1), the average contents (2), total entries (3), total time spent in the storage (4), current contents (5), and maximum contents (6):

```
*CRTSTG1=(XY(16,30)
*CRTSTG2A=(XY(25,30)
*CRTSTG2B=(XY(30,30)
*CRTSTG2C=(XY(27,30)
*CRTSTG3=(XY(37,30)
*CRTSTG4=(XY(46,30)
*CRTSTG5=(XY(60,30)
*CRTSTG6=(XY(71,30)
```

Note how the subfields of the second data region overlap in order to produce the decimal average.

The remaining overlay positions deal with the queue statistics for the regular checkout, express checkout, and the cart waiting line. Each queue's fields are ordered as maximum contents (1), average contents (2), total entries (3), total time spent in queue (4), and current contents (5).

```
*REGQUE1=(XY(14,36)
*REGQUE2A=(XY(24,36)
*REGQUE2B=(XY(29,36)
*REGQUE2C=(XY(26,36)
*REGQUE3=(XY(35,36)
*REGQUE4=(XY(44,36)
*REGQUE5=(XY(58,36)

*EXPQUE1=(XY(14,37)
*EXPQUE2A=(XY(24,37)
*EXPQUE2B=(XY(29,37)
*EXPQUE2C=(XY(26,37)
*EXPQUE3=(XY(35,37)
*EXPQUE4=(XY(44,37)
*EXPQUE5=(XY(58,37)

*CRTQUE1=(XY(14,38)
*CRTQUE2A=(XY(24,38)
*CRTQUE2B=(XY(29,38)
*CRTQUE2C=(XY(26,38)
*CRTQUE3=(XY(35,38)
*CRTQUE4=(XY(44,38)
*CRTQUE5=(XY(58,38)
```

Following the symbol definitions are the job directives. The derivation of each was explained at the end of its respective route.

```
J=(1,X,1,0,0,0,1)
J=(2,#,2,0,0,1,1)
J=(3,C,3,0,0,2,1000)
```

The last group of directives are those used to define utilization locations. The statistics generated for the positions are essentially the percentages of occupied time on an hourly basis. As the checkout processes take place at single positions on the overlay, those spots can be specified here to create information that corresponds to that of a facility in GPSS. A facility is defined as a permanent entity which can accomodate only a single model object at a time. This is certainly the case with a single overlay position in a PCModel system.

```
U=(1,Reg Chk,*REGLANE)
U=(2,Exp Chk,*EXPLANE)
```

Note that meaningful label is specified to identify the corresponding data on the utilization screen.

As the two positions for checkout are the only facilities used in the program, the other 19 of the 21 possible utilization positions are not required. However, they can be put to good use in checking the utilizations of the reserve positions of the aisles. This will provide a simple method of telling whether or not the allocated number of aisles is sufficient.

```
U=(3,Lane 1,*LANE1)
U=(4,Lane 2,*LANE2)
      .        .
      .        .
      .        .
U=(14,Lane C,*LANEC)
```

This completes the sequence of load-time directives. The rest of the program is ordered to include the routes sequentially by number, as a matter of programming style. The program is thus complete and ready to be run for simulation of the supermarket. Appendix B contains a complete listing of the program.

D. RUN THE SIMULATION

As was done for the warehouse simulation, the supermarket problem was run under the incremental mode to obtain as much accuracy as possible. The O command was used to save the utilization statistics every ten hours of the 40 hour simulation period. The output after the 10 hour and 20 hour times are grouped in Table V; likewise for the statistics at the 30 and 40 hour marks in Table VI.

The utilization statistics, it will be remembered, were used in part to aid in the debugging phase. Twelve of the twenty-one possible data positions were assigned to the lane reserve positions in order that they could be monitored for usage. Of particular interest are the statistics for the reserve spots of lanes 9 and C, as they are only used when all other positions for their type of customer are full. Examination of Tables V and VI show that possible trouble spots are the sixth and seventh hours for lane 9 and the tenth and sixteenth hours for lane C. *LANEC is never occupied after the sixteenth hour and *LANE9 never after the seventh hour of the forty hour period. This suggests that the simulation goes through a settling-in period after which the number of aisles allocated is more than sufficient. One lane never being used is only good for indicating that the others are sufficient. Concerning the indicated trouble spots, it can be seen for hour

TABLE V

SUPERMARKET STATISTICS: HOURS 10 AND 20

HOURLY UTILIZATION FIGURES

--TOOL--	HOUR1	HOUR2	HOUR3	HOUR4	HOUR5	HOUR6	HOUR7	HOUR8	HOUR9	HOUR10
Reg Chk	32.63	94.80	99.91	64.16	75.94	76.27	100.00	100.00	100.00	100.00
Exp Chk	04.55	14.38	22.38	29.13	13.61	13.66	11.36	13.08	06.25	25.00
Lane 1	81.66	77.94	82.08	69.16	72.86	78.58	92.27	80.08	90.97	82.33
Lane 2	73.83	41.44	86.33	70.58	74.47	66.72	84.19	63.13	82.08	74.41
Lane 3	33.08	63.47	61.58	71.80	62.44	61.55	63.52	66.75	59.66	52.47
Lane 4	46.00	77.41	60.75	20.97	40.08	98.77	51.05	64.66	56.30	74.69
Lane 5	17.63	76.27	20.27	02.22	95.19	48.13	56.05	61.77	20.52	80.02
Lane 6	25.27	64.08	27.61	00.00	56.27	40.19	20.69	63.63	00.00	37.05
Lane 7	00.00	30.86	00.00	00.00	41.63	52.30	00.00	61.88	00.00	00.00
Lane 8	00.00	00.00	00.00	00.00	00.00	41.27	19.00	00.00	00.00	00.00
Lane 9	00.00	00.00	00.00	00.00	00.00	40.38	16.30	00.00	00.00	00.00
Lane A	15.44	30.13	40.41	59.41	36.02	40.25	13.05	56.11	25.22	35.97
Lane B	00.00	00.00	03.25	16.05	04.66	14.16	00.00	00.00	16.13	17.47
Lane C	00.00	00.00	00.00	00.00	00.00	00.00	00.00	00.00	00.00	07.22

--TOOL--	HOUR11	HOUR12	HOUR13	HOUR14	HOUR15	HOUR16	HOUR17	HOUR18	HOUR19	HOUR20
Reg Chk	100.00	100.00	100.00	100.00	100.00	100.00	100.00	85.72	100.00	96.94
Exp Chk	14.36	08.80	18.00	06.16	11.80	19.61	19.69	03.27	15.91	24.94
Lane 1	90.91	68.38	83.80	72.52	81.38	82.38	81.33	77.88	87.77	81.38
Lane 2	86.13	56.50	81.80	78.72	80.47	71.69	69.58	94.77	73.77	78.75
Lane 3	77.86	68.38	53.02	84.13	68.55	88.58	43.16	62.63	69.22	83.05
Lane 4	69.38	48.25	18.55	47.02	42.80	44.52	35.63	81.02	31.11	34.36
Lane 5	67.80	16.16	16.00	27.27	24.33	45.83	26.72	53.13	46.41	44.11
Lane 6	41.05	12.66	04.50	50.97	21.38	37.22	00.00	37.97	51.69	31.38
Lane 7	15.38	21.83	00.00	17.13	00.00	55.05	00.00	34.58	07.19	43.02
Lane 8	00.00	00.00	00.00	00.00	00.00	15.13	00.00	00.00	00.00	00.00
Lane 9	00.00	00.00	00.00	00.00	00.00	00.00	00.00	00.00	00.00	00.00
Lane A	29.38	29.72	28.63	34.88	26.19	48.22	64.11	04.94	24.58	38.36
Lane B	14.13	17.05	04.66	00.00	06.02	21.83	04.61	00.00	05.44	27.50
Lane C	00.00	00.00	00.00	00.00	00.00	01.36	00.00	00.00	00.00	00.00

TABLE VI

SUPERMARKET STATISTICS: HOURS 30 AND 40

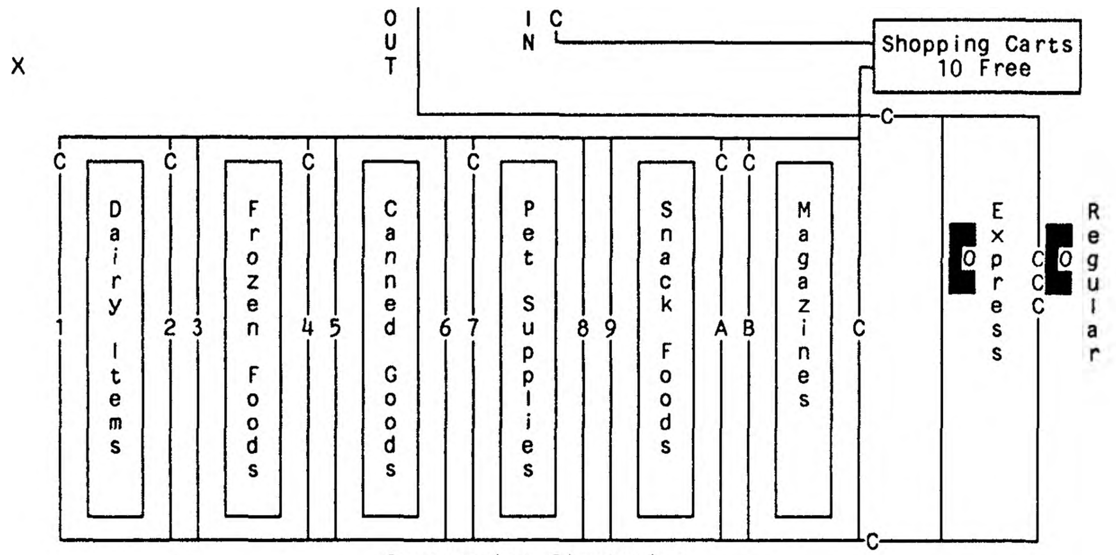
HOURLY UTILIZATION FIGURES

--TOOL--	HOUR21	HOUR22	HOUR23	HOUR24	HOUR25	HOUR26	HOUR27	HOUR28	HOUR29	HOUR30
Reg Chk	99.47	77.36	100.00	100.00	100.00	91.08	96.50	75.13	98.11	100.00
Exp Chk	20.38	16.27	21.97	11.77	10.97	05.91	13.38	14.36	10.88	16.36
Lane 1	85.94	92.94	87.41	82.47	82.16	73.88	70.33	78.52	85.22	75.66
Lane 2	79.80	85.11	57.33	94.97	72.16	63.41	70.47	83.22	88.33	90.36
Lane 3	59.61	73.25	59.58	97.05	55.11	62.83	78.38	63.36	83.30	38.19
Lane 4	86.75	57.61	51.08	69.86	79.58	53.38	62.63	88.80	69.11	71.58
Lane 5	51.83	49.50	25.11	68.80	25.55	78.63	49.88	85.55	75.61	16.13
Lane 6	00.00	26.44	15.94	54.77	21.36	00.00	27.86	30.55	38.16	39.61
Lane 7	11.25	00.00	15.83	14.02	00.00	00.00	00.00	00.00	54.08	00.00
Lane 8	00.00	00.00	13.75	00.00	00.00	00.00	00.00	00.00	00.00	00.00
Lane 9	00.00	00.00	00.00	00.00	00.00	00.00	00.00	00.00	00.00	00.00
Lane A	22.94	31.91	25.22	21.58	22.41	15.44	32.47	30.88	40.83	35.19
Lane B	23.30	00.00	12.75	01.13	00.00	16.13	00.00	11.02	01.38	10.63
Lane C	00.00	00.00	00.00	00.00	00.00	00.00	00.00	00.00	00.00	00.00

--TOOL--	HOUR31	HOUR32	HOUR33	HOUR34	HOUR35	HOUR36	HOUR37	HOUR38	HOUR39	HOUR40
Reg Chk	71.13	100.00	100.00	100.00	84.41	89.22	85.02	61.36	82.47	84.63
Exp Chk	31.55	09.44	08.61	06.44	12.86	12.38	20.30	20.08	11.86	11.80
Lane 1	85.55	92.27	82.16	82.80	91.94	96.61	72.00	90.44	79.50	83.91
Lane 2	63.08	67.86	92.63	69.75	67.58	68.91	72.00	52.80	57.75	93.77
Lane 3	38.16	76.00	91.66	41.86	87.22	90.80	80.05	97.19	38.86	82.22
Lane 4	75.30	45.86	79.25	43.61	42.11	96.50	54.08	55.02	82.33	67.94
Lane 5	14.63	29.83	49.61	59.13	17.38	63.52	06.33	25.61	67.63	57.77
Lane 6	50.52	21.58	71.22	00.00	00.00	47.44	00.00	00.00	41.25	48.41
Lane 7	49.44	03.83	81.41	23.38	00.00	32.97	00.00	00.00	00.00	31.61
Lane 8	51.58	08.30	15.55	00.00	00.00	00.00	00.00	00.00	00.00	00.00
Lane 9	00.00	00.00	00.00	00.00	00.00	00.00	00.00	00.00	00.00	00.00
Lane A	55.77	13.05	14.36	23.97	37.00	16.19	47.27	31.41	26.30	25.08
Lane B	20.19	00.00	00.00	00.00	02.63	18.13	12.80	06.27	13.00	16.83
Lane C	00.00	00.00	00.00	00.00	00.00	00.00	00.00	00.00	00.00	00.00

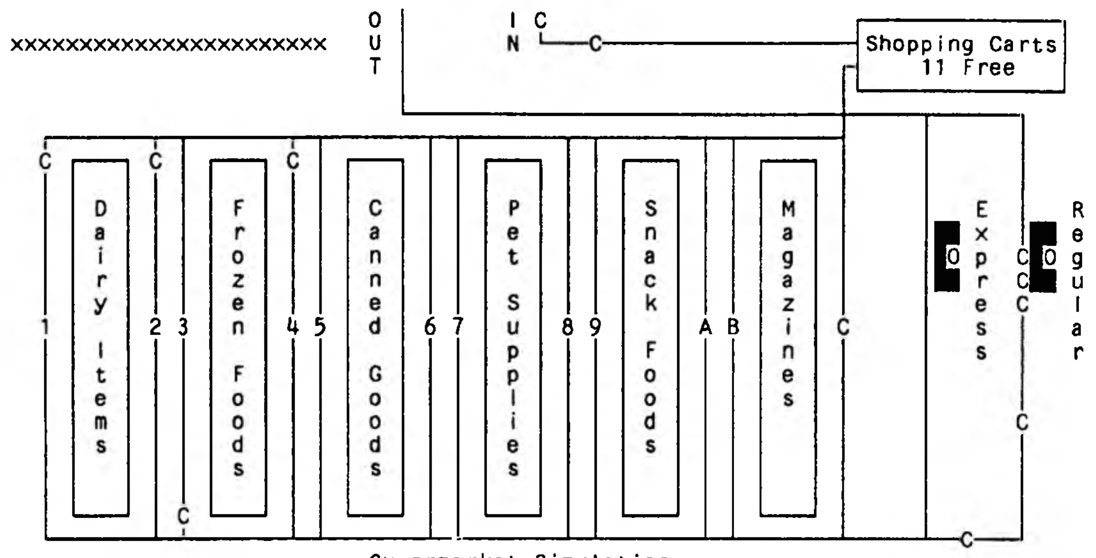
16 that *LANEC is only occupied for 1.36% of the hour, while for hour 10 the figure is 7.22%. Neither is very high and, as will be seen shortly, the use of the positions did not exaggerate the statistics. Indeed, it may well be that no other lane was needed at all; however this fact cannot be gleaned from the utilization figures alone. For *LANE9, it can be observed that during hour 7, although *LANE9 was occupied, *LANE7 was not. Thus, there is no possible problem here. (This occurrence of figures indicates aisle 9 was used during the previous hour while aisle 7 was busy; aisle 7 was then cleared before the end of the hour and not used at all during the next hour.) The utilization figure for *LANE9 in hour 6 is 40.38% and is cause for concern, as is the figure for *LANEC in hour 10 above. To justify that neither of these indicate an appreciable error in the statistics, consider what would happen if there were not enough lanes for an extended period. In that case, customers would begin to back up from the *READY position they branch from. When 4 customers are held up, the position *CARTS becomes occupied which will then keep customers from being able to exit the cart queue. As the generated statistics will show, the accumulated time spent in the cart queue is zero; thus, no object ever had to wait for any appreciable period. With this analysis complete, the results of the simulation can be accepted for evaluation.

