APPENDIX 4: MUSIER AND EVANS SCHEDULING ALGORITHM CODE

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//      ME_Heuristic.cpp                Musier and Evans Algorithm                                                        //
//                                                                                                                        //
//      Progammer:              Niem-Trung L. Tra                                                                         //
//      Compiler:               Microsoft Visual C++ ver 5.0                                                              //
//      Platform:               Pentium II 350, Windows 98                                                                //
//      Date:                   October 1999                                                                              //
//                                                                                                                        //
//      This algorithm schedules jobs randomly.                                                                           //
//                                                                                                                        //
//      First, the program reads an external file named "orders.txt" consisting of:                                       //
//              Customer Number                                                                                           //
//              Product Type                                                                                              //
//              Due Date                                                                                                  //
//              Customer Order Quantity  (in batches)                                                                     //
//      Next, the customer orders are assigned a random number and then the orders are sorted                             //
//      by the assigned random number in accending order, list N. Next each job is placed on                              //
//      another list, S, and the partial schedule tardiness is tested. If there is no tardy jobs, then                    //
//      the next job on list N is placed on list S and the new partial schedule sequence is tested                        //
//      for tardiness.  If a job from list N cannot be added to the schedule without causing                              //
//      tardiness, it is removed from list S until all other non-tardy jobs have been scheduled.                          //
//      Finally, the production schedule is saved to a file named "schedule.txt".                                         //
//                                                                                                                        //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

#include <fstream.h>                    // for file I/O
#include <iomanip.h>                    // allows use of setw and setprecision
#include <stdlib.h>                     // allows use of rand()
#include <time.h>                       // allows use of clock to determine CPU time

        int     index = 0,              // position where item belongs
                j,
                list=0,                 // partial list length
                count,                  // loop control variable
                end = 0,                // end of the possible list of jobs
                total,                  // total jobs on partial sequence list
                found = 0,              // 0 = position not found, 1 = position found
                length = 0,             // length of ordered list
                change = 0,             // number of changeovers
                random,
                custNumber,             // customer number
                prodType,               // product type ordered
                dueDate,                // order due date
                quantity;               // quantity customer ordered in batches

        double  totTardy = 0,           // total hours of tardy jobs
                scheduleTime,
                tardyTime,
                lateTime[100][4],       // customer #, type, time in weeks, tardy
                temp[100][5],
                order[100][5];          // custNumber, prodType, dueDate, quantity

        clock_t beginScheduleTime,      // CPU time to find schedule
                endScheduleTime,
```

```
                beginTardyTime,         // CPU time to find total tardiness time
                endTardyTime;

        ifstream inorders;              // input file with customer order information
        ofstream outschedule;           // output file for production schedule
        ofstream outtardy;              //
        ofstream outimprovschd;         //

void main()
{
        beginScheduleTime = clock();
        void shift();                   // subroutine to shift job list down
        void shiftup();
        void changeover();              // subroutine to add changeover "jobs"
        void write();                   // subroutine to write schedule to file
        void tardiness();               // subroutine to calculate tardiness

        inorders.open("book1.txt");             // open input file
        if (!inorders)
        {
                cout << "** Can't open input file **" << endl;
                return;
        }

        outschedule.open("schedule.txt");       // open output file
        if (!outschedule)
        {
                cout << "** Can't open output file**" << endl;
                return;
        }

        outschedule.setf(ios::fixed, ios::floatfield);  // turn on manipulators

        outtardy.open("tardy.txt");             // open output file
        if (!outtardy)
        {
                cout << "** Can't open output file**" << endl;
                return;
        }

        outtardy.setf(ios::fixed, ios::floatfield);     // turn on manipulators

        outimprovschd.open("improvschd.txt");   // open output file
        if (!outimprovschd)
        {
                cout << "** Can't open output file**" << endl;
                return;
        }

        outimprovschd.setf(ios::fixed, ios::floatfield);        // turn on manipulators

// Read first line of input file and puts it on the ordered list.
// There is an assumption made in this program that the data file has
// been setup correctly to input the correct data types.

        srand(1);
```

```
        inorders >> order[0][0] >> order[0][1] >> order[0][2] >> order[0][3];
        order[0][4] = rand()%100;
        length++;

// Read next line of input file.

        inorders >> custNumber >> prodType >> dueDate >> quantity;
        random = rand()%100;

// Check to see if there are anymore jobs to schedule.

        while (dueDate!=-999)
        {

// Find where it goes in ordered list by random number.

                while (found == 0)
                {
                        if (random >= order[index][4])
                        {
                                index++;
                                if (index == length)
                                {
                                        found = 1;
                                }
                        }
                        else
                        {
                                found = 1;
                        }
                }

// Shift list[index...length-1] down one if necessary.

                shift();

// Insert customer order information into ordered list and increment list length.

                order[index][0] = custNumber;
                order[index][1] = prodType;
                order[index][2] = dueDate;
                order[index][3] = quantity;
                order[index][4] = random;
                length++;

// Read another line of input from file;

                inorders >> custNumber >> prodType >> dueDate >> quantity;
                random = rand()%100;
                index = 0;
                found = 0;
        }

// Partial Sequence Job List
```

```
for (count = 0; count<length; count++)
{
	for (j = 0; j < 4; j++)
	{
		temp[count][j] = order[count][j];
		order[count][j] = 0;
	}
}

end = length;
length = 0;
order[length][0] = 0;
order[length][1] = 4;
order[length][2] = 1000;
order[length][3] = 4./5.;
length++;
change++;
tardiness();
total = length;

for (list = 0; list <= end; list++)
{
	if (total<length)
		total = length;
	if (totTardy==0)
	{
		if (list<end)
		{
			if (total>length)
			{
				index = length;
				length = total;
				shift();
				length = index;
				total++;
			}

			order[length][0] = temp[list][0];
			order[length][1] = temp[list][1];
			order[length][2] = temp[list][2];
			order[length][3] = temp[list][3];

			if (order[length-1][1]==order[length][1] || order[length-1][1]>3)
			{
				if (order[length-1][1]>3)
				{
					if (order[length][1]<3)
						order[length-1][1]=4;
					else
						order[length-1][1]=5;
				}
				length++;
				tardiness();
			}
			else
			{
```