In addition to storing the utilization statistics, the overlay itself was filed at the 20 and 40 hour marks; the overlays are presented in Figures 7 and 8, respectively. The



```
                       Supermarket Simulation

Number of Counts Taken:     299     Time of Last Update:  0019:56:00

         Storages
 Storage   Capacity    Average    Entries   Total Time   Current    Maximum
                       Contents              in Storage   Contents   Contents
 Carts          20      10.4        309     0202:56:53     10         18


         Queues
 Queue     Maximum    Average    Total     Total Time    Current
           Contents   Contents   Entries   in Queue      Contents
 RegCk        13        4.2       213      0070:15:38       2
 ExpCk         2        0.0        89      0000:15:52       0
 Cart          4        0.4       310      0000:00:00       1


        Statistics are updated every four minutes of simulation time.



0020:00:00 RIH P=  0 M=  100 W=   14 C=  299
```

Figure 7. Supermarket Overlay: Hour 20



```
                    Supermarket Simulation

 Number of Counts Taken:    599     Time of Last Update:  0039:56:00

         Storages
  Storage    Capacity     Average    Entries    Total Time    Current    Maximum
                          Contents              in Storage    Contents   Contents
  Carts          20          9.4        609     0370:01:18       9          18


         Queues
  Queue     Maximum     Average     Total     Total Time     Current
            Contents    Contents    Entries   in Queue       Contents
  RegCk        13          3.2        430     0098:14:29        2
  ExpCk         2          0.0        173     0000:20:33        0
  Cart          4          0.4        609     0000:00:00        0


         Statistics are updated every four minutes of simulation time.



0040:00:00 RIH P=   0 M=   100 W=    12 C=   601
```

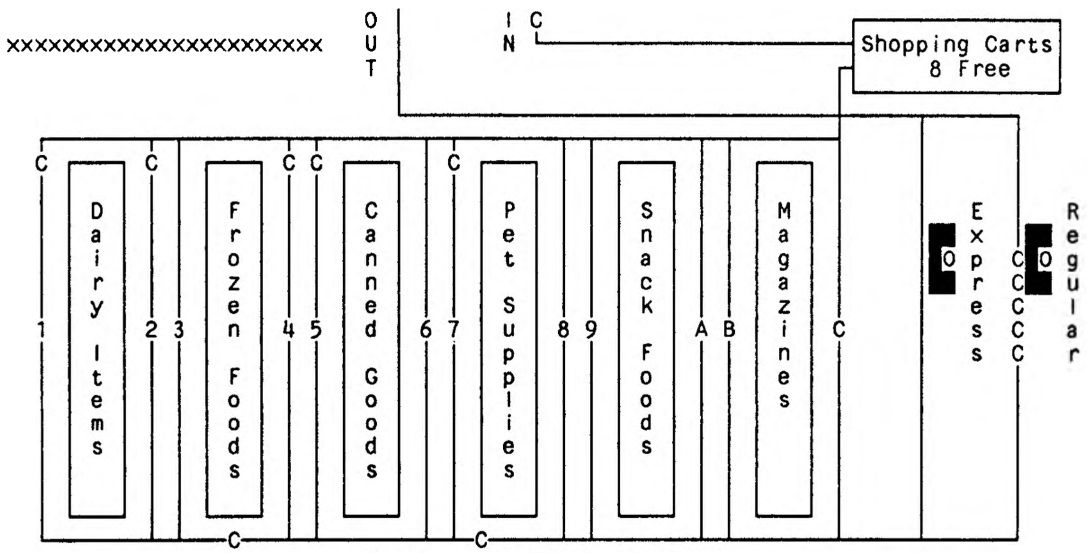
Figure 8. Supermarket Overlay: Hour 40

overlay at the time the GPSS simulation was terminated was saved also and can be seen in Figure 9; this will allow comparison of results in the next section.

E. EVALUATE THE RESULTS

The comparison of the output from the PCModel simulation and the GPSS simulation is best accomplished by assimilating like statistics for each. The GPSS statistics of interest are concerned with the queues (waiting lines for a cart, the regular checkout, and the express checkout), the storage (carts available to customers), and the facilities (the regular and express checkout counters). The overlay of Figure 9 displays a number of the statistics of interest; the rest will be determined here.

As discussed in the development of the software, it is impossible to generate average times as division of clock variables is not supported. However, running totals were kept for the amounts of time spent in the various queues and storage. The total time spent in the cart storage was 317 hours, 22 minutes, and 27 seconds (0317:22:27), or 1,142,547 seconds, as of the time of GPSS termination. The total number of entries into the storage was 512, with 12 still in the storage at the time the statistics were updated. Therefore 500 objects contributed to the running time total, so the average per object is 2285.094 seconds (1,142,547 / 500). The average times spent in the queues are obtained in a similar fashion: 916.969 seconds for the regular checkout queue, 7.239 seconds for the express checkout queue, and 0.000 seconds for the cart queue.



```
                      Supermarket Simulation
Number of Counts Taken:     497      Time of Last Update:  0033:08:00

       Storages
 Storage   Capacity     Average    Entries    Total Time    Current     Maximum
                        Contents              in Storage    Contents    Contents
 Carts          20         9.7        512     0317:22:27       12          18


       Queues
 Queue     Maximum     Average    Total     Total Time    Current
           Contents    Contents   Entries   in Queue      Contents
 RegCk        13          3.5       363     0091:26:32       4
 ExpCk         2          0.0       142     0000:17:08       0
 Cart          4          0.4       512     0000:00:00       0


        Statistics are updated every four minutes of simulation time.


0033:10:17 RIH P=  0 M=  100 W=   14 C=  501
```

Figure 9. Supermarket Overlay: Time of GPSS Termination

The determination of facility usage makes use of the utilization statistics. For the regular checkout location, the average of the hourly percentage utilizations over the 40 hour period is 90.557%; thus the counter was in use 36.223 hours. However, the customer route has each object move to the counter and pause for 9 seconds before the delay for checkout. (This is the last move up the aisle to the counter.) This time needs to be subtracted out before the utilization figure is comparable to that of GPSS. The overlay at the end of the 40 hour period, shown in Figure 8, indicates that the regular checkout queue had a total of 430 entries with 2 of them still in the queue at the time of update. The overlay also indicates that 1 customer is still at the checkout counter. The remaining 427 have all been handled by the checkout facility. The total time due to the 9 second move delay is then 3843 seconds (9 * 427). Subtracting this time from the 36.223 hours the counter was in use leaves 35.156 hours the counter was occupied due to the checkout process alone. Division by the entire period gives a true utilization factor of 0.879 for the regular checkout facility. The same process applied to the express checkout counter produces an average total utilization figure of 14.558%. The amount of time occupied was then 5.8351 hours, 173 seconds of which is movement delay time. Thus the counter delay time was 5.787 hours and the facility utilization factor 0.145.

Two points should be mentioned here. First, the overlay screen was last updated 4 minutes prior to the end of the 40 hour period. While the PCModel utilization figures are accurate for the period, the number of objects that occupied the facility locations may be slightly under the true count; this would affect, although slightly, the statistics. For example, if one more object completed use of the regular checkout counter, the movement delay would change from 3843 seconds to 3852. This leaves 35.153 hours for the facility utilization time which yields the same factor of 0.879. Thus it is negligible and can be ignored. The 1 second figure for the express checkout amounts to an even smaller discrepancy.

Second, the facility utilizations for the PCModel simulation are derived for the entire 40 hour period, rather than up to the time the GPSS simulation halted (0033:10:15). This was done because the utilization statistics are not accurate for the current hour until its end. Using the utilizations at the end of the forty hour period solves this problem. In any case, the result after forty hours should be more accurate, if any difference occurs at all, because more objects have been simulated.

The accumulated statistics are arranged in Table VII for comparison. The figures for PCModel's queue contents are rather exaggerated because of the fact that the customer objects were moving about the screen at a finite speed while inside the queues. Thus objects inside queues were not necessarily waiting for something. This is the trade off with generating graphic output that is meaningful. The average storage contents are not influenced on the other hand, since

TABLE VII

SUPERMARKET STATISTICS: PCMODEL VERSUS GPSS

PCMODEL Facilities

Facility	Average Utilization
RegCk	0.879
ExpCk	0.145

GPSS Facilities

Facility	Average Utilization
RegCk	0.837
ExpCk	0.145

PCMODEL Storages

Storage	Capacity	Average Contents	Entries	Average Time/Cust	Current Contents	Maximum Contents
Carts	20	9.7	512	2285.094	12	18

GPSS Storages

Storage	Capacity	Average Contents	Entries	Average Time/Cust	Current Contents	Maximum Contents
Carts	20	8.364	508	2034.390	8	19

PCMODEL Queues

Queue	Maximum Contents	Average Contents	Total Entries	Average Time/Cust	Current Contents
RegCk	13	3.5	363	916.969	4
ExpCk	2	0.0	142	7.239	0
Cart	4	0.4	512	0.000	0

GPSS Queues

Queue	Maximum Contents	Average Contents	Total Entries	Average Time/Cust	Current Contents
RegCk	11	1.746	352	613.141	5
ExpCk	2	0.005	153	4.771	0
Cart	1	0.000	508	0.000	0

the movement delays inside the storage were incorporated in the shopping time. The average times for the storage and queues should be accurate, as the movement delays were removed from them before the calculations.

Upon comparison, it is easy to see that for the three non-zero average times, PCModel's times are greater than those of GPSS. The utilization figure for the regular checkout facility is higher than GPSS's, also. Finally, the average contents of the cart storage is greater for PCModel than GPSS, indicating that on the average there were more customers in the store than in the GPSS simulation. Of course, the PCModel average calculation is rather crude, but all of the statistics seem to support it. The operation of the PCModel simulation appears to run at a noticeably slower pace than the GPSS simulation. Several things might explain this. For example, if the random number sequence generated delay times on the high side of the designated average, then the delays would tend to be longer than they were supposed to be. The result would be that the checkout procedures take too long and the queues consequently back up. This in turn would increase the average number of customers in the store, as would higher than defined average shopping times. The results discussed here and those of the warehouse simulation will be considered jointly in the concluding section.

V. CONCLUSIONS

The thrust of this thesis has been to present the PCModel simulation environment by means of examples and then compare the example results to what has gone before. In the warehouse problem it was obvious that the simulation did not behave in the same manner as the GPSS model did; however, the complexity of the two models allowed room for variations in results to occur. The supermarket simulation, on the other hand, was considerably less complicated; as such, the results derived should be more comparable to those for the same model in another language. The statistics formulated for the PCModel system did indeed correspond to those found via GPSS, but the system behaved in a pattern which kept its results different from those of GPSS -- all of the statistics indicated that the PCModel processes were not operating as quickly as GPSS's. In other words, delays and processing times appeared to be taking longer than they should. By itself, this observation might be attributed to the variations necessary to handle the problem under PCModel. When taken in conjunction with the results obtained for the warehouse simulation, a pattern seems to be forming. Both models behaved in a manner that leads to some degree of congestion. Further examples and analysis will be required to determine if this is indeed the case. If so, there may be some steps to take to adjust for the discrepancies. Again, these would be determined from further study.

Concerning the programming environment and language itself, PCModel has been shown to be easy to visualize. The instruction set lends itself to rapid translation from programming ideas to actual program code. This is one of the points Emshoff and Sisson[15] deem as essential: simulation languages should be easy to think in terms of. As far as the user is concerned, the display capability of PCModel eases the verification of the model as correct. Going one step further, presentation of a simulation to someone else, even someone completely unfamiliar with programming of any type, is simplified with PCModel. There is no need for the simulation results to be trusted on the word of the programmer; anyone can observe the simulation in action and verify for themselves that it behaves as it should. Talavage[16] comments on the need to reduce the fear of what is not understood and trusted when dealing with simulations and their results.

In conclusion, PCModel is clearly a step in the right direction for the simulation of systems, as the ease of programming and interactive features indicate. Further study is however warranted to verify that the system behaves as it should for comparison with other languages and to determine what adjustments, if any, can be taken.

BIBLIOGRAPHY

1. Mittra, Sitansu S. "Discrete System Simulation Concepts," *Simulation*, XXXXIII, 3 (September, 1984), 142-144.
2. Stephenson, Robert E. *Computer Simulation for Engineers*. New York: Harcourt Brace Jovanovich, Inc., 1971.
3. Culik, K. "On Simulation Methods and on Some Characteristics of Simulation Languages," *Simulation Programming Languages*, pp 234-247. Amsterdam: North-Holland Publishing Company, 1968.
4. White, David A. *PCModel: Personal Computer Screen Graphics Modeling System User's Guide*. 1985.
5. Meier, Robert C. and others. *Simulation in Business and Economics*. New Jersey: Prentice-Hall, Inc., 1969.
6. Pritsker, A. Alan B. and Wilson, James R. "A Survey of Research on the Simulation Startup Problem," *Simulation*, XXXI, 2 (August, 1978), 55-58.
7. Tocher, K. D. *The Art of Simulation*. London: The English Universities Press LTD, 1963.
8. Graybeal, Wayne J. and Pooch, Udo W. *Simulation: Principles and Methods*. Cambridge: Winthrop Publishers, Inc., 1980.
9. Chorafas, Dimitris N. *Systems and Simulation*. New York: Academic Press, 1965.
10. Crosbie, Roy E. and Hay, John L. "ISIM -- A Simulation Language for Microprocessors," *Simulation*, XXXXIII, 3 (September, 1984), 133-136.

11. Licklider, J. C. R. "Interactive Dynamic Modeling," Prospects for Simulation and Simulators of Dynamic Systems, 279-289. New York: Spartan Books, 1967.

12. Jones, Alfred W. "The Design of Interactive Simulations," Record of Proceedings: The 18th Annual Simulation Symposium, 1985.

13. Bobillier, P. A. and others. Simulation with GPSS and GPSSV. New Jersey: Prentice-Hall, Inc., 1976.

14. GPSSV: An Introduction. International Business Machines Corporation, 1979.

15. Emshoff, James R. and Sisson, Roger L. Design and Use of Computer Simulation Models. New York: The Macmillan Company, 1970.

16. Talavage, Joseph J. "Models for the Automatic Factory," Simulation, XXX, 3 (March 1978), 80-84.

VITA

Edward Telley Hammerand was born on November 21, 1961 in Lebanon, Missouri. He received his primary and secondary education in Lebanon. He has received his college education from the University of Missouri-Rolla, in Rolla, Missouri, where he was designated a Chevron Scholar during his last year. He received his Bachelor of Science degree in Computer Science in May 1984.

He has been enrolled in the Graduate School of the University of Missouri-Rolla since August 1984 and has held a Chancellor's Fellowship since that time. He has been a graduate teaching assistant in the Department of Computer Science for the full time of his enrollment.