```
                                        index = length;
                                        total++;
                                        length = total;
                                        changeover();
                                        change++;
                                        index++;
                                        length = index + 1;
                                        total++;
                                        tardiness();

for (count = 0; count<total; count++)
{
        if (count > length && order[count][0]==0)
                total--;
}
                                                }
                        }
                }
                else
                {
                        length--;
                        order[total][0] = order[length][0];
                        order[total][1] = order[length][1];
                        order[total][2] = order[length][2];
                        order[total][3] = order[length][3];

                        index = length;
                        length = total;
                        shiftup();
                        length = index;

                        if (order[length-1][1]>3)
                        {
                                length--;
                                index = length;
                                length = total;
                                total--;
                                shiftup();
                                length = index;
                        }

                        totTardy = 0;
                        if (list<end)
                                list--;
                }
}

//      Sort remaining jobs by EDD
for (index = length; index < total-1; index++)
{
        if (order[index][2]>order[index+1][2])
        {       custNumber = order[index][0];
                prodType = order[index][1];
                dueDate = order[index][2];
                quantity = order[index][3];
                order[index][0]=order[index+1][0];
```

```
                        order[index][1]=order[index+1][1];
                        order[index][2]=order[index+1][2];
                        order[index][3]=order[index+1][3];
                        order[index+1][0]=custNumber;
                        order[index+1][1]=prodType;
                        order[index+1][2]=dueDate;
                        order[index+1][3]=quantity;
                        index = length-1;
                }
        }

// Add changeovers for remaining late jobs

        for (count = 0; count<total; count++)
        {
                if (count > length && order[count][0]==0)
                        total--;
        }

        for (index = length; index < total; index++)
        {
        if (order[index-1][1]!=order[index][1] && order[index-1][1]!=4 && order[index-1][1]!=5)
                {
                        length = total;
                        changeover();
                        total++;
                        change++;
                }
        }

        endScheduleTime = clock();
        scheduleTime = double (endScheduleTime - beginScheduleTime)/CLOCKS_PER_SEC;
        length = total;

// Calculate Completion Date
        beginTardyTime = clock();
        tardiness();
        endTardyTime = clock();
        tardyTime = double (endTardyTime - beginTardyTime)/CLOCKS_PER_SEC;

// Writing schedule to output file.

        write();

        inorders.close();                       // close input and output files
        outschedule.close();
        outtardy.close();
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void shift()        // shifts job list down
{
        for (count = length - 1; count >= index; count--)
                {
```

```
                        order[count+1][0] = order[count][0];
                        order[count+1][1] = order[count][1];
                        order[count+1][2] = order[count][2];
                        order[count+1][3] = order[count][3];
                        order[count+1][4] = order[count][4];
                }
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void shiftup()          // shifts the schedule up a unit
{
        for (count = index; count <= length ; count++)
                {
                        order[count][0] = order[count+1][0];
                        order[count][1] = order[count+1][1];
                        order[count][2] = order[count+1][2];
                        order[count][3] = order[count+1][3];
                        order[count][4] = order[count+1][4];
                }
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void changeover()       // add changeover 'jobs' to the schedule
{

// Insert first changeover for partial job list
        shift();

        order[index][0] = 0;
        order[index][2] = 1000;
        order[index][3] = 4./5.;

        if(order[index+1][1]==1 || order[index+1][1]==2)
                order[index][1] = 4;
        else
                order[index][1] = 5;
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void tardiness()        // determines the tardiness of a job based on 6.5 working hours/shift,
                        //                      3 shifts per day, 7 days per week
{
        const double    m14Change = 195./60,            // changeover time (shift-hr) for M14
                        rpChange = 243.75/60,           // changeover time (shift-hr) for RP910
                        mix865 = 128.25/60,             // process time / batch of M865
                        mix831 = 152.02/60,             // process time / batch of M831

                        mix910 = 196.52/60,             // process time / batch of RP910
                        cutter = 15.38/60;              // dehy process time / batch
        double          process = 0.;                   // total process time for all jobs
```

```
	void tardy(double process);

	for (count = 0; count < length; count++)
	{
		lateTime[count][0] = order[count][0];
		lateTime[count][1] = order[count][1];

		if (order[count][1]==4)
		{
			process = process + m14Change;
			lateTime[count][2] = process;
			lateTime[count][3] = 0;
		}
		else if (order[count][1]==5)
		{
			process = process + rpChange;
			lateTime[count][2] = process;
			lateTime[count][3] = 0;
		}
		else if (order[count][1]==1)
		{
			if (order[count-1][1]>3)
				process = process + (mix865+cutter*(order[count][3]-1));
			else
				process = process + (cutter*order[count][3]);
			lateTime[count][2] = process;
			tardy(process);
		}
		else if (order[count][1]==2)
		{
			if (order[count-1][1]>3)
				process = process + (mix831+cutter*(order[count][3]-1));
			else
				process = process + (cutter*order[count][3]);
			lateTime[count][2] = process;
			tardy(process);
		}
		else if (order[count][1]==3)
		{
			if (order[count-1][1]>3)
				process = process + (mix910+cutter*(order[count][3]-1));
			else
				process = process + (cutter*order[count][3]);
			lateTime[count][2] = process;
			tardy(process);
		}
	}
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void tardy(double process)
{
	double	late;
```

```
		late = process - order[count][2];
		if (late < 0)
			lateTime[count][3] = 0;
		else
			lateTime[count][3] = late;

		totTardy = totTardy + lateTime[count][3];
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void write()		// write schedule to file
{
		for (count = 0; count<length; count++)
		{
			for (index = 0; index < 5.*order[count][3]; index++)
				outschedule << order[count][0] << " "<< order[count][1] << " "
					<< order[count][2] << " "<< order[count][3] << endl;
		}

		for (count = 0; count< length; count++)
		{
			for (index = 0; index < 4; index++)
			{
				outimprovschd<<order[count][index]<<" ";
				outtardy << lateTime[count][index] << "  ";
				cout << lateTime[count][index] << "  ";
			}
			outimprovschd<<endl;
			outtardy <<order[count][2]<<endl;
			cout << endl;
		}
		outimprovschd<<0<<" "<<0<<" "<<0<<" "<<-999;
		cout << "Total Tardiness (Hours): " << totTardy << endl;
		cout << "CPU time to develop schedule: " <<
			setw(10) << setprecision(8) << scheduleTime << endl;
		cout << "CPU time to determine total tardy hours: " << tardyTime << endl;
                                                }
```

APPENDIX 5: KU AND KARIMI SEQUENCE BUILDING ALGORITHM CODE

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//      KK_Heuristic.cpp                Ku and Karimi Algorithm                                                           //
//                                                                                                                        //
//      Programmer:             Niem-Trung L. Tra                                                                         //
//      Compiler:               Microsoft Visual C++ ver 5.0                                                              //
//      Platform:               Pentium II 350, Windows 98                                                                //
//      Date:                   October 1999                                                                              //
//                                                                                                                        //
//      This algorithm schedules jobs based on inserting each job in all possible positions of a                          //
//      good partial sequence.                                                                                            //
//                                                                                                                        //
//      First, the program reads an external file named "book2.txt" consisting of:                                        //
//              Customer Number                                                                                           //
//              Product Type                                                                                              //
//              Due Date                                                                                                  //
//              Customer Order Quantity  (in batches)                                                                     //
//      Next, the customer orders are sorted by earliest due date, list order. Next each job is                          //
//      placed on another list, sequence, in every position and the partial schedule tardiness is                        //
//      tested. The sequence w/ the minimal tardy jobs is selected and the next job is added to                           //
//      the sequence. If the sequence tie for minimal tardy jobs, the first sequence generated is                         //
//      selected. This process is continued until all jobs, form the N list, have been placed on                          //
//      the S list. Finally, the production schedule is saved to a file named "schedule.txt".                             //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

#include <fstream.h>                                    // for file I/O
#include <iomanip.h>                                    // allows use of setw and setprecision
#include <stdlib.h>                                     // allows use of rand()
#include <time.h>                                       // allows use of clock to determine CPU time

        int             i,                              // index for where new job is placed
                        oldi,                           // original i value
                        j,                              // index for array position
                        index = 0,                      // position where item belongs
                        list=0,                         // index for jobs placed
                        count,                          // number of jobs
                        end = 0,                        // end of the possible list of jobs
                        found = 0,                      // 0 = position not found, 1 = position found
                        origlength,                     // original length before inserting new job
                        length = 0,                     // length of ordered list
                        change = 0,                     // number of changeovers
                        minSequence,                    // sequence that gives minimal tardy hours
                        custNumber,                     // customer number
                        prodType,                       // product type ordered
                        dueDate,                        // order due date
                        quantity;                       // quantity customer ordered in batches
        double          totTardy = 0,                   // total hours of tardy jobs
                        minTotTardy = 1000000,          // minimal tardy hours
                        scheduleTime,
                        tardyTime,
                        lateTime[100][4],               // customer #, type, time in weeks, tardy
                        sequence[100][5],               // custNumber, prodType, dueDate, quantity
                        temp[100][5],                   // custNumber, prodType, dueDate, quantity
                        order[100][5];                  // custNumber, prodType, dueDate, quantity
```

```
        clock_t     beginScheduleTime,      // CPU time to find schedule
                    endScheduleTime,
                    beginTardyTime,         // CPU time to find total tardiness time
                    endTardyTime;

        ifstream inorders;                  // input file with customer order information
        ofstream outschedule;               // output file for production schedule
        ofstream outtardy;
        ofstream outimprovschd;

void main()
{
        beginScheduleTime = clock();
        void shift();                       // subroutine to shift job list down
        void changeover();                  // subroutine to add changeover "jobs"
        void copyjoblist();                 // subroutine to copy job order to new list
        void write();                       // subroutine to write schedule to file
        void tardiness();                   // subroutine to calculate tardiness

        inorders.open("book1.txt");         // open input file
        if (!inorders)
        {
                cout << "** Can't open input file **" << endl;
                return;
        }

        outschedule.open("schedule.txt");   // open output file
        if (!outschedule)
        {
                cout << "** Can't open output file**" << endl;
                return;
        }

        outschedule.setf(ios::fixed, ios::floatfield);  // turn on manipulators

        outtardy.open("tardy.txt");         // open output file
        if (!outtardy)
        {
                cout << "** Can't open output file**" << endl;
                return;
        }

        outtardy.setf(ios::fixed, ios::floatfield);  // turn on manipulators

        outimprovschd.open("improvschd.txt");  // open output file
        if (!outimprovschd)
        {
                cout << "** Can't open output file**" << endl;
                return;
        }

        outimprovschd.setf(ios::fixed, ios::floatfield);  // turn on manipulators

// Read first line of input file and puts it on the ordered list.
// There is an assumption made in this program that the data file has
```

```
// been setup correctly to input the correct data types.

        inorders >> order[0][0] >> order[0][1] >> order[0][2] >> order[0][3];
        length++;

// Read next line of input file.

        inorders >> custNumber >> prodType >> dueDate >> quantity;

// Check to see if there are anymore jobs to schedule.

        while (dueDate!=-999)
        {

// Find where it goes in ordered list by EDD.

                while (found == 0)
                {
                        if (dueDate > order[index][2])
                        {
                                index++;
                                if (index == length)
                                {
                                        found = 1;
                                }
                        }
                        else if (dueDate == order[index][2])
                        {
                                if (quantity > order[index][3])
                                        found = 1;
                                else
                                {
                                        index++;
                                        if (index == length)
                                        {
                                                found = 1;
                                        }
                                }
                        }
                        else
                        {
                                found = 1;
                        }
                }

// Shift list[index...length-1] down one if necessary.

                shift();

// Insert customer order information into ordered list and increment list length.

                order[index][0] = custNumber;
                order[index][1] = prodType;
                order[index][2] = dueDate;
                order[index][3] = quantity;
```

```
                length++;

// Read another line of input from file;

                inorders >> custNumber >> prodType >> dueDate >> quantity;
                index = 0;
                found = 0;
        }

// Partial Sequence Job List

        for (count = 0; count<length; count++)
        {
                for (j = 0; j < 4; j++)
                {
                        temp[count][j] = order[count][j];
                        order[count][j] = sequence[count][j];