APPENDIX A

THE AUTOMATED WAREHOUSE PROGRAM

```
M=(200)                                     ; max number of objects to ever
                                            ;  be active at any one time
W=(200)                                     ; init num of objects that may be
                                            ;  active at any given time
S=(600)                                     ; count of symbols to have storage
                                            ;  reserved for

X=(100)                                     ; x dimension of screen
Y=(70)                                      ; y dimension of screen



D=          *****  Simulation of an Automated Warehouse  *****
   A warehouse is a building used for storing products until they are
required.  Goods are kept on standard-sized trays, called pallets.
Movement of pallets means movement of the goods on those pallets.
Shelves or racks are divided into bins, each of which can accomodate one
pallet.  Racks are in turn arranged vertically in corridors.
   The warehouse of this simulation is built to handle pallets of 1x1x1
meter; its bins can hold one pallet each.  The warehouse consists of
three principal parts:  a shipping/receiving bay, the warehouse itself,
consisting of racks of bins, and a conveyor which connects the two.
   The shipping and receiving bay is the interface between the warehouse
and the outside world, which can be thought of as the trucks and
railcars that transport goods to and from the warehouse.  This problem
models one shipping and one receiving bay.
   The warehouse has 10 corridors; each has a stacker crane, which moves
pallets in and out of the 2 adjacent racks.  The length of a corridor is
50 m and the height 10 m; thus a rack has 500 bins in which 500 pallets
can be stored.  Each crane has access to 1000 bins (left and right
track) and the total capacity of the 10 corridors is 10,000 pallets.
   The conveyor connects the receiving and shipping bays with the racks.
It is continuous and designed on 2 levels to move the pallets between
the corridors on the upper level and the shipping/receiving bays on the
lower level.  The connections between levels are made by 2 elevators.
   The problem is to simulate the operation of the warehouse to check if
it can operate satisfactorily, especially during peak hours.  $


O=(=)                                       ; overlay kept in WH.OLY

V=(XY(1,46))                                ; viewing-window location



                                            ; Symbol Definitions
```

```
@RECVRATE=(60)                        ; rate for recvd goods (sec per)
@SHIPRATE=(120)                       ; rate for shp requests (sec per)

@HORZRATE=(12)                        ; crane horizontal rate (sec/m)
@VERTRATE=(120)                       ; crane vertical rate (sec/m)

@RNDMCORR=(0)                         ; random corridor number
@RNDMZONE=(0)                         ; random zone value

@COLOR=(7)                            ; background color for objects
@BUFCAP=(4)                           ; capacity of corr input buffers
@GO=(0)                               ; conveyor synchronization
@TEMP=(0)                             ; work variable used in links

                                      ; variables for num of objects on
                                      ;  input conveyors
@INBUF1=(0)
@INBUF2=(0)
@INBUF3=(0)
@INBUF4=(0)
@INBUF5=(0)
@INBUF6=(0)
@INBUF7=(0)
@INBUF8=(0)
@INBUF9=(0)
@INBUF0=(0)
                                      ; num of objs waiting to be placed
                                      ;  by the crane in the corridor
@RECRQ1=(0)
@RECRQ2=(0)
@RECRQ3=(0)
@RECRQ4=(0)
@RECRQ5=(0)
@RECRQ6=(0)
@RECRQ7=(0)
@RECRQ8=(0)
@RECRQ9=(0)
@RECRQ0=(0)
                                      ; num of objs the crane has taken
                                      ;  from those waiting to be put up
@OKRECRQ1=(0)
@OKRECRQ2=(0)
@OKRECRQ3=(0)
@OKRECRQ4=(0)
@OKRECRQ5=(0)
@OKRECRQ6=(0)
@OKRECRQ7=(0)
@OKRECRQ8=(0)
@OKRECRQ9=(0)
@OKRECRQ0=(0)
                                      ; num of shipping objs that are
                                      ;  waiting to be brought from
                                      ;  their bins to the conveyor
@SHPRQ1=(0)
```

```
@SHPRQ2=(0)
@SHPRQ3=(0)
@SHPRQ4=(0)
@SHPRQ5=(0)
@SHPRQ6=(0)
@SHPRQ7=(0)
@SHPRQ8=(0)
@SHPRQ9=(0)
@SHPRQ0=(0)
                                    ; num of shipping objs the crane
                                    ;  has brought to the conveyor and
                                    ;  thus may be released onto it
@OKSHPRQ1=(0)
@OKSHPRQ2=(0)
@OKSHPRQ3=(0)
@OKSHPRQ4=(0)
@OKSHPRQ5=(0)
@OKSHPRQ6=(0)
@OKSHPRQ7=(0)
@OKSHPRQ8=(0)
@OKSHPRQ9=(0)
@OKSHPRQ0=(0)
                                    ; work variables for quantities
                                    ;  in zones
@ZONEA=(0)
@ZONEB=(0)
@ZONEC=(0)
@ZONED=(0)
                                    ; corridor 1 zone quantities
@ZONE1A=(0)
@ZONE1B=(0)
@ZONE1C=(0)
@ZONE1D=(0)
                                    ; corridor 2 zone quantities
@ZONE2A=(0)
@ZONE2B=(0)
@ZONE2C=(0)
@ZONE2D=(0)
                                    ; corridor 3 zone quantities
@ZONE3A=(0)
@ZONE3B=(0)
@ZONE3C=(0)
@ZONE3D=(0)
                                    ; corridor 4 zone quantities
@ZONE4A=(0)
@ZONE4B=(0)
@ZONE4C=(0)
@ZONE4D=(0)
                                    ; corridor 5 zone quantities
@ZONE5A=(0)
@ZONE5B=(0)
@ZONE5C=(0)
@ZONE5D=(0)
```

```
                                        ; corridor 6 zone quantities
@ZONE6A=(0)
@ZONE6B=(0)
@ZONE6C=(0)
@ZONE6D=(0)
                                        ; corridor 7 zone quantities
@ZONE7A=(0)
@ZONE7B=(0)
@ZONE7C=(0)
@ZONE7D=(0)
                                        ; corridor 8 zone quantities
@ZONE8A=(0)
@ZONE8B=(0)
@ZONE8C=(0)
@ZONE8D=(0)
                                        ; corridor 9 zone quantities
@ZONE9A=(0)
@ZONE9B=(0)
@ZONE9C=(0)
@ZONE9D=(0)
                                        ; corridor 10 zone quantities
@ZONE0A=(0)
@ZONE0B=(0)
@ZONE0C=(0)
@ZONE0D=(0)

#SEED=(9997)                            ; random number sequence seed



                                        ; Job Descriptions

J=(1,X,1,0,0,0,1)                       ; initialization

J=(2,#,2,0,0,1,1)                       ; synchronization of conveyor belt

J=(3,R,3,0,0,2,5000)                    ; receiving items
J=(4,S,4,0,0,2,5000)                    ; shipping items

J=(11,1,11,0,0,2,1)                     ; Crane # 1
J=(12,2,12,0,0,2,1)                     ; Crane # 2
J=(13,3,13,0,0,2,1)                     ; Crane # 3
J=(14,4,14,0,0,2,1)                     ; Crane # 4
J=(15,5,15,0,0,2,1)                     ; Crane # 5
J=(16,6,16,0,0,2,1)                     ; Crane # 6
J=(17,7,17,0,0,2,1)                     ; Crane # 7
J=(18,8,18,0,0,2,1)                     ; Crane # 8
J=(19,9,19,0,0,2,1)                     ; Crane # 9
J=(20,0,20,0,0,2,1)                     ; Crane # 0
```

```
                                     ; Utilization Locations

U=(1,REC 1,XY(8,47))                 ; Crane # 1
U=(2,SHP 1,XY(8,49))

U=(3,REC 2,XY(15,47))                ; Crane # 2
U=(4,SHP 2,XY(15,49))

U=(5,REC 3,XY(22,47))                ; Crane # 3
U=(6,SHP 3,XY(22,49))

U=(7,REC 4,XY(29,47))                ; Crane # 4
U=(8,SHP 4,XY(29,49))

U=(9,REC 5,XY(36,47))                ; Crane # 5
U=(10,SHP 5,XY(36,49))

U=(11,REC 6,XY(43,47))               ; Crane # 6
U=(12,SHP 6,XY(43,49))

U=(13,REC 7,XY(50,47))               ; Crane # 7
U=(14,SHP 7,XY(50,49))

U=(15,REC 8,XY(57,47))               ; Crane # 8
U=(16,SHP 8,XY(57,49))

U=(17,REC 9,XY(64,47))               ; Crane # 9
U=(18,SHP 9,XY(64,49))

U=(19,REC 0,XY(71,47))               ; Crane # 0
U=(20,SHP 0,XY(71,49))



                                     ; Link to determine zone
                                     ;  for receiving
         BL(!RECVZONE)
                                     ; OBJ@1:  unaltered (work variable)
                                     ; OBJ@2:  full-zone flag
                                     ; OBJ@3:  # in the selected zone
                                     ; OBJ@4:  offset to zone
                                     ; OBJ@5:  unaltered (h. position)
                                     ; OBJ@6:  unaltered (v. position)

         SV(OBJ@2,0)                 ; clear full flag

         RV(@RNDMZONE,1,100)         ; generate random zone value
         IF(@RNDMZONE,LT,41,:RZONEA)  ; 40% = ( 1, 40) so Zone A
         IF(@RNDMZONE,LT,71,:RZONEB)  ; 30% = (41, 70) so Zone B
         IF(@RNDMZONE,LT,91,:RZONEC)  ; 20% = (71, 90) so Zone C
         JP(:RZONED)                  ; 10% = (91,100) so Zone D

:RZONEA  IF(@ZONEA,EQ,260,:ZONEFULL)  ; if zone A is full, go set flag
```

```
          SV(OBJ@3,@ZONEA)            ; ZONEA is the # in zone A of corr
          IV(@ZONEA)                  ; increment for storage
          SV(OBJ@4,0)                 ; set offset for zone A
          JP(:OKSTORE)                ; go to store the pallet

:RZONEB   IF(@ZONEB,EQ,240,:ZONEFULL)  ; if zone B is full, go set flag
          SV(OBJ@3,@ZONEB)            ; ZONEB is the # in zone B of corr
          IV(@ZONEB)                  ; increment for storage
          SV(OBJ@4,13)                ; set offset for zone B
          JP(:OKSTORE)                ; go to store the pallet

:RZONEC   IF(@ZONEC,EQ,260,:ZONEFULL)  ; if zone C is full, go set flag
          SV(OBJ@3,@ZONEC)            ; ZONEC is the # in zone C of corr
          IV(@ZONEC)                  ; increment for storage
          SV(OBJ@4,25)                ; set offset for zone C
          JP(:OKSTORE)                ; go to store the pallet

:RZONED   IF(@ZONED,EQ,240,:ZONEFULL)  ; if zone D is full, go set flag
          SV(OBJ@3,@ZONED)            ; ZONED is the # in zone D of corr
          IV(@ZONED)                  ; increment for storage
          SV(OBJ@4,38)                ; set offset for zone D
          JP(:OKSTORE)                ; go to store the pallet

:ZONEFULL SV(OBJ@2,1)                 ; OBJ@2 is set if the zone was full

                                      ; At this point, it is known (by
                                      ;  OBJ@4) which zone the crane is
                                      ;  headed for & (by OBJ@3) how
                                      ;  many are in the zone
:OKSTORE  EL



                                      ; Link to move crane for receiving
          BL(!RECVMOVE)
                                      ; OBJ@1:  work variable
                                      ; OBJ@2:  work variable
                                      ; OBJ@3:  # in zone
                                      ; OBJ@4:  offset to zone
                                      ; OBJ@5:  crane horizontal position
                                      ; OBJ@6:  crane vertical position

                                      ; First, the crane must move from
                                      ; current pos to the input position
                                      ; (H=1, V=1) to pick up the item

          SV(OBJ@1,OBJ@5)             ; copy the horizontal position
          DV(OBJ@1)                   ; decrement for horizontal distance
          AO(OBJ@1,*,@HORZRATE)       ; multiply distance by sec/meter
          ST(OBJ@1)                   ; time to move horizontally
          SV(OBJ@5,1)                 ; set the new horizontal position
SV(OBJ@1,OBJ@6)           ; copy the vertical position
DV(OBJ@1)                 ; decrement for vertical distance
AO(OBJ@1,*,@VERTRATE)     ; multiply distance by sec/meter
ST(OBJ@1)                 ; time to move vertically
SV(OBJ@6,1)               ; set the new vertical position

                          ; With the crane at the input pos
                          ; at the conveyor belt, the item
                          ; can be picked up and moved to
                          ; its bin

                          ; First, move horizontally
                          ;  strategy = determine number in
                          ;  horz row to move by determining
                          ;  num of whole 20's in the zone

SV(OBJ@1,OBJ@3)           ; # in the zone being examined
AO(OBJ@1,/,20)            ; integer division by # in each m
AO(OBJ@1,+,OBJ@4)         ; add zone offset
SV(OBJ@2,OBJ@1)           ; OBJ@ 1 & 2 are distance to move
AO(OBJ@2,+,1)             ; add 1 for horizontal position

AO(OBJ@1,*,@HORZRATE)     ; multiply distance by sec/meter
ST(OBJ@1)                 ; time to move horizontally
SV(OBJ@5,OBJ@2)           ; set the new horizontal position

                          ; Second, move vertically
                          ;  strategy = determine num in vert
                          ;  column to move by removing whole
                          ;  20's (horiz) & then dividing
                          ;  remaining quantity by # per m
                          ;  vertically (2)

SV(OBJ@1,OBJ@3)           ; # in the zone being examined
AO(OBJ@1,/,20)            ; integer division by #/m horz
AO(OBJ@1,*,20)            ; mult. for greatest multiple of 20
SV(OBJ@2,OBJ@3)           ; # in the zone again
AO(OBJ@2,-,OBJ@1)         ; get the remainder after 20's gone
AO(OBJ@2,/,2)             ; integer division by #/m vert
SV(OBJ@1,OBJ@2)           ; copy this distance
AO(OBJ@2,+,1)             ; add 1 for vertical position

AO(OBJ@1,*,@VERTRATE)     ; multiply distance by sec/meter
ST(OBJ@1)                 ; time to move vertically
SV(OBJ@6,OBJ@2)           ; set the new vertical position

EL



                          ; Link to determine zone for
                          ;  shipping
```

```
          BL(!SHIPZONE)
                                         ; OBJ@1:  unaltered (work variable)
                                         ; OBJ@2:  empty-zone flag
                                         ; OBJ@3:  # in the selected zone
                                         ; OBJ@4:  offset to zone
                                         ; OBJ@5:  unaltered (h. position)
                                         ; OBJ@6:  unaltered (v. position)

          SV(OBJ@2,0)                    ; clear zone empty flag

          RV(@RNDMZONE,1,100)            ; generate random zone value
          IF(@RNDMZONE,LT,41,:SZONEA)     ; 40% = ( 1, 40) SO ZONE A
          IF(@RNDMZONE,LT,71,:SZONEB)     ; 30% = (41, 70) SO ZONE B
          IF(@RNDMZONE,LT,91,:SZONEC)     ; 20% = (71, 90) SO ZONE C
          JP(:SZONED)                     ; 10% = (91,100) SO ZONE D

:SZONEA   IF(@ZONEA,EQ,0,:ZONEMPTY)      ; if no pallets in zone A, set flag
          SV(OBJ@3,@ZONEA)               ; ZONEA is the # in the zone
          DV(@ZONEA)                     ; decrement for retrieval
          SV(OBJ@4,0)                    ; set offset for zone A
          JP(:OKGET)                     ; go get the pallet

:SZONEB   IF(@ZONEB,EQ,0,:ZONEMPTY)      ; if no pallets in zone B, set flag
          SV(OBJ@3,@ZONEB)               ; ZONEB is the # in the zone
          DV(@ZONEB)                     ; decrement for retrieval
          SV(OBJ@4,13)                   ; set offset for zone B
          JP(:OKGET)                     ; go get the pallet

:SZONEC   IF(@ZONEC,EQ,0,:ZONEMPTY)      ; if no pallets in zone C, set flag
          SV(OBJ@3,@ZONEC)               ; ZONEC is the # in the zone
          DV(@ZONEC)                     ; decrement for retrieval
          SV(OBJ@4,25)                   ; set offset for zone C
          JP(:OKGET)                     ; go get the pallet

:SZONED   IF(@ZONED,EQ,0,:ZONEMPTY)      ; if no pallets in zone D, set flag
          SV(OBJ@3,@ZONED)               ; ZONED is the # in the zone
          DV(@ZONED)                     ; decrement for retrieval
          SV(OBJ@4,38)                   ; set offset for zone D
          JP(:OKGET)                     ; go get the pallet

:ZONEMPTY SV(OBJ@2,1)                    ; set OBJ@2 if the zone was empty

                                         ; At this point, it is known (by
                                         ;  OBJ@4) which zone the crane is
                                         ;  headed for & (by OBJ@3) how many
                                         ;  are in the zone
:OKGET    EL



                                         ; Link to move crane for shipping
          BL(!SHIPMOVE)
```

```
                                 ; OBJ@1:  work variable
                                 ; OBJ@2:  work variable
                                 ; OBJ@3:  # in zone
                                 ; OBJ@4:  offset to zone
                                 ; OBJ@5:  crane horizontal position
                                 ; OBJ@6:  crane vertical position

                                 ; First, move horizontally
        SV(OBJ@1,OBJ@3)          ; # in the zone being examined
        AO(OBJ@1,/,20)           ; integer division by # in each m
        AO(OBJ@1,+,OBJ@4)        ; add zone offset
        AO(OBJ@1,+,1)            ; add 1 for horizontal position
        SV(OBJ@2,OBJ@1)          ; OBJ@2 is now horiz pos desired

        IF(OBJ@5,GT,OBJ@1,:DOWN1) ; if current pos > new pos, jump
        AO(OBJ@1,-,OBJ@5)        ; get diff of positions in OBJ@1
        JP(:DOWN2)

:DOWN1  SV(@TEMP,OBJ@5)          ; copy to avoid altering OBJ@5
        AO(@TEMP,-,OBJ@1)        ; get difference of positions
        SV(OBJ@1,@TEMP)          ; put it in OBJ@1

:DOWN2  AO(OBJ@1,*,@HORZRATE)    ; multiply distance by sec/meter
        ST(OBJ@1)                ; time to move horizontally
        SV(OBJ@5,OBJ@2)          ; set the new horizontal position

                                 ; Second, move vertically
                                 ;  strategy = determine num in vert
                                 ;  column to move by removing whole
                                 ;  20's (horiz) & dividing
                                 ;  remaining quantity by # per m
                                 ;  vertically (2)

        SV(OBJ@1,OBJ@3)          ; # in the zone being examined
        AO(OBJ@1,/,20)           ; integer division by # in each m
        AO(OBJ@1,*,20)           ; mult. for greatest multiple of 20
        SV(OBJ@2,OBJ@3)          ; # in the zone again
        AO(OBJ@2,-,OBJ@1)        ; get the remainder after 20's gone
        AO(OBJ@2,/,2)            ; integer division by #/m vert
        AO(OBJ@2,+,1)            ; add 1 for vertical position
        SV(OBJ@1,OBJ@2)          ; copy pos for updating parameter

        IF(OBJ@6,GT,OBJ@1,:DOWN3) ; if current pos. > new pos., jump
        AO(OBJ@1,-,OBJ@6)        ; get diff of positions in OBJ@1
        JP(:DOWN4)

:DOWN3  SV(@TEMP,OBJ@6)          ; copy to prevent altering
        AO(@TEMP,-,OBJ@1)        ; get difference of positions
        SV(OBJ@1,@TEMP)          ; put it in OBJ@1

:DOWN4  AO(OBJ@1,*,@VERTRATE)    ; multiply distance by sec/meter
        ST(OBJ@1)                ; time to move vertically
        SV(OBJ@6,OBJ@2)          ; set the new vertical position
```

```
                              ; At this point, the crane has the
                              ;  item & must move to the exit
                              ;  pos (H=1, V=1) at conveyor belt

SV(OBJ@1,OBJ@5)               ; copy parameter to avoid altering
DV(OBJ@1)                     ; decrement for horizontal distance
AO(OBJ@1,*,@HORZRATE)         ; multiply distance by sec/meter
ST(OBJ@1)                     ; time to move horizontally
SV(OBJ@5,1)                   ; set horizontal position at floor

SV(OBJ@1,OBJ@6)               ; copy parameter to avoid altering
DV(OBJ@1)                     ; decrement for vertical distance
AO(OBJ@1,*,@VERTRATE)         ; multiply distance by sec/meter
ST(OBJ@1)                     ; time to move vertically
SV(OBJ@6,1)                   ; set vertical position at floor

EL



                              ; Job which initializes
                              ;  the simulation
BR(1,XY(60,68),0)

RS(#SEED)                     ; init random number sequence
SV(@GO,0)                     ; set conveyor synchronization
VW(XY(1,46))                  ; set initial window position

                              ; init variables defined above
SV(@INBUF1,0)
SV(@INBUF2,0)
SV(@INBUF3,0)
SV(@INBUF4,0)
SV(@INBUF5,0)
SV(@INBUF6,0)
SV(@INBUF7,0)
SV(@INBUF8,0)
SV(@INBUF9,0)
SV(@INBUF0,0)

SV(@RECRQ1,0)
SV(@RECRQ2,0)
SV(@RECRQ3,0)
SV(@RECRQ4,0)
SV(@RECRQ5,0)
SV(@RECRQ6,0)
SV(@RECRQ7,0)
SV(@RECRQ8,0)
SV(@RECRQ9,0)
SV(@RECRQ0,0)

SV(@OKRECRQ1,0)
SV(@OKRECRQ2,0)
SV(@OKRECRQ3,0)
SV(@OKRECRQ4,0)
SV(@OKRECRQ5,0)
SV(@OKRECRQ6,0)
SV(@OKRECRQ7,0)
SV(@OKRECRQ8,0)
SV(@OKRECRQ9,0)
SV(@OKRECRQ0,0)

SV(@SHPRQ1,0)
SV(@SHPRQ2,0)
SV(@SHPRQ3,0)
SV(@SHPRQ4,0)
SV(@SHPRQ5,0)
SV(@SHPRQ6,0)
SV(@SHPRQ7,0)
SV(@SHPRQ8,0)
SV(@SHPRQ9,0)
SV(@SHPRQ0,0)

SV(@OKSHPRQ1,0)
SV(@OKSHPRQ2,0)
SV(@OKSHPRQ3,0)
SV(@OKSHPRQ4,0)
SV(@OKSHPRQ5,0)
SV(@OKSHPRQ6,0)
SV(@OKSHPRQ7,0)
SV(@OKSHPRQ8,0)
SV(@OKSHPRQ9,0)
SV(@OKSHPRQ0,0)

SV(@ZONE1A,0)
SV(@ZONE1B,0)
SV(@ZONE1C,0)
SV(@ZONE1D,0)

SV(@ZONE2A,0)
SV(@ZONE2B,0)
SV(@ZONE2C,0)
SV(@ZONE2D,0)

SV(@ZONE3A,0)
SV(@ZONE3B,0)
SV(@ZONE3C,0)
SV(@ZONE3D,0)

SV(@ZONE4A,0)
SV(@ZONE4B,0)
SV(@ZONE4C,0)
SV(@ZONE4D,0)

SV(@ZONE5A,0)
```

```
           SV(@ZONE5B,0)
           SV(@ZONE5C,0)
           SV(@ZONE5D,0)

           SV(@ZONE6A,0)
           SV(@ZONE6B,0)
           SV(@ZONE6C,0)
           SV(@ZONE6D,0)

           SV(@ZONE7A,0)
           SV(@ZONE7B,0)
           SV(@ZONE7C,0)
           SV(@ZONE7D,0)

           SV(@ZONE8A,0)
           SV(@ZONE8B,0)
           SV(@ZONE8C,0)
           SV(@ZONE8D,0)

           SV(@ZONE9A,0)
           SV(@ZONE9B,0)
           SV(@ZONE9C,0)
           SV(@ZONE9D,0)

           SV(@ZONE0A,0)
           SV(@ZONE0B,0)
           SV(@ZONE0C,0)
           SV(@ZONE0D,0)

           ER



                                  ; Job which continuously sets
                                  ;  "GO" with each clock second
           BR(2,XY(65,68),0)

:BACK      IV(@GO)                ; increment to next value on (0,11)
           IF(@GO,LT,12,:OVER)    ; on (0,11), "GO" is valid
           SV(@GO,0)              ; otherwise, reset sequence

:OVER      ST(1)                  ; pause with value for 1 second
           JP(:BACK)              ; repeat each second
           ER



                                  ; Receiving

           BR(3,XY(44,69),@RECVRATE)  ; items arrive at bottom of screen
```

```
          MU(9,6)                     ; move up to conveyor belt

:WAIT     DN
          IF(@GO,EQ,0,:READY)         ; wait until synchronized with conv
          JP(:WAIT)
                                      ; wait if space on conv is occupied
:READY    JB(2,XY(44,59),XY(45,59),:WAIT)
                                      ; space open
          RV(@RNDMCORR,1,10)          ; select corridor number
          SA(@COLOR,@RNDMCORR)        ; set obj color to match corridor
                                      ; branch to selected corridor
          IF(@RNDMCORR,EQ,1,:CORR1)
          IF(@RNDMCORR,EQ,2,:CORR2)
          IF(@RNDMCORR,EQ,3,:CORR3)
          IF(@RNDMCORR,EQ,4,:CORR4)
          IF(@RNDMCORR,EQ,5,:CORR5)
          IF(@RNDMCORR,EQ,6,:CORR6)
          IF(@RNDMCORR,EQ,7,:CORR7)
          IF(@RNDMCORR,EQ,8,:CORR8)
          IF(@RNDMCORR,EQ,9,:CORR9)
          JP(:CORR0)

:CORR1    R=(XY(44,60))
          MU(1,0)                     ; move onto conveyor
          MR(30,3)                    ; move to right edge of conveyor

:CONT1    MU(4,6)                     ; move to upper level of conveyor
          ML(65,3)                    ; move to corridor # 1
          IF(@INBUF1,LT,@BUFCAP,:INPUT1) ; if room on input conveyor,
                                         ;  leave main belt
          ML(3,3)                     ; else go around again
          MD(4,6)
          MR(68,3)
          JP(:CONT1)

:INPUT1   R=(XY(9,55))
          IV(@INBUF1)                 ; inc # on input conveyor
          MU(4,6)                     ; move up input conveyor
          IV(@RECRQ1)                 ; update count waiting on crane

:BACK1    DN
          IF(@OKRECRQ1,EQ,0,:BACK1)   ; wait for # picked up to inc
          DV(@OKRECRQ1)               ; decrement # picked up by crane
          DV(@RECRQ1)                 ; decrement # waiting for crane
          DV(@INBUF1)                 ; decrement # on input conveyor
          JP(:INDONE)                 ; leave simulation


:CORR2    R=(XY(44,60))               ; See comments for Corridor # 1
          MU(1,0)
          MR(30,3)

:CONT2    MU(4,6)
```

```
          ML(58,3)
          IF(@INBUF2,LT,@BUFCAP,:INPUT2)
          ML(10,3)
          MD(4,6)
          MR(68,3)
          JP(:CONT2)

:INPUT2   R=(XY(16,55))
          IV(@INBUF2)
          MU(4,6)
          IV(@RECRQ2)

:BACK2    DN
          IF(@OKRECRQ2,EQ,0,:BACK2)
          DV(@OKRECRQ2)
          DV(@RECRQ2)
          DV(@INBUF2)
          JP(:INDONE)


:CORR3    R=(XY(44,60))           ; See comments for Corridor # 1
          MU(1,0)
          MR(30,3)

:CONT3    MU(4,6)
          ML(51,3)
          IF(@INBUF3,LT,@BUFCAP,:INPUT3)
          ML(17,3)
          MD(4,6)
          MR(68,3)
          JP(:CONT3)

:INPUT3   R=(XY(23,55))
          IV(@INBUF3)
          MU(4,6)
          IV(@RECRQ3)

:BACK3    DN
          IF(@OKRECRQ3,EQ,0,:BACK3)
          DV(@OKRECRQ3)
          DV(@RECRQ3)
          DV(@INBUF3)
          JP(:INDONE)


:CORR4    R=(XY(44,60))           ; See comments for Corridor # 1
          MU(1,0)
          MR(30,3)

:CONT4    MU(4,6)
          ML(44,3)
          IF(@INBUF4,LT,@BUFCAP,:INPUT4)
          ML(24,3)
          MD(4,6)
          MR(68,3)
          JP(:CONT4)

:INPUT4   R=(XY(30,55))
          IV(@INBUF4)
          MU(4,6)
          IV(@RECRQ4)

:BACK4    DN
          IF(@OKRECRQ4,EQ,0,:BACK4)
          DV(@OKRECRQ4)
          DV(@RECRQ4)
          DV(@INBUF4)
          JP(:INDONE)


:CORR5    R=(XY(44,60))              ; See comments for Corridor # 1
          MU(1,0)
          MR(30,3)

:CONT5    MU(4,6)
          ML(37,3)
          IF(@INBUF5,LT,@BUFCAP,:INPUT5)
          ML(31,3)
          MD(4,6)
          MR(68,3)
          JP(:CONT5)

:INPUT5   R=(XY(37,55))
          IV(@INBUF5)
          MU(4,6)
          IV(@RECRQ5)

:BACK5    DN
          IF(@OKRECRQ5,EQ,0,:BACK5)
          DV(@OKRECRQ5)
          DV(@RECRQ5)
          DV(@INBUF5)
          JP(:INDONE)


:CORR6    R=(XY(44,60))              ; See comments for Corridor # 1
          MU(1,0)
          MR(30,3)

:CONT6    MU(4,6)
          ML(30,3)
          IF(@INBUF6,LT,@BUFCAP,:INPUT6)
          ML(38,3)
          MD(4,6)
          MR(68,3)
          JP(:CONT6)
```

```
:INPUT6    R=(XY(44,55))
           IV(@INBUF6)
           MU(4,6)
           IV(@RECRQ6)

:BACK6     DN
           IF(@OKRECRQ6,EQ,0,:BACK6)
           DV(@OKRECRQ6)
           DV(@RECRQ6)
           DV(@INBUF6)