                }
        }

        end = length;
        length = 0;
        order[length][0] = temp[list][0];
        order[length][1] = temp[list][1];
        order[length][2] = temp[list][2];
        order[length][3] = temp[list][3];
        length++;
        changeover();                                   // adding initial changeover
        change++;

        for (index = 0; index<=length; index++)         // initial partial sequence list
        {
                for (j = 0; j < 4; j++)
                        sequence[index][j] = order[index][j];
        }

        for (list = 1; list < end; list++)              // for all jobs ordered
        {
                minTotTardy = 1000000;
                length++;
                origlength = length;
                for (i = 1; i <= length; i++)           // for all postions in partial sequence
                {       oldi = i;

                if (i<length)                           // insert job in first i-th position
                        {
                                index = i;
                                shift();
                        }
                        order[i][0] = temp[list][0];
                        order[i][1] = temp[list][1];
                        order[i][2] = temp[list][2];
                        order[i][3] = temp[list][3];

                // determine if a CO needs to be added
                        if (order[i-1][1]>3||order[i][1]==order[i-1][1])        // if job same as previous
```

```
                {
                        if (order[i-1][1]>3)
                        {
                                if (order[i][1]<3)
                                        order[i-1][1]=4;
                                else
                                        order[i-1][1]=5;
                        }
                }
                else
                {
                        length++;
                        index = i;
                        changeover();
                        change++;
                        i++;
                }

                if (i!=length)
                {                                       // if job same as next
                        if (order[i+1][1]<4 && order[i][1]!=order[i+1][1])
                        {
                                index=i+1;
                                length++;
                                changeover();
                                index=i;
                                change++;
                        }
                }

                tardiness();
                totTardy = 0;
                length = origlength;
                copyjoblist();
                i = oldi;
        }
// defining good partial sequence to use for next iteration
        index = minSequence;
        shift();
        order[index][0] = temp[list][0];
        order[index][1] = temp[list][1];
        order[index][2] = temp[list][2];
        order[index][3] = temp[list][3];

        for (index = 0; index<=length; index++)       // partial sequence list w/ changeovers added
        {if (index > 0)
                {if(order[index][1]==order[index-1][1]||order[index-1][1]>3||order[index][1]>3)
                        {       for (j = 0; j < 4; j++)
                                        sequence[index][j] = order[index][j];
                        }
                        else
                        {
                                length++;
                                changeover();
                                for (j = 0; j < 4; j++)
                                        sequence[index][j] = order[index][j];
```

```
                                        }
                                }
                                else
                                {
                                        for (j = 0; j < 4; j++)
                                                sequence[index][j] = order[index][j];
                                }
                        }
                }

                endScheduleTime = clock();
                scheduleTime = double (endScheduleTime - beginScheduleTime)/CLOCKS_PER_SEC;

// Calculate Completion Date
                beginTardyTime = clock();
                tardiness();
                endTardyTime = clock();
                tardyTime = double (endTardyTime - beginTardyTime)/CLOCKS_PER_SEC;

// Writing schedule to output file.

                write();

                inorders.close();                                       // close input and output files
                outschedule.close();
                outtardy.close();
                outimprovschd.close();
}

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void shift()            // shifts job list down
{
        for (count = length - 1; count >= index; count--)
                {
                        order[count+1][0] = order[count][0];
                        order[count+1][1] = order[count][1];
                        order[count+1][2] = order[count][2];
                        order[count+1][3] = order[count][3];
                }
}

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void copyjoblist()// copy job list from order[][] to sequence[][]
{
        for (index = 0; index<length; index++)                  // refresh partial sequence list
        {
                for (j = 0; j < 4; j++)
                        order[index][j] = sequence[index][j];
        }
}

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void changeover()                                       // add changeover 'jobs' to the schedule
{
        shift();
        order[index][0] = 0;
        order[index][2] = 1000;
        order[index][3] = 4./5.;

        if(order[index+1][1]==1 || order[index+1][1]==2)
                order[index][1] = 4;
        else
                order[index][1] = 5;
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void tardiness()        // determines the tardiness of a job based on 6.5 working hours/shift,
                        //              3 shifts per day, 7 days per week
{
        const double    P1Change = 195./60,             // changeover time (shift-hr) for M14
                        P3Change = 243.75/60,           // changeover time (shift-hr) for RP910
                        P1 = 128.25/60,                 // process time / batch of M865
                        P2 = 152.02/60,                 // process time / batch of M831
                        P3 = 196.52/60,                 // process time / batch of RP910
                        dehy = 15.38/60;                // dehy process time / batch
        double          process = 0.;                   // total process time for all jobs

        void tardy(double process);

        for (count = 0; count <= length; count++)
        {
                lateTime[count][0] = order[count][0];
                lateTime[count][1] = order[count][1];

                if (order[count][1]==4)
                {
                        process = process + P1Change;
                        lateTime[count][2] = process;
                        lateTime[count][3] = 0;
                }
                else if (order[count][1]==5)
                {
                        process = process + P3Change;
                        lateTime[count][2] = process;
                        lateTime[count][3] = 0;
                }
                else if (order[count][1]==1)
                {
                        if (order[count-1][1]>3)
                                process = process + (P1+dehy*(order[count][3]-1));
                        else
                                process = process + (dehy*order[count][3]);
                        lateTime[count][2] = process;
                        tardy(process);
```

```
                }
                else if (order[count][1]==2)
                {
                        if (order[count-1][1]>3)
                                process = process + (P2+dehy*(order[count][3]-1));
                        else
                                process = process + (dehy*order[count][3]);
                        lateTime[count][2] = process;
                        tardy(process);
                }
                else if (order[count][1]==3)
                {
                        if (order[count-1][1]>3)
                                process = process + (P3+dehy*(order[count][3]-1));
                        else
                                process = process + (dehy*order[count][3]);
                        lateTime[count][2] = process;
                        tardy(process);
                }
        }
        if (totTardy < minTotTardy)
        {
                if (totTardy==0)
                        i=length+1;
                minTotTardy = totTardy;
                minSequence = index;
        }
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void tardy(double process)
{
        double   late;

        late = process - order[count][2];

        if (late < 0)
                lateTime[count][3] = 0;
        else
                lateTime[count][3] = late;

        totTardy = totTardy + lateTime[count][3];
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void write()                                                    // write schedule to file
{
        for (count = 0; count<=length; count++)
        {
                for (index = 0; index < 5.*sequence[count][3]; index++)
                        outschedule << sequence[count][0] << " "<< sequence[count][1] << " "
                                << sequence[count][2] << " "<< sequence[count][3] << endl;
```

```
		}

		for (count = 0; count<= length; count++)
		{
			for (index = 0; index < 4; index++)
			{
				outimprovschd << sequence[count][index] << " ";
				outtardy << lateTime[count][index] << "  ";
				cout << lateTime[count][index] << "  ";
			}
			outimprovschd<<endl;
			outtardy << endl;
			cout << endl;
		}
		outimprovschd<<0<<" "<<0<<" "<<0<<" "<<-999;
		cout << "Total Tardiness (Hours): " << totTardy << endl;
		cout << "CPU time to develop schedule: " <<
			setw(10) << setprecision(8) << scheduleTime << endl;
		cout << "CPU time to determine total tardy hours: " << tardyTime << endl;
}
```