           JP(:INDONE)


:CORR7     R=(XY(44,60))              ; See comments for Corridor # 1
           MU(1,0)
           MR(30,3)

:CONT7     MU(4,6)
           ML(23,3)
           IF(@INBUF7,LT,@BUFCAP,:INPUT7)
           ML(45,3)
           MD(4,6)
           MR(68,3)
           JP(:CONT7)

:INPUT7    R=(XY(51,55))
           IV(@INBUF7)
           MU(4,6)
           IV(@RECRQ7)

:BACK7     DN
           IF(@OKRECRQ7,EQ,0,:BACK7)
           DV(@OKRECRQ7)
           DV(@RECRQ7)
           DV(@INBUF7)
           JP(:INDONE)


:CORR8     R=(XY(44,60))              ; See comments for Corridor # 1
           MU(1,0)
           MR(30,3)

:CONT8     MU(4,6)
           ML(16,3)
           IF(@INBUF8,LT,@BUFCAP,:INPUT8)
           ML(52,3)
           MD(4,6)
           MR(68,3)
           JP(:CONT8)

:INPUT8    R=(XY(58,55))
           IV(@INBUF8)
           MU(4,6)
```

```
          IV(@RECRQ8)

:BACK8    DN
          IF(@OKRECRQ8,EQ,0,:BACK8)
          DV(@OKRECRQ8)
          DV(@RECRQ8)
          DV(@INBUF8)
          JP(:INDONE)


:CORR9    R=(XY(44,60))              ; See comments for Corridor # 1
          MU(1,0)
          MR(30,3)

:CONT9    MU(4,6)
          ML(9,3)
          IF(@INBUF9,LT,@BUFCAP,:INPUT9)
          ML(59,3)
          MD(4,6)
          MR(68,3)
          JP(:CONT9)

:INPUT9   R=(XY(65,55))
          IV(@INBUF9)
          MU(4,6)
          IV(@RECRQ9)

:BACK9    DN
          IF(@OKRECRQ9,EQ,0,:BACK9)
          DV(@OKRECRQ9)
          DV(@RECRQ9)
          DV(@INBUF9)
          JP(:INDONE)


:CORR0    R=(XY(44,60))              ; See comments for Corridor # 1
          MU(1,0)
          MR(30,3)

:CONT0    MU(4,6)
          ML(2,3)
          IF(@INBUF0,LT,@BUFCAP,:INPUT0)
          ML(66,3)
          MD(4,6)
          MR(68,3)
          JP(:CONT0)

:INPUT0   R=(XY(72,55))
          IV(@INBUF0)
          MU(4,6)
          IV(@RECRQ0)

:BACK0    DN
```

```
          IF(@OKRECRQ0,EQ,0,:BACK0)
          DV(@OKRECRQ0)
          DV(@RECRQ0)
          DV(@INBUF0)
          JP(:INDONE)


:INDONE   ER




                                    ; Shipping
          BR(4,XY(34,69),@SHIPRATE)

:REPEAT   DN
          RV(@RNDMCORR,1,10)        ; select corridor number
          SA(@COLOR,@RNDMCORR)      ; set obj color to match corridor
                                    ; branch to selected corridor
          IF(@RNDMCORR,EQ,1,:EXIT1)
          IF(@RNDMCORR,EQ,2,:EXIT2)
          IF(@RNDMCORR,EQ,3,:EXIT3)
          IF(@RNDMCORR,EQ,4,:EXIT4)
          IF(@RNDMCORR,EQ,5,:EXIT5)
          IF(@RNDMCORR,EQ,6,:EXIT6)
          IF(@RNDMCORR,EQ,7,:EXIT7)
          IF(@RNDMCORR,EQ,8,:EXIT8)
          IF(@RNDMCORR,EQ,9,:EXIT9)
          JP(:EXIT0)


:EXIT1    R=(XY(7,47))
          JB(1,XY(7,47),:REPEAT)    ; if the shp request lane for the
                                    ;  corr is full, try a diff corr
          MA(XY(7,47),0)            ; else go to the corridor
          MD(3,0)                   ; move down request lane
          DN                        ; pause for output buffer check
          TP(1,XY(7,53))            ; wait here if output buffer full
          IV(@SHPRQ1)               ; inc the # of shipping requests

:HOLD1    DN
          IF(@OKSHPRQ1,EQ,0,:HOLD1) ; wait for # brought to conveyor by
                                    ;  crane to increment
          DV(@OKSHPRQ1)             ; decrement # brought to conveyor
          DV(@SHPRQ1)               ; decrement # of shipping requests
          MD(4,0)                   ; move down output buffer

:WAIT1    DN
          IF(@GO,EQ,3,:READY1)      ; wait until synchronized with conv
          JP(:WAIT1)

:READY1   R=(XY(7,54))
          JB(2,XY(7,55),XY(6,55),:WAIT1) ; wait if space on conv taken
          MD(1,0)                    ; move on to conveyor
          ML(1,3)                    ; go to left edge of conveyor
          MD(4,6)                    ; move down to lower level
          MR(29,3)                   ; move right to shipping conveyor
          MD(10,6)                   ; move down shipping conveyor
          JP(:OUTDONE)               ; leave simulation


:EXIT2    R=(XY(14,47))              ; See comments for Exit # 1
          JB(1,XY(14,47),:REPEAT)
          MA(XY(14,47),0)
          MD(3,0)
          DN
          TP(1,XY(14,53))
          IV(@SHPRQ2)
:HOLD2    DN
          IF(@OKSHPRQ2,EQ,0,:HOLD2)
          DV(@OKSHPRQ2)
          DV(@SHPRQ2)
          MD(4,0)

:WAIT2    DN
          IF(@GO,EQ,6,:READY2)
          JP(:WAIT2)

:READY2   R=(XY(14,54))
          JB(2,XY(14,55),XY(13,55),:WAIT2)
          MD(1,0)
          ML(8,3)
          MD(4,6)
          MR(29,3)
          MD(10,6)
          JP(:OUTDONE)


:EXIT3    R=(XY(21,47))              ; See comments for Exit # 1
          JB(1,XY(21,47),:REPEAT)
          MA(XY(21,47),0)
          MD(3,0)
          DN
          TP(1,XY(21,53))
          IV(@SHPRQ3)
:HOLD3    DN
          IF(@OKSHPRQ3,EQ,0,:HOLD3)
          DV(@OKSHPRQ3)
          DV(@SHPRQ3)
          MD(4,0)

:WAIT3    DN
          IF(@GO,EQ,9,:READY3)
          JP(:WAIT3)

:READY3   R=(XY(21,54))
```

```
        JB(2,XY(21,55),XY(20,55),:WAIT3)
        MD(1,0)
        ML(15,3)
        MD(4,6)
        MR(29,3)
        MD(10,6)
        JP(:OUTDONE)


:EXIT4  R=(XY(28,47))              ; See comments for Exit # 1
        JB(1,XY(28,47),:REPEAT)
        MA(XY(28,47),0)
        MD(3,0)
        DN
        TP(1,XY(28,53))
        IV(@SHPRQ4)
:HOLD4  DN
        IF(@OKSHPRQ4,EQ,0,:HOLD4)
        DV(@OKSHPRQ4)
        DV(@SHPRQ4)
        MD(4,0)

:WAIT4  DN
        IF(@GO,EQ,0,:READY4)
        JP(:WAIT4)

:READY4 R=(XY(28,54))
        JB(2,XY(28,55),XY(27,55),:WAIT4)
        MD(1,0)
        ML(22,3)
        MD(4,6)
        MR(29,3)
        MD(10,6)
        JP(:OUTDONE)


:EXIT5  R=(XY(35,47))              ; See comments for Exit # 1
        JB(1,XY(35,47),:REPEAT)
        MA(XY(35,47),0)
        MD(3,0)
        DN
        TP(1,XY(35,53))
        IV(@SHPRQ5)
:HOLD5  DN
        IF(@OKSHPRQ5,EQ,0,:HOLD5)
        DV(@OKSHPRQ5)
        DV(@SHPRQ5)
        MD(4,0)

:WAIT5  DN
        IF(@GO,EQ,3,:READY5)
        JP(:WAIT5)

:READY5 R=(XY(35,54))
```

```
          JB(2,XY(35,55),XY(34,55),:WAIT5)
          MD(1,0)
          ML(29,3)
          MD(4,6)
          MR(29,3)
          MD(10,6)
          JP(:OUTDONE)


:EXIT6    R=(XY(42,47))              ; See comments for Exit # 1
          JB(1,XY(42,47),:REPEAT)
          MA(XY(42,47),0)
          MD(3,0)
          DN
          TP(1,XY(42,53))
          IV(@SHPRQ6)
:HOLD6    DN
          IF(@OKSHPRQ6,EQ,0,:HOLD6)
          DV(@OKSHPRQ6)
          DV(@SHPRQ6)
          MD(4,0)

:WAIT6    DN
          IF(@GO,EQ,6,:READY6)
          JP(:WAIT6)

:READY6   R=(XY(42,54))
          JB(2,XY(42,55),XY(41,55),:WAIT6)
          MD(1,0)
          ML(36,3)
          MD(4,6)
          MR(29,3)
          MD(10,6)
          JP(:OUTDONE)


:EXIT7    R=(XY(49,47))              ; See comments for Exit # 1
          JB(1,XY(49,47),:REPEAT)
          MA(XY(49,47),0)
          MD(3,0)
          DN
          TP(1,XY(49,53))
          IV(@SHPRQ7)
:HOLD7    DN
          IF(@OKSHPRQ7,EQ,0,:HOLD7)
          DV(@OKSHPRQ7)
          DV(@SHPRQ7)
          MD(4,0)

:WAIT7    DN
          IF(@GO,EQ,9,:READY7)
          JP(:WAIT7)

:READY7   R=(XY(49,54))
```

```
          JB(2,XY(49,55),XY(48,55),:WAIT7)
          MD(1,0)
          ML(43,3)
          MD(4,6)
          MR(29,3)
          MD(10,6)
          JP(:OUTDONE)


:EXIT8    R=(XY(56,47))              ; See comments for Exit # 1
          JB(1,XY(56,47),:REPEAT)
          MA(XY(56,47),0)
          MD(3,0)
          DN
          TP(1,XY(56,53))
          IV(@SHPRQ8)
:HOLD8    DN
          IF(@OKSHPRQ8,EQ,0,:HOLD8)
          DV(@OKSHPRQ8)
          DV(@SHPRQ8)
          MD(4,0)

:WAIT8    DN
          IF(@GO,EQ,0,:READY8)
          JP(:WAIT8)

:READY8   R=(XY(56,54))
          JB(2,XY(56,55),XY(55,55),:WAIT8)
          MD(1,0)
          ML(50,3)
          MD(4,6)
          MR(29,3)
          MD(10,6)
          JP(:OUTDONE)


:EXIT9    R=(XY(63,47))              ; See comments for Exit # 1
          JB(1,XY(63,47),:REPEAT)
          MA(XY(63,47),0)
          MD(3,0)
          DN
          TP(1,XY(63,53))
          IV(@SHPRQ9)
:HOLD9    DN
          IF(@OKSHPRQ9,EQ,0,:HOLD9)
          DV(@OKSHPRQ9)
          DV(@SHPRQ9)
          MD(4,0)

:WAIT9    DN
          IF(@GO,EQ,3,:READY9)
          JP(:WAIT9)

:READY9   R=(XY(63,54))
```

```
          JB(2,XY(63,55),XY(62,55),:WAIT9)
          MD(1,0)
          ML(57,3)
          MD(4,6)
          MR(29,3)
          MD(10,6)
          JP(:OUTDONE)


:EXIT0    R=(XY(70,47))                  ; See comments for Exit # 1
          JB(1,XY(70,47),:REPEAT)
          MA(XY(70,47),0)
          MD(3,0)
          DN
          TP(1,XY(70,53))
          IV(@SHPRQ0)
:HOLD0    DN
          IF(@OKSHPRQ0,EQ,0,:HOLD0)
          DV(@OKSHPRQ0)
          DV(@SHPRQ0)
          MD(4,0)

:WAIT0    DN
          IF(@GO,EQ,6,:READY0)
          JP(:WAIT0)

:READY0   R=(XY(70,54))
          JB(2,XY(70,55),XY(69,55),:WAIT0)
          MD(1,0)
          ML(64,3)
          MD(4,6)
          MR(29,3)
          MD(10,6)
          JP(:OUTDONE)


:OUTDONE  ER




                                         ; Crane #1
          BR(11,XY(8,50),0)
                                         ; Crane parameters:
          SV(OBJ@5,1)                    ;  horizontal position of crane
          SV(OBJ@6,1)                    ;  vertical position of crane

:CRANE1   DN
:RECVING1 IF(@RECRQ1,GT,0,:REC1)         ; check for pallet to store in bins
          DN                             ; delay to prevent infinite loop
:SHPPING1 IF(@SHPRQ1,GT,0,:SHP1)         ; check for a retrieval from bins
          JP(:CRANE1)                    ; else wait
```

```
                                 ; This section covers placement of
                                 ; a pallet when received

:REC1     SV(@ZONEA,@ZONE1A)     ; set zone quantities for the corr
          SV(@ZONEB,@ZONE1B)     ;  for use by RECZONE
          SV(@ZONEC,@ZONE1C)
          SV(@ZONED,@ZONE1D)
          LK(!RECVZONE)          ; select zone & set parameters
          SV(@ZONE1A,@ZONEA)     ; update zone quantities for corr
          SV(@ZONE1B,@ZONEB)     ;  after zone selected by RECZONE
          SV(@ZONE1C,@ZONEC)
          SV(@ZONE1D,@ZONED)

          IF(OBJ@2,EQ,1,:FULL1)  ; if zone was full, nothing put up

          IV(@OKRECRQ1)          ; inc # to exit from input buffer
          MA(XY(8,47),0)         ; step back 3 to signal "receiving"
          LK(!RECVMOVE)          ; move for receiving
                                 ; print zone quantities
          PV(XY(4,39),@ZONE1D)
          PV(XY(4,41),@ZONE1C)
          PV(XY(4,43),@ZONE1B)
          PV(XY(4,45),@ZONE1A)

          PV(XY(4,32),OBJ@5)     ; print horizontal position
          PV(XY(4,34),OBJ@6)     ; print vertical position

                                 ; At this point, the crane has
                                 ;  placed the pallet at its
                                 ;  position & will wait here until
                                 ;  needed
          MA(XY(8,48),0)         ; forward 1 to signal "waiting"

:FULL1    JP(:SHPPING1)          ; check for shipping request


                                 ; This section covers retrieval of
                                 ;  a pallet for shipping

:SHP1     SV(@ZONEA,@ZONE1A)     ; set zone quantities for the corr
          SV(@ZONEB,@ZONE1B)     ;  for use by SHPZONE
          SV(@ZONEC,@ZONE1C)
          SV(@ZONED,@ZONE1D)
          LK(!SHIPZONE)          ; select zone & set parameters
          SV(@ZONE1A,@ZONEA)     ; update zone quantities for corr
          SV(@ZONE1B,@ZONEB)     ;  after zone selected by SHIPZONE
          SV(@ZONE1C,@ZONEC)
          SV(@ZONE1D,@ZONED)

          IF(OBJ@2,EQ,1,:EMPTY1) ; if zone empty, nothing to ship

          MA(XY(8,49),0)         ; step back 1 to signal "shipping"
          LK(!SHIPMOVE)          ; move for shipping
          IV(@OKSHPRQ1)               ; increment # that can leave corr

                                      ; print zone quantities
          PV(XY(4,39),@ZONE1D)
          PV(XY(4,41),@ZONE1C)
          PV(XY(4,43),@ZONE1B)
          PV(XY(4,45),@ZONE1A)

          PV(XY(4,32),OBJ@5)          ; print horizontal position
          PV(XY(4,34),OBJ@6)          ; print vertical position

                                      ; At this point, the crane has
                                      ;  reached the conveyor and
                                      ;  released the item
          MA(XY(8,50),0)              ; move forward 1 to be at conveyor

:EMPTY1   JP(:RECVING1)               ; check for received item

          ER



                                      ; Crane #2
          BR(12,XY(15,50),0)
                                      ; See comments for Crane # 1
          SV(OBJ@5,1)
          SV(OBJ@6,1)

:CRANE2   DN
:RECVING2 IF(@RECRQ2,GT,0,:REC2)
          DN
:SHPPING2 IF(@SHPRQ2,GT,0,:SHP2)
          JP(:CRANE2)

:REC2     SV(@ZONEA,@ZONE2A)
          SV(@ZONEB,@ZONE2B)
          SV(@ZONEC,@ZONE2C)
          SV(@ZONED,@ZONE2D)
          LK(!RECVZONE)
          SV(@ZONE2A,@ZONEA)
          SV(@ZONE2B,@ZONEB)
          SV(@ZONE2C,@ZONEC)
          SV(@ZONE2D,@ZONED)

          IF(OBJ@2,EQ,1,:FULL2)

          IV(@OKRECRQ2)
          MA(XY(15,47),0)
          LK(!RECVMOVE)

          PV(XY(11,39),@ZONE2D)
          PV(XY(11,41),@ZONE2C)
          PV(XY(11,43),@ZONE2B)
          PV(XY(11,45),@ZONE2A)

          PV(XY(11,32),OBJ@5)
          PV(XY(11,34),OBJ@6)

          MA(XY(15,48),0)

:FULL2    JP(:SHPPING2)


:SHP2     SV(@ZONEA,@ZONE2A)
          SV(@ZONEB,@ZONE2B)
          SV(@ZONEC,@ZONE2C)
          SV(@ZONED,@ZONE2D)
          LK(!SHIPZONE)
          SV(@ZONE2A,@ZONEA)
          SV(@ZONE2B,@ZONEB)
          SV(@ZONE2C,@ZONEC)
          SV(@ZONE2D,@ZONED)

          IF(OBJ@2,EQ,1,:EMPTY2)

          MA(XY(15,49),0)
          LK(!SHIPMOVE)
          IV(@OKSHPRQ2)

          PV(XY(11,39),@ZONE2D)
          PV(XY(11,41),@ZONE2C)
          PV(XY(11,43),@ZONE2B)
          PV(XY(11,45),@ZONE2A)

          PV(XY(11,32),OBJ@5)
          PV(XY(11,34),OBJ@6)

          MA(XY(15,50),0)

:EMPTY2   JP(:RECVING2)

          ER



                                          ; Crane #3
          BR(13,XY(22,50),0)
                                          ; See comments for Crane # 1
          SV(OBJ@5,1)
          SV(OBJ@6,1)

:CRANE3   DN
:RECVING3 IF(@RECRQ3,GT,0,:REC3)
          DN
:SHPPING3 IF(@SHPRQ3,GT,0,:SHP3)
```

```
          JP(:CRANE3)

:REC3     SV(@ZONEA,@ZONE3A)
          SV(@ZONEB,@ZONE3B)
          SV(@ZONEC,@ZONE3C)
          SV(@ZONED,@ZONE3D)
          LK(!RECVZONE)
          SV(@ZONE3A,@ZONEA)
          SV(@ZONE3B,@ZONEB)
          SV(@ZONE3C,@ZONEC)
          SV(@ZONE3D,@ZONED)

          IF(OBJ@2,EQ,1,:FULL3)

          IV(@OKRECRQ3)
          MA(XY(22,47),0)
          LK(!RECVMOVE)

          PV(XY(18,39),@ZONE3D)
          PV(XY(18,41),@ZONE3C)
          PV(XY(18,43),@ZONE3B)
          PV(XY(18,45),@ZONE3A)

          PV(XY(18,32),OBJ@5)
          PV(XY(18,34),OBJ@6)

          MA(XY(22,48),0)

:FULL3    JP(:SHPPING3)


:SHP3     SV(@ZONEA,@ZONE3A)
          SV(@ZONEB,@ZONE3B)
          SV(@ZONEC,@ZONE3C)
          SV(@ZONED,@ZONE3D)
          LK(!SHIPZONE)
          SV(@ZONE3A,@ZONEA)
          SV(@ZONE3B,@ZONEB)
          SV(@ZONE3C,@ZONEC)
          SV(@ZONE3D,@ZONED)

          IF(OBJ@2,EQ,1,:EMPTY3)

          MA(XY(22,49),0)
          LK(!SHIPMOVE)
          IV(@OKSHPRQ3)

          PV(XY(18,39),@ZONE3D)
          PV(XY(18,41),@ZONE3C)
          PV(XY(18,43),@ZONE3B)
          PV(XY(18,45),@ZONE3A)

          PV(XY(18,32),OBJ@5)
          PV(XY(18,34),OBJ@6)

          MA(XY(22,50),0)

:EMPTY3   JP(:RECVING3)

          ER



                                        ; Crane #4
          BR(14,XY(29,50),0)
                                        ; See comments for Crane # 1
          SV(OBJ@5,1)
          SV(OBJ@6,1)

:CRANE4   DN
:RECVING4 IF(@RECRQ4,GT,0,:REC4)
          DN
:SHPPING4 IF(@SHPRQ4,GT,0,:SHP4)
          JP(:CRANE4)

:REC4     SV(@ZONEA,@ZONE4A)
          SV(@ZONEB,@ZONE4B)
          SV(@ZONEC,@ZONE4C)
          SV(@ZONED,@ZONE4D)
          LK(!RECVZONE)
          SV(@ZONE4A,@ZONEA)
          SV(@ZONE4B,@ZONEB)
          SV(@ZONE4C,@ZONEC)
          SV(@ZONE4D,@ZONED)

          IF(OBJ@2,EQ,1,:FULL4)

          IV(@OKRECRQ4)
          MA(XY(29,47),0)
          LK(!RECVMOVE)

          PV(XY(25,39),@ZONE4D)
          PV(XY(25,41),@ZONE4C)
          PV(XY(25,43),@ZONE4B)
          PV(XY(25,45),@ZONE4A)

          PV(XY(25,32),OBJ@5)
          PV(XY(25,34),OBJ@6)

          MA(XY(29,48),0)

:FULL4    JP(:SHPPING4)


:SHP4     SV(@ZONEA,@ZONE4A)
          SV(@ZONEB,@ZONE4B)
```

```
            SV(@ZONEC,@ZONE4C)
            SV(@ZONED,@ZONE4D)
            LK(!SHIPZONE)
            SV(@ZONE4A,@ZONEA)
            SV(@ZONE4B,@ZONEB)
            SV(@ZONE4C,@ZONEC)
            SV(@ZONE4D,@ZONED)

            IF(OBJ@2,EQ,1,:EMPTY4)

            MA(XY(29,49),0)
            LK(!SHIPMOVE)
            IV(@OKSHPRQ4)

            PV(XY(25,39),@ZONE4D)
            PV(XY(25,41),@ZONE4C)
            PV(XY(25,43),@ZONE4B)
            PV(XY(25,45),@ZONE4A)

            PV(XY(25,32),OBJ@5)
            PV(XY(25,34),OBJ@6)

            MA(XY(29,50),0)

:EMPTY4     JP(:RECVING4)

            ER



                                            ; Crane #5
            BR(15,XY(36,50),0)
                                            ; See comments for Crane # 1
            SV(OBJ@5,1)
            SV(OBJ@6,1)

:CRANE5     DN
:RECVING5   IF(@RECRQ5,GT,0,:REC5)
            DN
:SHPPING5   IF(@SHPRQ5,GT,0,:SHP5)
            JP(:CRANE5)

:REC5       SV(@ZONEA,@ZONE5A)
            SV(@ZONEB,@ZONE5B)
            SV(@ZONEC,@ZONE5C)
            SV(@ZONED,@ZONE5D)
            LK(!RECVZONE)
            SV(@ZONE5A,@ZONEA)
            SV(@ZONE5B,@ZONEB)
            SV(@ZONE5C,@ZONEC)
            SV(@ZONE5D,@ZONED)

            IF(OBJ@2,EQ,1,:FULL5)
            IV(@OKRECRQ5)
            MA(XY(36,47),0)
            LK(!RECVMOVE)

            PV(XY(32,39),@ZONE5D)
            PV(XY(32,41),@ZONE5C)
            PV(XY(32,43),@ZONE5B)
            PV(XY(32,45),@ZONE5A)

            PV(XY(32,32),OBJ@5)
            PV(XY(32,34),OBJ@6)

            MA(XY(36,48),0)

:FULL5      JP(:SHPPING5)


:SHP5       SV(@ZONEA,@ZONE5A)
            SV(@ZONEB,@ZONE5B)
            SV(@ZONEC,@ZONE5C)
            SV(@ZONED,@ZONE5D)
            LK(!SHIPZONE)
            SV(@ZONE5A,@ZONEA)
            SV(@ZONE5B,@ZONEB)
            SV(@ZONE5C,@ZONEC)
            SV(@ZONE5D,@ZONED)

            IF(OBJ@2,EQ,1,:EMPTY5)

            MA(XY(36,49),0)
            LK(!SHIPMOVE)
            IV(@OKSHPRQ5)

            PV(XY(32,39),@ZONE5D)
            PV(XY(32,41),@ZONE5C)
            PV(XY(32,43),@ZONE5B)
            PV(XY(32,45),@ZONE5A)

            PV(XY(32,32),OBJ@5)
            PV(XY(32,34),OBJ@6)

            MA(XY(36,50),0)

:EMPTY5     JP(:RECVING5)

            ER


                                      ; Crane #6
            BR(16,XY(43,50),0)
                                      ; See comments for Crane # 1
            SV(OBJ@5,1)
```

```
          SV(OBJ@6,1)

:CRANE6   DN
:RECVING6 IF(@RECRQ6,GT,0,:REC6)
          DN
:SHPPING6 IF(@SHPRQ6,GT,0,:SHP6)
          JP(:CRANE6)

:REC6     SV(@ZONEA,@ZONE6A)
          SV(@ZONEB,@ZONE6B)
          SV(@ZONEC,@ZONE6C)
          SV(@ZONED,@ZONE6D)
          LK(!RECVZONE)
          SV(@ZONE6A,@ZONEA)
          SV(@ZONE6B,@ZONEB)
          SV(@ZONE6C,@ZONEC)
          SV(@ZONE6D,@ZONED)

          IF(OBJ@2,EQ,1,:FULL6)

          IV(@OKRECRQ6)
          MA(XY(43,47),0)
          LK(!RECVMOVE)

          PV(XY(39,39),@ZONE6D)
          PV(XY(39,41),@ZONE6C)
          PV(XY(39,43),@ZONE6B)
          PV(XY(39,45),@ZONE6A)

          PV(XY(39,32),OBJ@5)
          PV(XY(39,34),OBJ@6)

          MA(XY(43,48),0)

:FULL6    JP(:SHPPING6)


:SHP6     SV(@ZONEA,@ZONE6A)
          SV(@ZONEB,@ZONE6B)
          SV(@ZONEC,@ZONE6C)
          SV(@ZONED,@ZONE6D)
          LK(!SHIPZONE)
          SV(@ZONE6A,@ZONEA)
          SV(@ZONE6B,@ZONEB)
          SV(@ZONE6C,@ZONEC)
          SV(@ZONE6D,@ZONED)

          IF(OBJ@2,EQ,1,:EMPTY6)

          MA(XY(43,49),0)
          LK(!SHIPMOVE)
          IV(@OKSHPRQ6)
```

```
            PV(XY(39,39),@ZONE6D)
            PV(XY(39,41),@ZONE6C)
            PV(XY(39,43),@ZONE6B)
            PV(XY(39,45),@ZONE6A)

            PV(XY(39,32),OBJ@5)
            PV(XY(39,34),OBJ@6)

            MA(XY(43,50),0)

:EMPTY6     JP(:RECVING6)

            ER



                                      ; Crane #7
            BR(17,XY(50,50),0)
                                      ; See comments for Crane # 1
            SV(OBJ@5,1)
            SV(OBJ@6,1)

:CRANE7     DN
:RECVING7   IF(@RECRQ7,GT,0,:REC7)
            DN
:SHPPING7   IF(@SHPRQ7,GT,0,:SHP7)
            JP(:CRANE7)

:REC7       SV(@ZONEA,@ZONE7A)
            SV(@ZONEB,@ZONE7B)
            SV(@ZONEC,@ZONE7C)
            SV(@ZONED,@ZONE7D)
            LK(!RECVZONE)
            SV(@ZONE7A,@ZONEA)
            SV(@ZONE7B,@ZONEB)
            SV(@ZONE7C,@ZONEC)
            SV(@ZONE7D,@ZONED)

            IF(OBJ@2,EQ,1,:FULL7)

            IV(@OKRECRQ7)
            MA(XY(50,47),0)
            LK(!RECVMOVE)

            PV(XY(46,39),@ZONE7D)
            PV(XY(46,41),@ZONE7C)
            PV(XY(46,43),@ZONE7B)
            PV(XY(46,45),@ZONE7A)

            PV(XY(46,32),OBJ@5)
            PV(XY(46,34),OBJ@6)
```

```
          MA(XY(50,48),0)

:FULL7    JP(:SHPPING7)


:SHP7     SV(@ZONEA,@ZONE7A)
          SV(@ZONEB,@ZONE7B)
          SV(@ZONEC,@ZONE7C)
          SV(@ZONED,@ZONE7D)
          LK(!SHIPZONE)
          SV(@ZONE7A,@ZONEA)
          SV(@ZONE7B,@ZONEB)
          SV(@ZONE7C,@ZONEC)
          SV(@ZONE7D,@ZONED)

          IF(OBJ@2,EQ,1,:EMPTY7)

          MA(XY(50,49),0)
          LK(!SHIPMOVE)
          IV(@OKSHPRQ7)

          PV(XY(46,39),@ZONE7D)
          PV(XY(46,41),@ZONE7C)
          PV(XY(46,43),@ZONE7B)
          PV(XY(46,45),@ZONE7A)

          PV(XY(46,32),OBJ@5)
          PV(XY(46,34),OBJ@6)

          MA(XY(50,50),0)

:EMPTY7   JP(:RECVING7)

          ER



                                   ; Crane #8
          BR(18,XY(57,50),0)
                                   ; See comments for Crane # 1
          SV(OBJ@5,1)
          SV(OBJ@6,1)

:CRANE8   DN
:RECVING8 IF(@RECRQ8,GT,0,:REC8)
          DN
:SHPPING8 IF(@SHPRQ8,GT,0,:SHP8)
          JP(:CRANE8)

:REC8     SV(@ZONEA,@ZONE8A)
          SV(@ZONEB,@ZONE8B)
          SV(@ZONEC,@ZONE8C)
          SV(@ZONED,@ZONE8D)
          LK(!RECVZONE)
          SV(@ZONE8A,@ZONEA)
          SV(@ZONE8B,@ZONEB)
          SV(@ZONE8C,@ZONEC)
          SV(@ZONE8D,@ZONED)

          IF(OBJ@2,EQ,1,:FULL8)

          IV(@OKRECRQ8)
          MA(XY(57,47),0)
          LK(!RECVMOVE)

          PV(XY(53,39),@ZONE8D)
          PV(XY(53,41),@ZONE8C)
          PV(XY(53,43),@ZONE8B)
          PV(XY(53,45),@ZONE8A)

          PV(XY(53,32),OBJ@5)
          PV(XY(53,34),OBJ@6)

          MA(XY(57,48),0)

:FULL8    JP(:SHPPING8)


:SHP8     SV(@ZONEA,@ZONE8A)
          SV(@ZONEB,@ZONE8B)
          SV(@ZONEC,@ZONE8C)
          SV(@ZONED,@ZONE8D)
          LK(!SHIPZONE)
          SV(@ZONE8A,@ZONEA)
          SV(@ZONE8B,@ZONEB)
          SV(@ZONE8C,@ZONEC)
          SV(@ZONE8D,@ZONED)

          IF(OBJ@2,EQ,1,:EMPTY8)

          MA(XY(57,49),0)
          LK(!SHIPMOVE)
          IV(@OKSHPRQ8)

          PV(XY(53,39),@ZONE8D)
          PV(XY(53,41),@ZONE8C)
          PV(XY(53,43),@ZONE8B)
          PV(XY(53,45),@ZONE8A)

          PV(XY(53,32),OBJ@5)
          PV(XY(53,34),OBJ@6)

          MA(XY(57,50),0)

:EMPTY8   JP(:RECVING8)
```

```
          ER



                                                  ; Crane #9
          BR(19,XY(64,50),0)
                                                  ; See comments for Crane # 1
          SV(OBJ@5,1)
          SV(OBJ@6,1)

:CRANE9   DN
:RECVING9 IF(@RECRQ9,GT,0,:REC9)
          DN
:SHPPING9 IF(@SHPRQ9,GT,0,:SHP9)
          JP(:CRANE9)

:REC9     SV(@ZONEA,@ZONE9A)
          SV(@ZONEB,@ZONE9B)
          SV(@ZONEC,@ZONE9C)
          SV(@ZONED,@ZONE9D)
          LK(!RECVZONE)
          SV(@ZONE9A,@ZONEA)
          SV(@ZONE9B,@ZONEB)
          SV(@ZONE9C,@ZONEC)
          SV(@ZONE9D,@ZONED)

          IF(OBJ@2,EQ,1,:FULL9)

          IV(@OKRECRQ9)
          MA(XY(64,47),0)
          LK(!RECVMOVE)

          PV(XY(60,39),@ZONE9D)
          PV(XY(60,41),@ZONE9C)
          PV(XY(60,43),@ZONE9B)
          PV(XY(60,45),@ZONE9A)

          PV(XY(60,32),OBJ@5)
          PV(XY(60,34),OBJ@6)

          MA(XY(64,48),0)

:FULL9    JP(:SHPPING9)


:SHP9     SV(@ZONEA,@ZONE9A)
          SV(@ZONEB,@ZONE9B)
          SV(@ZONEC,@ZONE9C)
          SV(@ZONED,@ZONE9D)
          LK(!SHIPZONE)
          SV(@ZONE9A,@ZONEA)
          SV(@ZONE9B,@ZONEB)
          SV(@ZONE9C,@ZONEC)
          SV(@ZONE9D,@ZONED)

          IF(OBJ@2,EQ,1,:EMPTY9)

          MA(XY(64,49),0)
          LK(!SHIPMOVE)
          IV(@OKSHPRQ9)

          PV(XY(60,39),@ZONE9D)
          PV(XY(60,41),@ZONE9C)
          PV(XY(60,43),@ZONE9B)
          PV(XY(60,45),@ZONE9A)

          PV(XY(60,32),OBJ@5)
          PV(XY(60,34),OBJ@6)

          MA(XY(64,50),0)

:EMPTY9   JP(:RECVING9)

          ER



                                      ; Crane #0
          BR(20,XY(71,50),0)
                                      ; See comments for Crane # 1
          SV(OBJ@5,1)
          SV(OBJ@6,1)

:CRANE0   DN
:RECVING0 IF(@RECRQ0,GT,0,:REC0)
          DN
:SHPPING0 IF(@SHPRQ0,GT,0,:SHP0)
          JP(:CRANE0)

:REC0     SV(@ZONEA,@ZONE0A)
          SV(@ZONEB,@ZONE0B)
          SV(@ZONEC,@ZONE0C)
          SV(@ZONED,@ZONE0D)
          LK(!RECVZONE)
          SV(@ZONE0A,@ZONEA)
          SV(@ZONE0B,@ZONEB)
          SV(@ZONE0C,@ZONEC)
          SV(@ZONE0D,@ZONED)

          IF(OBJ@2,EQ,1,:FULL0)

          IV(@OKRECRQ0)
          MA(XY(71,47),0)
          LK(!RECVMOVE)

          PV(XY(67,39),@ZONE0D)
          PV(XY(67,41),@ZONE0C)
          PV(XY(67,43),@ZONE0B)
          PV(XY(67,45),@ZONE0A)

          PV(XY(67,32),OBJ@5)
          PV(XY(67,34),OBJ@6)

          MA(XY(71,48),0)

:FULL0    JP(:SHPPING0)


:SHP0     SV(@ZONEA,@ZONE0A)
          SV(@ZONEB,@ZONE0B)
          SV(@ZONEC,@ZONE0C)
          SV(@ZONED,@ZONE0D)
          LK(!SHIPZONE)
          SV(@ZONE0A,@ZONEA)
          SV(@ZONE0B,@ZONEB)
          SV(@ZONE0C,@ZONEC)
          SV(@ZONE0D,@ZONED)

          IF(OBJ@2,EQ,1,:EMPTY0)

          MA(XY(71,49),0)
          LK(!SHIPMOVE)
          IV(@OKSHPRQ0)

          PV(XY(67,39),@ZONE0D)
          PV(XY(67,41),@ZONE0C)
          PV(XY(67,43),@ZONE0B)
          PV(XY(67,45),@ZONE0A)

          PV(XY(67,32),OBJ@5)
          PV(XY(67,34),OBJ@6)

          MA(XY(71,50),0)

:EMPTY0   JP(:RECVING0)

          ER
```