APPENDIX 6: GREEDY HEURISTIC ALGORITHM CODE

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//      EDDHeuristic.cpp                        Greedy Heuristic                                                                  //
//                                                                                                                                //
//      Programmer:             Niem-Trung L. Tra                                                                                 //
//      Compiler:               Microsoft Visual C++ ver 5.0                                                                      //
//      Platform:               Pentium II 350, Windows 98                                                                        //
//      Date:                   September 1999                                                                                    //
//                                                                                                                                //
//      This algorithm schedules jobs in the order of earliest due date.                                                          //
//                                                                                                                                //
//      First, the program reads an external file named "book2.txt" consisting of:                                                //
//              Customer Number                                                                                                   //
//              Product Type                                                                                                      //
//              Due Date                                                                                                          //
//              Customer Order Quantity  (in batches)                                                                             //
//      Next, the customer orders are sorted by earliest due date, forming the production schedule.                               //
//      Finally, the production schedule is saved to a file named "schedule.txt".                                                 //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

#include <fstream.h>                                    // for file I/O
#include <iomanip.h>                                    // allows use of setw and setprecision
#include <time.h>                                       // allows use of clock to determine CPU time

        int             index = 0,                      // position where item belongs
                        count,                          // loop control variable
                        found = 0,                      // 0 = position not found, 1 = position found
                        length = 0,                     // length of ordered list
                        change = 0,                     // number of changeovers
                        custNumber,                     // customer number
                        prodType,                       // product type ordered
                        dueDate,                        // order due date
                        quantity;                       // quantity customer ordered in batches

        double          totTardy = 0,                   // total weeks tardy
                        scheduleTime,
                        tardyTime,
                        lateTime[100][4],               // customer #, type, time in weeks, tardy
                        order[100][4];                  // custNumber, prodType, dueDate, quantity

        clock_t         beginScheduleTime,              // CPU time to find schedule
                        endScheduleTime,
                        beginTardyTime,                 // CPU time to find total tardiness time
                        endTardyTime;

        ifstream inorders;                              // input file with customer order information
        ofstream outschedule;                           // output file for production schedule
        ofstream outtardy;                              // output file for production schedule
        ofstream outimprovschd;                         // output file for improvement

void main()
{
        beginScheduleTime = clock();
        void shift();                                   // subroutine to shift job list down
        void changeover();                              // subroutine to add changeover "jobs"
```

```
        void write();                                   // subroutine to write schedule to file
        void tardiness();                               // subroutine to calculate tardiness
        inorders.open("book1.txt");                     // open input file
        if (!inorders)
        {
                cout << "** Can't open input file **" << endl;
                return;
        }

        outschedule.open("schedule.txt");               // open output file
        if (!outschedule)
        {
                cout << "** Can't open output file**" << endl;
                return;
        }

        outschedule.setf(ios::fixed, ios::floatfield);  // turn on manipulators

        outtardy.open("tardy.txt");                     // open output file
        if (!outtardy)
        {
                cout << "** Can't open output file**" << endl;
                return;
        }

        outtardy.setf(ios::fixed, ios::floatfield);     // turn on manipulators

        outimprovschd.open("improvschd.txt");           // open output file
        if (!outimprovschd)
        {
                cout << "** Can't open output file**" << endl;
                return;
        }

        outimprovschd.setf(ios::fixed, ios::floatfield); // turn on manipulators

// Read first line of input file and puts it on the ordered list.
// There is an assumption made in this program that the data file has
// been setup correctly to input the correct data types.

        inorders >> order[0][0] >> order[0][1] >> order[0][2] >> order[0][3];
        length++;

// Read next line of input file.

        inorders >> custNumber >> prodType >> dueDate >> quantity;

// Check to see if there are anymore jobs to schedule.

        while (dueDate!=-999)
        {

// Find where it goes in ordered list by EDD.

                while (found == 0)
                {
```

```
                        if (dueDate > order[index][2])
                        {
                                index++;
                                if (index == length)
                                {
                                        found = 1;
                                }
                        }
                        else if (dueDate == order[index][2])
                        {
                                if (quantity > order[index][3])
                                        found = 1;
                                else
                                {
                                        index++;
                                        if (index == length)
                                        {
                                                found = 1;
                                        }
                                }
                        }
                        else
                        {
                                found = 1;
                        }
                }

// Shift list[index...length-1] down one if necessary.

                shift();

// Insert customer order information into ordered list and increment list length.

                order[index][0] = custNumber;
                order[index][1] = prodType;
                order[index][2] = dueDate;
                order[index][3] = quantity;

                length++;

// Read another line of input from file;

                inorders >> custNumber >> prodType >> dueDate >> quantity;
                index = 0;
                found = 0;
        }

// Add Changeover "Jobs"

        changeover();
        endScheduleTime = clock();
        scheduleTime = double (endScheduleTime - beginScheduleTime)/CLOCKS_PER_SEC;

// Calculate Completion Date

        beginTardyTime = clock();
```

```
	tardiness();
	endTardyTime = clock();
	tardyTime = double (endTardyTime - beginTardyTime)/CLOCKS_PER_SEC;

// Writing schedule to output file.

	write();

	inorders.close();                                   // close input and output files
	outschedule.close();
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void shift()                                            // shifts job list down
{
	for (count = length - 1; count >= index; count--)
		{
			order[count+1][0] = order[count][0];
			order[count+1][1] = order[count][1];
			order[count+1][2] = order[count][2];
			order[count+1][3] = order[count][3];
		}
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void changeover()                                       // adds changeover 'jobs' to the schedule
{
	int change;                                         // number of changeovers scheduled
	shift();

// Insert customer order information into ordered list and increment list length.
		order[index][0] = 0;
		order[index][2] = 1000;
		order[index][3] = 4./5.;

		if(order[index+1][1]==1 || order[index+1][1]==2)
			order[index][1] = 4;
		else
			order[index][1] = 5;

		length++;
		change++;

	for (index = 2; index<length; index++)
	{
		if (order[index-1][1]!=order[index][1] && order[index-1][1]!=4 && order[index-1][1]!=5)
		{
			shift();

// Insert customer order information into ordered list and increment list length.
				order[index][0] = 0;
				order[index][2] = 1000;
```

```
                                order[index][3] = 4./5.;

                                if(order[index+1][1]==1 || order[index+1][1]==2)
                                        order[index][1] = 4;
                                else
                                        order[index][1] = 5;
                                length++;
                                change++;
                        }
                }
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void tardiness()                        // determines the tardiness of a job based on 6.5 working hours/shift,
                                        //              3 shifts per day, 7 days per week
{
        const double    P1Change = 195./60,             // changeover time (shift-hr) for M14
                        P3Change = 243.75/60,           // changeover time (shift-hr) for RP910
                        P1 = 128.25/60,                 // process time / batch of M865
                        P2 = 152.02/60,                 // process time / batch of M831
                        P3 = 196.52/60,                 // process time / batch of RP910
                        dehy = 15.38/60;                // dehy process time / batch

        double          process = 0.;                   // total process time for all jobs

        void tardy(double process);

        for (count = 0; count < length; count++)
        {
                lateTime[count][0] = order[count][0];
                lateTime[count][1] = order[count][1];

                if (order[count][1]==4)
                {
                        process = process + P1Change;
                        lateTime[count][2] = process;
                        lateTime[count][3] = 0;
                }
                else if (order[count][1]==5)
                {
                        process = process + P3Change;
                        lateTime[count][2] = process;
                        lateTime[count][3] = 0;
                }
                else if (order[count][1]==1)
                {
                        if (order[count-1][1]>3)
                                process = process + (P1+dehy*(order[count][3]-1));
                        else
                                process = process + (dehy*order[count][3]);
                        lateTime[count][2] = process;
                        tardy(process);
                }
                else if (order[count][1]==2)
```

```
		{
			if (order[count-1][1]>3)
				process = process + (P2+dehy*(order[count][3]-1));
			else
				process = process + (dehy*order[count][3]);
			lateTime[count][2] = process;
			tardy(process);
		}
		else if (order[count][1]==3)
		{
			if (order[count-1][1]>3)
				process = process + (P3+dehy*(order[count][3]-1));
			else
				process = process + (dehy*order[count][3]);
			lateTime[count][2] = process;
			tardy(process);
		}
	}
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void tardy(double process)
{
	double	late;

	late = lateTime[count][2] - order[count][2];
	if (late < 0)
		lateTime[count][3] = 0;
	else
		lateTime[count][3] = late;

	totTardy = totTardy + lateTime[count][3];
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void write()                                                    // write schedule to file
{
	for (count = 0; count<length; count++)
	{
		for (index = 0; index < 5.*order[count][3]; index++)
			outschedule << order[count][0] << " "<< order[count][1] << " "
				<< order[count][2] << " "<< order[count][3] << endl;

		for (index = 0; index < 4; index++)
		{
			outimprovschd<<order[count][index]<<" ";
			outtardy << lateTime[count][index] << "  ";
			cout << lateTime[count][index] << "  ";
		}
		outimprovschd<<endl;
		outtardy << endl;
		cout << endl;
```