APPENDIX B

THE SUPERMARKET PROGRAM

```
M=(100)                                ; max number of objects that are
                                       ;  allowed to be active at one time
W=(100)                                ; initial num of objects that may
                                       ;  be active at any given time
S=(250)                                ; count of symbols to have storage
                                       ;  reserved for

X=(81)                                 ; x dimension of the overlay
Y=(47)                                 ; y dimension of the overlay

V=(XY(1,0))                            ; load-time viewing window location



D=
             Grocery Store Simulation
   A supermarket is scheduled to have 20 shopping carts, 1
regular checkout counter, and 1 express checkout lane.
Customers are expected to arrive at the supermarket every 4
minutes on the average, with the time between successive
arrivals varying between 0 and 8 minutes according to a
discrete uniform distribution.  Each arriving customer tries
immediately to obtain a shopping cart.  If no shopping cart
is available, the customer will wait until one becomes
available before beginning shopping.  It is anticipated that
the 30% of entering customers who intend to perform express
shopping tasks will remain in the store for only 6 +/- 5
minutes and will then join the express checkout line.  The
other 70% of entering customers are expected to perform
regular shopping tasks, remaining in the shopping area of
the store for 30 +/- 15 minutes and eventually joining the
regular checkout line.  Checkout is expected to require
5+/-2 minutes at the regular checkout counter and 2 +/- 1
minutes at the express checkout counter.  After checkout,
the customer returns the shopping cart and leaves the store.
$

O=(=)                                  ; kept in SM.OLY


                                       ; SYMBOLS

@ARRIVE=(240)                          ; arrival rate of customers
@CHANCE=(0)                            ; random variable for exp vs reg
@SHOPTIME=(0)                          ; time to shop
@EXPCHK=(0)                            ; time for express checking
@REGCHK=(0)                            ; time for regular checking
```

```
@TERMCNT=(0)                       ; count of completed objects
@GPSSTERM=(500)                    ; termination count in GPSS model

@NUMBCTS=(0)                       ; number of statistic counts made
@ROUND=(0)                         ; variable for rounding
@AVERAGE=(0)                       ; variable for decimal average

@TTLCARTS=(20)                     ; number of carts in the store
@NUMCARTS=(0)                      ; number of carts available

@CRTINUSE=(0)                      ; number of carts being used
@TTLCRTSU=(0)                      ; total entries in storage
@MAXCRTSU=(0)                      ; max in storage at any one time
@CKCRTUSE=(0)                      ; sum of contents at stat times
%TTLCRTST=(0000:00:00)             ; sum of object times in cart stg

@REGQUEUE=(0)                      ; current contents of reg line que
@MAXREGQ=(0)                       ; maximum in reg line at any time
@TTLREGQ=(0)                       ; total entries in reg ckout line
@CKREGQ=(0)                        ; sum of contents at stat times
%TTLREGQT=(0000:00:00)             ; sum of object times in reg queue

@EXPQUEUE=(0)                      ; current contents of exp line que
@MAXEXPQ=(0)                       ; maximum in exp line at any time
@TTLEXPQ=(0)                       ; total entries in exp ckout line
@CKEXPQ=(0)                        ; sum of contents at stat times
%TTLEXPQT=(0000:00:00)             ; sum of object times in exp queue

@CRTQUEUE=(0)                      ; current contents of cart queue
@MAXCRTQ=(0)                       ; max waiting for cart at any time
@TTLCRTQ=(0)                       ; total entries in line for a cart
@CKCRTQ=(0)                        ; sum of contents at stat times
%TTLCRTQT=(0000:00:00)             ; sum of object times in cart queue

%OVERFLOW=(0021:48:00)             ; overflow time of stat variables
%LOOPTIME=(0000:00:00)             ; time of next statistics update
%DIFFRNCE=(0000:00:00)             ; variable for diff of object times

#SEED=(9997)                       ; seed of random number sequence
#PERCENT=(30)                      ; percent express shoppers
                                   ;  (complement is percent regular)



*DUMMY1=(XY(2,2))                  ; dummy position for init job
*DUMMY2=(XY(1,39))                 ; dummy position for statistics job

*ENTRY=(XY(40,0))                  ; entry position for shoppers
*EXIT=(XY(68,4))                   ; exit position for shoppers
*CARTS=(XY(62,2))                  ; location to wait for a cart
*READY=(XY(62,5))                  ; location where ready to shop
*REGLINE=(XY(44,22))               ; common position all reg shoppers
```

```
                                        ;  meet at headed for reg checkout
*EXPLINE=(XY(62,22))                    ; common position all exp shoppers
                                        ;  meet at headed for exp checkout
*REGLANE=(XY(75,10))                    ; spot occupied for reg checkout
*EXPLANE=(XY(68,10))                    ; spot occupied for exp checkout

*LANE1=(XY(4,6))                        ; reserve pos at head of lane 1
*LANE2=(XY(12,6))                       ; reserve pos at head of lane 2
*LANE3=(XY(14,6))                       ; reserve pos at head of lane 3
*LANE4=(XY(22,6))                       ; reserve pos at head of lane 4
*LANE5=(XY(24,6))                       ; reserve pos at head of lane 5
*LANE6=(XY(32,6))                       ; reserve pos at head of lane 6
*LANE7=(XY(34,6))                       ; reserve pos at head of lane 7
*LANE8=(XY(42,6))                       ; reserve pos at head of lane 8
*LANE9=(XY(44,6))                       ; reserve pos at head of lane 9
*LANEA=(XY(52,6))                       ; reserve pos at head of lane A
*LANEB=(XY(54,6))                       ; reserve pos at head of lane B
*LANEC=(XY(62,6))                       ; reserve pos at head of lane C

*CRTSFREE=(XY(65,2))                    ; number of carts available area
*MESSAGE=(XY(2,1))                      ; overflow & term message spot
*COUNTS=(XY(28,25))                     ; position for count of stat calcs
*TIME=(XY(60,25))                       ; spot for time of last stat update

                                        ; fields for cart stg statistics
*CRTSTG1=(XY(16,30))                    ;  storage capacity
*CRTSTG2A=(XY(25,30))                   ;  average contents
*CRTSTG2B=(XY(30,30))
*CRTSTG2C=(XY(27,30))
*CRTSTG3=(XY(37,30))                    ;  total entries
*CRTSTG4=(XY(46,30))                    ;  total time spent in storage
*CRTSTG5=(XY(60,30))                    ;  current contents
*CRTSTG6=(XY(71,30))                    ;  maximum contents

                                        ; fields for reg line queue stats
*REGQUE1=(XY(14,36))                    ;  maximum contents
*REGQUE2A=(XY(24,36))                   ;  average contents
*REGQUE2B=(XY(29,36))
*REGQUE2C=(XY(26,36))
*REGQUE3=(XY(35,36))                    ;  total entries
*REGQUE4=(XY(44,36))                    ;  total time spent in queue
*REGQUE5=(XY(58,36))                    ;  current contents

                                        ; fields for exp line queue stats
*EXPQUE1=(XY(14,37))                    ;  maximum contents
*EXPQUE2A=(XY(24,37))                   ;  average contents
*EXPQUE2B=(XY(29,37))
*EXPQUE2C=(XY(26,37))
*EXPQUE3=(XY(35,37))                    ;  total entries
*EXPQUE4=(XY(44,37))                    ;  total time spent in queue
*EXPQUE5=(XY(58,37))                    ;  current contents

                                        ; fields for cart queue statistics
*CRTQUE1=(XY(14,38))                    ;  maximum contents
```

```
*CRTQUE2A=(XY(24,38))                ;  average contents
*CRTQUE2B=(XY(29,38))
*CRTQUE2C=(XY(26,38))
*CRTQUE3=(XY(35,38))                 ;  total entries
*CRTQUE4=(XY(44,38))                 ;  total time spent in queue
*CRTQUE5=(XY(58,38))                 ;  current contents



                                     ; JOBS

J=(1,X,1,0,0,0,1)                    ; initialization
J=(2,#,2,0,0,1,1)                    ; statistics
J=(3,C,3,0,0,2,1000)                 ; shoppers



                                     ; UTILIZATION LOCATIONS

U=(1,Reg Chk,*REGLANE)               ; regular checkout
U=(2,Exp Chk,*EXPLANE)               ; express checkout

U=(3,Lane 1,*LANE1)                  ; reserve position of lane 1
U=(4,Lane 2,*LANE2)                  ; reserve position of lane 2
U=(5,Lane 3,*LANE3)                  ; reserve position of lane 3
U=(6,Lane 4,*LANE4)                  ; reserve position of lane 4
U=(7,Lane 5,*LANE5)                  ; reserve position of lane 5
U=(8,Lane 6,*LANE6)                  ; reserve position of lane 6
U=(9,Lane 7,*LANE7)                  ; reserve position of lane 7
U=(10,Lane 8,*LANE8)                 ; reserve position of lane 8
U=(11,Lane 9,*LANE9)                 ; reserve position of lane 9
U=(12,Lane A,*LANEA)                 ; reserve position of lane A
U=(13,Lane B,*LANEB)                 ; reserve position of lane B
U=(14,Lane C,*LANEC)                 ; reserve position of lane C



                                     ; INITIALIZATION ROUTE

          BR(1,*DUMMY1,0)
          RS(#SEED)                  ; initialize random number sequence

                                     ; init variables defined above
          SV(@ARRIVE,240)
          SV(@TERMCNT,0)

          SV(@NUMCARTS,@TTLCARTS)
          SV(@NUMBCTS,0)

          SV(@CRTINUSE,0)
```

```
         SV(@TTLCRTSU,0)
         SV(@MAXCRTSU,0)
         SV(@CKCRTUSE,0)
         SV(%TTLCRTST,0)

         SV(@REGQUEUE,0)
         SV(@MAXREGQ,0)
         SV(@TTLREGQ,0)
         SV(@CKREGQ,0)
         SV(%TTLREGQT,0)

         SV(@EXPQUEUE,0)
         SV(@MAXEXPQ,0)
         SV(@TTLEXPQ,0)
         SV(@CKEXPQ,0)
         SV(%TTLEXPQT,0)

         SV(@CRTQUEUE,0)
         SV(@MAXCRTQ,0)
         SV(@TTLCRTQ,0)
         SV(@CKCRTQ,0)
         SV(%TTLCRTQT,0)

                                   ; check for termination count
         IF(@TERMCNT,LT,@GPSSTERM,THEN,WAIT)
                                   ; if count reached, output message
         PM(*MESSAGE,500 OBJECTS COMPLETED)
         WK                        ; pause for ouput of statistics
                                   ; erase message to go on
         PM(*MESSAGE,xxxxxxxxxxxxxxxxxxxx)

         ER



                                   ; STATISTICS ROUTE

         BR(2,*DUMMY2,240)

         SV(%LOOPTIME,240)         ; initialize loop time

:TOP     IV(@NUMBCTS)              ; update number of statistic counts
         PV(*COUNTS,@NUMBCTS)      ; output with statistics
         SV(@ROUND,@NUMBCTS)       ; copy to get value for rounding
         AO(@ROUND,/,2)            ; @ROUND to be added in before div
                                   ;  by @NUMBCTS to provide rounding

                                   ; Statistics for cart storage
         PV(*CRTSTG1,@TTLCARTS)    ; capacity of storage
         AO(@CKCRTUSE,+,@CRTINUSE) ; sum # currently in use for check
         SV(@AVERAGE,@CKCRTUSE)    ; copy check-cart-use for average
         AO(@AVERAGE,*,10)         ; multiply by 10; then int division
```

```
                            ;  will leave tenths in one's place
AO(@AVERAGE,+,@ROUND)       ; add half divisor for rounding
AO(@AVERAGE,/,@NUMBCTS)     ; div by number of counts taken
                            ; note:  *CRTSTG2 A, B, & C overlay
                            ;  one another to get decimal place
PV(*CRTSTG2C,@AVERAGE)      ; print average-multiplied-by-10
PM(*CRTSTG2B,.)             ; overlay a decimal point on it
AO(@AVERAGE,/,10)           ; divide by 10 again; this throws
                            ;  away the one's (tenths) position
PV(*CRTSTG2A,@AVERAGE)      ; overlay integer part of average
                            ;  on the field
PV(*CRTSTG3,@TTLCRTSU)      ; total entries in storage
PV(*CRTSTG4,%TTLCRTST)      ; total of times spent in cart stg
PV(*CRTSTG5,@CRTINUSE)      ; current number in use
PV(*CRTSTG6,@MAXCRTSU)      ; max in storage at any one time

                            ; Statistics for regular line queue
PV(*REGQUE1,@MAXREGQ)       ; maximum in queue at any one time
AO(@CKREGQ,+,@REGQUEUE)     ; the avg queue size is calculated
SV(@AVERAGE,@CKREGQ)        ;  to tenths by the technique used
AO(@AVERAGE,*,10)           ;  for the cart storage average
AO(@AVERAGE,+,@ROUND)
AO(@AVERAGE,/,@NUMBCTS)
PV(*REGQUE2C,@AVERAGE)
PM(*REGQUE2B,.)
AO(@AVERAGE,/,10)
PV(*REGQUE2A,@AVERAGE)
PV(*REGQUE3,@TTLREGQ)       ; total entries into regular queue
PV(*REGQUE4,%TTLREGQT)      ; total of times spent in reg queue
PV(*REGQUE5,@REGQUEUE)      ; current contents of reg queue

                            ; Statistics for express line queue
PV(*EXPQUE1,@MAXEXPQ)       ; see comments for reg line queue
AO(@CKEXPQ,+,@EXPQUEUE)
SV(@AVERAGE,@CKEXPQ)
AO(@AVERAGE,*,10)
AO(@AVERAGE,+,@ROUND)
AO(@AVERAGE,/,@NUMBCTS)
PV(*EXPQUE2C,@AVERAGE)
PM(*EXPQUE2B,.)
AO(@AVERAGE,/,10)
PV(*EXPQUE2A,@AVERAGE)
PV(*EXPQUE3,@TTLEXPQ)
PV(*EXPQUE4,%TTLEXPQT)
PV(*EXPQUE5,@EXPQUEUE)

                            ; Statistics for cart queue
PV(*CRTQUE1,@MAXCRTQ)       ; see comments for reg line queue
AO(@CKCRTQ,+,@CRTQUEUE)
SV(@AVERAGE,@CKCRTQ)
AO(@AVERAGE,*,10)
AO(@AVERAGE,+,@ROUND)
AO(@AVERAGE,/,@NUMBCTS)
```

```
        PV(*CRTQUE2C,@AVERAGE)
        PM(*CRTQUE2B,.)
        AO(@AVERAGE,/,10)
        PV(*CRTQUE2A,@AVERAGE)
        PV(*CRTQUE3,@TTLCRTQ)
        PV(*CRTQUE4,%TTLCRTQT)
        PV(*CRTQUE5,@CRTQUEUE)

        IF(CLOCK,LT,%OVERFLOW,:SKIP) ; check clock against time that
                                     ;  it would be possible for stat