```
        }
        outimprovschd<<0<<" "<<0<<" "<<0<<" "<<-999;
        cout << "Total Tardiness (Hours): " << totTardy << endl;
        cout << "CPU time to develop schedule: " <<
                setw(10) << setprecision(8) << scheduleTime << endl;
        cout << "CPU time to determine total tardy hours: " << tardyTime << endl;
}
```

APPENDIX 7: MUSIER AND EVANS HEURISTIC IMPROVEMENT METHOD ALGORITHM CODE

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//      ME_HIM_Heuristic.cpp                    Musier and Evans' HIM Algorithm                                           //
//                                                                                                                        //
//      Programmer:             Niem-Trung L. Tra                                                                         //
//      Compiler:               Microsoft Visual C++ ver 5.0                                                              //
//      Platform:               Pentium II 350, Windows 98                                                                //
//      Date:                   November 1999                                                                             //
//                                                                                                                        //
//      This algorithm improves production schedules by performing 1-opt and 2-opt analysis.                              //
//                                                                                                                        //
//      First, the program reads an external file named "improvschd.txt" consisting of:                                   //
//              Customer Number                                                                                           //
//              Product Type                                                                                              //
//              Due Date                                                                                                  //
//              Customer Order Quantity  (in batches)                                                                     //
//      Next, each job is inserted in all possible positions to find a new good job sequence.  The one with               //
//      the new minimum total tardiness is used for the next iteration. This is then repeated for pair-wise               //
//      job exchanges. Finally, the production schedule is saved to a file named "schedule.txt".                          //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

#include <fstream.h>                                    // for file I/O
#include <iomanip.h>                                    // allows use of setw and setprecision
#include <stdlib.h>                                     // allows use of rand()
#include <time.h>                                       // allows use of clock to determine CPU time

        int             i,                              // index for where new job is placed
                        oldi,                           // original i value
                        j,                              // index for arry position
                        index = 0,                      // position where item belongs
                        list=0,                         // index for jobs placed
                        count,                          // number of jobs
                        end = 0,                        // end of the possible list of jobs
                        optlength,
                        origlength,                     // original length before inserting new job
                        length = 0,                     // length of ordered list
                        minSequence;                    // sequence that gives minimal tardy hours

        double          totTardy = 0,                   // total hours of tardy jobs
                        minTotTardy = 1000000,          // minimal tardy hours
                        scheduleTime,
                        tardyTime,
                        custNumber,                     // customer number
                        prodType,                       // product type ordered
                        dueDate,                        // order due date
                        quantity,                       // quantity customer ordered in batches
                        sequence[100][5],               // custNumber, prodType, dueDate, quantity
                        temp[100][5],                   // custNumber, prodType, dueDate, quantity
                        lateTime[100][4],               // customer #, type, time in weeks, tardy
                        order[100][5];                  // custNumber, prodType, dueDate, quantity

        clock_t         beginScheduleTime,              // CPU time to find schedule
                        endScheduleTime,
```

```
                    beginTardyTime,                 // CPU time to find total tardiness time
                    endTardyTime;

        ifstream inimprovschd;                      // input file with original schedule
        ofstream outschedule;                       // output file for improved schedule
        ofstream outtardy;

void main()
{
        beginScheduleTime = clock();
        void shiftup();                             // subroutine to shift job list up
        void shift();                               // subroutine to shift job list down
        void changeover();                          // subroutine to add changeover "jobs"
        void copyjoblist();                         // subroutine to copy job order to new list
        void copyjoblist2();                        // subroutine to copy job order to new list
        void write();                               // subroutine to write schedule to file
        void tardiness();                           // subroutine to calculate tardiness

        inimprovschd.open("improvschd.txt");        // open input file
        if (!inimprovschd)
        {
                cout << "** Can't open input file **" << endl;
                return;
        }

        outschedule.open("improved.txt");           // open output file
        if (!outschedule)
        {
                cout << "** Can't open output file**" << endl;
                return;
        }

        outschedule.setf(ios::fixed, ios::floatfield);  // turn on manipulators

        outtardy.open("tardy.txt");                 // open output file
        if (!outtardy)
        {
                cout << "** Can't open output file**" << endl;
                return;
        }

        outtardy.setf(ios::fixed, ios::floatfield);     // turn on manipulators

// Read input file and puts it on the ordered list.
// There is an assumption made in this program that the data file has
// been setup correctly to input the correct data types.

        inimprovschd >> custNumber >> prodType >> dueDate >> quantity;

        while (quantity!=-999)
        {
                order[length][0] = custNumber;
                order[length][1] = prodType;
                order[length][2] = dueDate;
                order[length][3] = quantity;
                length++;
```

```
                inimprovschd >> custNumber >> prodType >> dueDate >> quantity;
        }

// copy original sequence list to sequence list
        for (count = 0; count<length; count++)
        {       for (j = 0; j < 4; j++)
                {       sequence[count][j] = order[count][j];}          // optimal sequence list
        }
        end = length;
        tardiness();
        minTotTardy = totTardy;
        totTardy = 0;

// 1-optimal algorithm
for (list = 1; list < end; list++)                                      // for all jobs ordered
{
        origlength = 0;
        minSequence = list;
        custNumber      = sequence[list][0];
        prodType        = sequence[list][1];
        dueDate         = sequence[list][2];
        quantity = sequence[list][3];
        if (prodType<4)
        {
                if (list>1)
// check to see if there will be consecutive COs
                {if (list<end-1)
                        {if (sequence[list-1][1]>3 && sequence[list+1][1]>3)
                                {       index = list-1;
                                        origlength = 1;
                                }
                                else
                                        index = list;
                        }
                 else
                 {if (sequence[list-1][1]>3)
                                {       index = list-1;
                                        origlength = 1;
                                }
                                else
                                        index = list;
                 }
                }
                else
                        index = list;
                shiftup();

                if (sequence[list+1][1]>3)
                {       shiftup();
                        origlength = 1;
                }

                length = length-1-origlength;
                origlength = length;

                for (count = 0; count<length; count++)
```

```
		{	for (j = 0; j < 4; j++)				// initial sequence list
			{	temp[count][j] = order[count][j];
			}
		}

		for (i = 1; i < length; i++)				// for all postions in partial sequence
		{	// insert job in first i-th position
			index = i;
			if (i<length-1)					// checking for extra CO
			{	if (order[i][1]<4)
				{	shift();
					length++;
				}
			}
			order[i][0] = custNumber;
			order[i][1] = prodType;
			order[i][2] = dueDate;
			order[i][3] = quantity;

// determine if a CO needs to be added
			if (order[i-1][1]>3||order[i][1]==order[i-1][1])	// if job same as previous
			{if (order[i-1][1]>3)
				{	if (order[i][1]<3)
						order[i-1][1]=4;
					else
						order[i-1][1]=5;
				}
			}
			else
			{	changeover();
				length++;
				index++;
			}

			if (index<length)
			{	if (order[index+1][1]<4 && order[index][1]!=order[index+1][1])	// if job
same as next
				{	index=index+1;
					changeover();
					index=i;
					length++;
				}
			}

			tardiness();

			if (totTardy < minTotTardy)
			{	if (totTardy==0)
					i=length;
				minTotTardy = totTardy;
				minSequence = index;
			}

			totTardy = 0;
			length = origlength;
			copyjoblist();
```

```
                        if (order[i][1]>3)
        // skips CO jobs
                                i++;
                }

// defining good partial sequence to use for next iteration
                if (minSequence!=list)
                {
                        list = 0;
                        index = minSequence;
                        shift();
                        length++;
                        order[index][0] = custNumber;
                        order[index][1] = prodType;
                        order[index][2] = dueDate;
                        order[index][3] = quantity;

                        for (index = 0; index<length; index++)          // partial sequence list with CO added
                        {       if (index > 0)
                                {if (order[index][1]==order[index-1][1]||order[index-1][1]>3||order[index][1]>3)
                                        {       for (j = 0; j < 4; j++)
                                                        sequence[index][j] = order[index][j];
                                        }
                                        else
                                        {       changeover();
                                                length++;
                                                for (j = 0; j < 4; j++)
                                                        sequence[index][j] = order[index][j];