                                     ;  variables to overflow

        PM(*MESSAGE,STATISTICS CAN OVERFLOW)
        WK                           ; halt simulation for user action
                                     ; erase message to go on
        PM(*MESSAGE,xxxxxxxxxxxxxxxxxxxxxx)

:SKIP   PV(*TIME,%LOOPTIME)          ; output time of current update
        AO(%LOOPTIME,+,240)          ; calculate next update time
        ST(240)                      ; wait until time of next update
        JP(:TOP)                     ; jump to computation of stats

        ER



                                     ; SHOPPERS' ROUTE

        BR(3,*ENTRY,@ARRIVE)
        RV(@ARRIVE,0,480)            ; next arrival in 4+/-4 min
        MD(1,5)                      ; come inside store for shopping
        IV(@TTLCRTQ)                 ; total number entering cart queue
        IV(@CRTQUEUE)                ; current number in cart queue
        SV(OBJ%1,CLOCK)              ; start time in cart queue
        MR(22,5)                     ; move to the cart storage area
                                     ; branch if current is not a max
        IF(@CRTQUEUE,LT,@MAXCRTQ,THEN,:PASS)
        SV(@MAXCRTQ,@CRTQUEUE)       ; save max waiting at any one time

                                     ; proceed if a cart is available
:PASS   IF(@NUMCARTS,GT,0,THEN,NEXT,ELSE,WAIT)

        DV(@CRTQUEUE)                ; got cart, so no longer waiting
        SV(%DIFFRNCE,CLOCK)          ; end time in cart queue
        AO(%DIFFRNCE,-,OBJ%1)        ; time spent in queue
        AO(%DIFFRNCE,-,110)          ; required time in queue
        AO(%TTLCRTQT,+,%DIFFRNCE)    ; update tot of times spent in que

        IV(@TTLCRTSU)                ; total number using a cart
        IV(@CRTINUSE)                ; current number of carts in use
        SV(OBJ%2,CLOCK)              ; start time in cart storage
                                     ; branch if current is not a max
```

```
          IF(@CRTINUSE,LE,@MAXCRTSU,THEN,:OVER)
          SV(@MAXCRTSU,@CRTINUSE)     ; save max in use at any one time

:OVER     MA(*CARTS,0)                ; move to get cart
          DV(@NUMCARTS)               ; one less cart available
          PV(*CRTSFREE,@NUMCARTS)     ; print num available in cart area
          MD(3,1)                     ; move to pt from which to branch
          RV(@CHANCE,1,100)           ; chance for reg vs exp shopper
                                      ; branch depending on #PERCENT
          IF(@CHANCE,LT,#PERCENT,THEN,:EXPLOOP)


:REGLOOP  JC(1,*LANE1,:SHPLANE1)      ; to prevent shoppers who will be
          JC(1,*LANE2,:SHPLANE2)      ;  staying in the simulation for
          JC(1,*LANE3,:SHPLANE3)      ;  varying amounts of time from
          JC(1,*LANE4,:SHPLANE4)      ;  interfering with one another,
          JC(1,*LANE5,:SHPLANE5)      ;  each successive shopper object
          JC(1,*LANE6,:SHPLANE6)      ;  is sent to a different lane
          JC(1,*LANE7,:SHPLANE7)      ;  (assuming 9 lanes will be
          JC(1,*LANE8,:SHPLANE8)      ;  sufficient for regular shoppers)
          JC(1,*LANE9,:SHPLANE9)
          DN                          ; pause if all lanes occupied
          JP(:REGLOOP)                ; try again

:SHPLANE1 R=(*READY)                  ; reference for relative moves
          PO(*LANE1)                  ; post reserve position of lane # 1
          ML(10,1)                    ; exp rate thru common rt (10 sec)
          ML(48,5)                    ; move to lane 1 (48 * 5 = 240 sec)
          CL(*LANE1)                  ; clear reserve position for object
          MD(1,5)                     ; take reserve pos. (1 * 5 = 5 sec)
          RV(@SHOPTIME,900,2700)      ; determine shop time (30+/-15 min)
          AO(@SHOPTIME,-,610)         ; subtract time spent by regular
                                      ;  shoppers peculiar to lane # 1
                                      ;  sum (10,240,5,150,5,200) = 610
          AO(@SHOPTIME,-,240)         ; time common to all reg shoppers
          ST(@SHOPTIME)               ; delay at reserve pos for the rest
          MD(15,10)                   ; down lane (15 * 10 = 150 sec)
          MD(1,5)                     ; move to back wall (5 sec)
          MR(40,5)                    ; move to common position for reg
                                      ;  shoppers (40 * 5 = 200 sec)
          JP(:REGCHK)                 ; transfer to common route for regs

:SHPLANE2 R=(*READY)                  ; see comments for :SHPLANE1
          PO(*LANE2)
          ML(10,1)
          ML(40,5)
          CL(*LANE2)
          MD(1,5)
          RV(@SHOPTIME,900,2700)
          AO(@SHOPTIME,-,530)
          AO(@SHOPTIME,-,240)
          ST(@SHOPTIME)
          MD(15,10)
          MD(1,5)
          MR(32,5)
          JP(:REGCHK)

:SHPLANE3 R=(*READY)                  ; see comments for :SHPLANE1
          PO(*LANE3)
          ML(10,1)
          ML(38,5)
          CL(*LANE3)
          MD(1,5)
          RV(@SHOPTIME,900,2700)
          AO(@SHOPTIME,-,510)
          AO(@SHOPTIME,-,240)
          ST(@SHOPTIME)
          MD(15,10)
          MD(1,5)
          MR(30,5)
          JP(:REGCHK)

:SHPLANE4 R=(*READY)                  ; see comments for :SHPLANE1
          PO(*LANE4)
          ML(10,1)
          ML(30,5)
          CL(*LANE4)
          MD(1,5)
          RV(@SHOPTIME,900,2700)
          AO(@SHOPTIME,-,430)
          AO(@SHOPTIME,-,240)
          ST(@SHOPTIME)
          MD(15,10)
          MD(1,5)
          MR(22,5)
          JP(:REGCHK)

:SHPLANE5 R=(*READY)                  ; see comments for :SHPLANE1
          PO(*LANE5)
          ML(10,1)
          ML(28,5)
          CL(*LANE5)
          MD(1,5)
          RV(@SHOPTIME,900,2700)
          AO(@SHOPTIME,-,410)
          AO(@SHOPTIME,-,240)
          ST(@SHOPTIME)
          MD(15,10)
          MD(1,5)
          MR(20,5)
          JP(:REGCHK)

:SHPLANE6 R=(*READY)                  ; see comments for :SHPLANE1
          PO(*LANE6)
          ML(10,1)
          ML(20,5)
          CL(*LANE6)
          MD(1,5)
          RV(@SHOPTIME,900,2700)
          AO(@SHOPTIME,-,330)
          AO(@SHOPTIME,-,240)
          ST(@SHOPTIME)
          MD(15,10)
          MD(1,5)
          MR(12,5)
          JP(:REGCHK)
                                           ; see comments for :SHPLANE1
:SHPLANE7 R=(*READY)
          PO(*LANE7)
          ML(10,1)
          ML(18,5)
          CL(*LANE7)
          MD(1,5)
          RV(@SHOPTIME,900,2700)
          AO(@SHOPTIME,-,310)
          AO(@SHOPTIME,-,240)
          ST(@SHOPTIME)
          MD(15,10)
          MD(1,5)
          MR(10,5)
          JP(:REGCHK)
                                           ; see comments for :SHPLANE1
:SHPLANE8 R=(*READY)
          PO(*LANE8)
          ML(10,1)
          ML(10,5)
          CL(*LANE8)
          MD(1,5)
          RV(@SHOPTIME,900,2700)
          AO(@SHOPTIME,-,230)
          AO(@SHOPTIME,-,240)
          ST(@SHOPTIME)
          MD(15,10)
          MD(1,5)
          MR(2,5)
          JP(:REGCHK)
                                           ; see comments for :SHPLANE1
:SHPLANE9 R=(*READY)
          PO(*LANE9)
          ML(10,1)
          ML(8,5)
          CL(*LANE9)
          MD(1,5)
          RV(@SHOPTIME,900,2700)
          AO(@SHOPTIME,-,210)
          AO(@SHOPTIME,-,240)
          ST(@SHOPTIME)
          MD(15,10)
          MD(1,5)
          JP(:REGCHK)

:REGCHK   R=(*REGLINE)
          IV(@TTLREGQ)               ; total num of entries into queue
          IV(@REGQUEUE)              ; current number in the queue
          SV(OBJ%1,CLOCK)            ; start time in reg line queue
                                     ; jump if current is not a maximum
          IF(@REGQUEUE,LT,@MAXREGQ,THEN,:DOWN1)
          SV(@MAXREGQ,@REGQUEUE)     ; update the queue size maximum

:DOWN1    MR(6,8)                    ; move to point common with express
          MR(1,9)                    ; extra move to balance timing
          MR(18,1)                   ; move at exp rate thru common rt
          MR(6,9)                    ; move to reg checkout aisle
          MU(12,9)                   ; move up to checkout counter
          DV(@REGQUEUE)              ; leave the queue
          SV(%DIFFRNCE,CLOCK)        ; end time in reg line queue
          AO(%DIFFRNCE,-,OBJ%1)      ; time spent in queue
          AO(%DIFFRNCE,-,237)        ; required time in queue
          AO(%TTLREGQT,+,%DIFFRNCE)  ; update tot of times spent in que
          RV(@REGCHK,180,420)        ; determine ckout time (5+/-2 min)
          ST(@REGCHK)                ; delay during checkout
          DV(@CRTINUSE)              ; cart is no longer being used
          SV(%DIFFRNCE,CLOCK)        ; end time in cart storage
          AO(%DIFFRNCE,-,OBJ%2)      ; time spent in storage
          AO(%TTLCRTST,+,%DIFFRNCE)  ; update tot of times spent in stg
          IV(@NUMCARTS)              ; cart is free for use
          PV(*CRTSFREE,@NUMCARTS)    ; update number available
          MU(6,5)                    ; up to go out
          ML(7,5)                    ; left to common pt to leave store
          JP(:EXITSTOR)              ; transfer to route for reg & exp


:EXPLOOP  JC(1,*LANEA,:SHPLANEA)     ; to prevent shoppers who will be
          JC(1,*LANEB,:SHPLANEB)     ;  staying in the simulation for
          JC(1,*LANEC,:SHPLANEC)     ;  varying amounts of time from
                                     ;  interfering with one another,
                                     ;  each successive shopper object
                                     ;  is sent to a different lane
                                     ;  (assuming 3 lanes will be
                                     ;  sufficient for express shoppers)
          DN                         ; pause if all lanes occupied
          JP(:EXPLOOP)               ; try again

:SHPLANEA R=(*READY)                 ; reference for relative moves
          PO(*LANEA)                 ; post reserve position of lane A
          ML(10,1)                   ; move to lane A (10 * 1 = 10 sec)
          CL(*LANEA)                 ; clear reserve position for object
          MD(1,1)                    ; assume reserve position (1 sec)
          RV(@SHOPTIME,60,660)       ; determine shppng time (6+/-5 min)
          AO(@SHOPTIME,-,37)         ; subtract time spent by express
                                     ;  shoppers peculiar to lane A
                                     ;  (10 + 1 + 15 + 1 + 10 = 37 sec)
          AO(@SHOPTIME,-,21)          ; time common to all exp shoppers
          ST(@SHOPTIME)               ; delay at reserve pos for the rest
          MD(15,1)                    ; move down lane (15 * 1 = 15 sec)
          MD(1,1)                     ; move to back wall (1 sec)
          MR(10,1)                    ; move to common position for exp
                                      ;  shoppers (10 * 1 = 10 sec)
          JP(:EXPCHK)                 ; transfer to common rt for exp

:SHPLANEB R=(*READY)                  ; see comments for :SHPLANEA
          PO(*LANEB)
          ML(8,1)
          CL(*LANEB)
          MD(1,1)
          RV(@SHOPTIME,60,660)
          AO(@SHOPTIME,-,33)
          AO(@SHOPTIME,-,21)
          ST(@SHOPTIME)
          MD(15,1)
          MD(1,1)
          MR(8,1)
          JP(:EXPCHK)

:SHPLANEC R=(*READY)                  ; see comments for :SHPLANE1
          MD(1,1)                     ;  (note that lane C is in the same
          RV(@SHOPTIME,60,660)        ;  col as *READY so no need
          AO(@SHOPTIME,-,17)          ;  to move left & right to and from
          AO(@SHOPTIME,-,21)          ;  the column, as in lanes A & B;
          ST(@SHOPTIME)               ;  further, as *READY is directly
          MD(15,1)                    ;  above the reserve pos for lane
          MD(1,1)                     ;  C, there is no need to post it
          JP(:EXPCHK)                 ;  since shpper is already here)

:EXPCHK   R=(*EXPLINE)
          IV(@TTLEXPQ)                ; total num of entries into queue
          IV(@EXPQUEUE)               ; current number in the queue
          SV(OBJ%1,CLOCK)             ; start time in express queue
                                      ; jump if current is not a maximum
          IF(@EXPQUEUE,LT,@MAXEXPQ,THEN,:DOWN2)
          SV(@MAXEXPQ,@EXPQUEUE)      ; update the queue size maximum

:DOWN2    MR(6,1)                     ; move to exp checkout aisle
          MU(12,1)                    ; move up to checkout counter
          DV(@EXPQUEUE)               ; leave the queue
          SV(%DIFFRNCE,CLOCK)         ; end time in exp line queue
          AO(%DIFFRNCE,-,OBJ%1)       ; time spent in queue
          AO(%DIFFRNCE,-,18)          ; required time in queue
          AO(%TTLEXPQT,+,%DIFFRNCE)   ; update tot of times spent in que
          RV(@EXPCHK,60,180)          ; determine ckout time (2+/-1 min)
          ST(@EXPCHK)                 ; delay during checkout
          DV(@CRTINUSE)               ; cart is no longer being used
          SV(%DIFFRNCE,CLOCK)         ; end time in cart storage
          AO(%DIFFRNCE,-,OBJ%2)       ; time spent in storage
          AO(%TTLCRTST,+,%DIFFRNCE)   ; update tot of times spent in stg
          IV(@NUMCARTS)               ; cart is free for use
          PV(*CRTSFREE,@NUMCARTS)     ; update number available
          MU(6,5)                     ; up to go out
          JP(:EXITSTOR)               ; trans to common rt for reg & exp

:EXITSTOR R=(*EXIT)
          ML(5,5)                     ; head for the door
          RM(-2,+0,5)                 ; jump over position common with
                                      ;  incoming customers
          ML(31,5)                    ; move left to aisle of door
          MU(4,5)                     ; move to the door

          IV(@TERMCNT)                ; update num completing simulation

          ER                          ; exit the store & the simulation
```