                                        }
                                }
                                else
                                {       for (j = 0; j < 4; j++)
                                                sequence[index][j] = order[index][j];
                                }
                        }
                        end = length;
                }
                else
                {       length = end;
                        for (count = 0; count<length; count++)
                        {       for (j = 0; j < 4; j++)
                                        order[count][j] = sequence[count][j];           // initial sequence list
                        }
                }
        }
}

// 2-optimal algorithm
// copy sequence list to order list
        for (count = 0; count<length; count++)
        {       for (j = 0; j < 4; j++)
                {       order[count][j] = sequence[count][j];}          // optimal sequence list
        }
        tardiness();
        minTotTardy = totTardy;
        totTardy = 0;
```

```
for (list = 1; list < length; list++)                    // for all jobs ordered
{
        origlength = length;
        minSequence = list;
        custNumber      = sequence[list][0];
        prodType        = sequence[list][1];
        dueDate         = sequence[list][2];
        quantity = sequence[list][3];
        if (prodType<4)
        {
                for (i = 1; i < length; i++)             // for all postions in partial sequence
                {                                        // exchange job in job in i-th position
                        if (order[i][1]>3)
                                i++;
                        if (i==list)
                                i++;
                        if (order[i][1]<4)
                                {       for (j = 0; j < 4; j++)
                                        {       order[list][j]=order[i][j];
                                                order[i][j]=sequence[list][j];
                                        }

// determine if a CO needs to be added/deleted/changed
                                for (j=1;j<length;j++)
                                {if(order[j][1]>3)
                                        j++;
                                 if (order[j-1][1]>3||order[j][1]==order[j-1][1])
                                                        // if job same as previous
                                        {if (order[j-1][1]>3)
                                                {if (order[j][1]<3)
                                                        // check for correct CO time
                                                        order[j-1][1]=4;
                                                else
                                                                order[j-1][1]=5;
                                                if (j>1)
                                                        // check to see if there is an extra CO
                                                        {if (order[j-2][1]==order[j][1])
                                                                {       index = j-1;
                                                                        shiftup();
                                                                        length--;
                                                                        j--;
                                                                }
                                                        }
                                                }
                                        }
                                 else                   // add CO
                                        {       index = j;
                                                changeover();
                                                length++;
                                        }
                                }

                                tardiness();

                                if (totTardy < minTotTardy)
```

```
                                        {       if (totTardy==0)
                                                        i=length;
                                                minTotTardy = totTardy;
                                                minSequence = i;
                                                optlength = length;
                                                for (count = 0; count<length; count++)
                                                {       for (j = 0; j < 4; j++)
                                                        {       temp[count][j] = order[count][j];}
                                                                // temp opt sequence list
                                                }
                                        }

                                        totTardy = 0;
                                        length = origlength;
                                        copyjoblist2();
                                }
                        }

// defining good partial sequence to use for next iteration
                        if (minSequence!=list)
                        {
                                list = 0;
                                length = optlength;
                                for (count = 0; count<length; count++)
                                {       for (j = 0; j < 4; j++)
                                        {       sequence[count][j] = temp[count][j];
                                                order[count][j] = sequence[count][j];}          // temp opt sequence list
                                }
                        }
                        }
}

        endScheduleTime = clock();
        scheduleTime = double (endScheduleTime - beginScheduleTime)/CLOCKS_PER_SEC;

// Calculate Completion Date
        beginTardyTime = clock();
        tardiness();
        endTardyTime = clock();
        tardyTime = double (endTardyTime - beginTardyTime)/CLOCKS_PER_SEC;

// Writing schedule to output file.

        write();

        inimprovschd.close();                                   // close input and output files
        outschedule.close();
        outtardy.close();
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void shiftup()                                                  // shifts the schedule up a unit
{
        for (count = index; count < length ; count++)
```

```
		{
			order[count][0] = order[count+1][0];
			order[count][1] = order[count+1][1];
			order[count][2] = order[count+1][2];
			order[count][3] = order[count+1][3];
			order[count][4] = order[count+1][4];
		}
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void shift()                                                    // shifts job list down
{
	for (count = length - 1; count >= index; count--)
		{
			order[count+1][0] = order[count][0];
			order[count+1][1] = order[count][1];
			order[count+1][2] = order[count][2];
			order[count+1][3] = order[count][3];
		}
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void copyjoblist()                                              // copy job list from order[][] to sequence[][]
{
	for (index = 0; index<length; index++)                  // refresh partial sequence list
	{for (j = 0; j < 4; j++)
		order[index][j] = temp[index][j];
	}
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void copyjoblist2()                                             // copy job list from order[][] to sequence[][]
{
	for (index = 0; index<length; index++)                  // refresh partial sequence list
	{for (j = 0; j < 4; j++)
		order[index][j] = sequence[index][j];
	}
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void changeover()                                               // add changeover 'jobs' to the schedule
{
	shift();
	order[index][0] = 0;
	order[index][2] = 1000;
	order[index][3] = 4./5.;

	if(order[index+1][1]==1 || order[index+1][1]==2)
```

```
                order[index][1] = 4;
        else
                order[index][1] = 5;
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void tardiness()                // determines the tardiness of a job based on 6.5 working hours/shift,
                                //                 3 shifts per day, 7 days per week
{
        const double    P1Change = 195./60,             // changeover time (shift-hr) for M14
                        P3Change = 243.75/60,           // changeover time (shift-hr) for RP910
                        P1 = 128.25/60,                 // process time / batch of M865
                        P2 = 152.02/60,                 // process time / batch of M831
                        P3 = 196.52/60,                 // process time / batch of RP910
                        dehy = 15.38/60;                // dehy process time / batch
        double          process = 0.;                   // total process time for all jobs

        void tardy(double process);

        for (count = 0; count < length; count++)
        {
                lateTime[count][0] = order[count][0];
                lateTime[count][1] = order[count][1];

                if (order[count][1]==4)
                {
                        process = process + P1Change;
                        lateTime[count][2] = process;
                        lateTime[count][3] = 0;
                }
                else if (order[count][1]==5)
                {
                        process = process + P3Change;
                        lateTime[count][2] = process;
                        lateTime[count][3] = 0;
                }
                else if (order[count][1]==1)
                {
                        if (order[count-1][1]>3)
                                process = process + (P1+dehy*(order[count][3]-1));
                        else
                                process = process + (dehy*order[count][3]);
                        lateTime[count][2] = process;
                        tardy(process);
                }
                else if (order[count][1]==2)
                {
                        if (order[count-1][1]>3)
                                process = process + (P2+dehy*(order[count][3]-1));
                        else
                                process = process + (dehy*order[count][3]);
                        lateTime[count][2] = process;
                        tardy(process);
                }
```

```
		else if (order[count][1]==3)
		{
			if (order[count-1][1]>3)
				process = process + (P3+dehy*(order[count][3]-1));
			else
				process = process + (dehy*order[count][3]);
			lateTime[count][2] = process;
			tardy(process);
		}
	}
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void tardy(double process)
{
	double	late;
//				hrPerWeek = 136.5;				// total working hours per week

	late = process - order[count][2];

	if (late < 0)
		lateTime[count][3] = 0;
	else
		lateTime[count][3] = late;

	totTardy = totTardy + lateTime[count][3];
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void write()							// write schedule to file
{
	for (count = 0; count<length; count++)
	{
		for (index = 0; index < 5.*sequence[count][3]; index++)
			outschedule << sequence[count][0] << " "<< sequence[count][1] << " "
				<< sequence[count][2] << " "<< sequence[count][3] << endl;
	}
	for (count = 0; count< length; count++)
	{
		for (index = 0; index < 4; index++)
		{
			outtardy << lateTime[count][index] << "  ";
			cout << lateTime[count][index] << "  ";
		}
		outtardy << endl;
		cout << endl;
	}
	cout << "Total Tardiness (Hours): " << totTardy << endl;
	cout << "CPU time to develop schedule: " <<
		setw(10) << setprecision(8) << scheduleTime << endl;
	cout << "CPU time to determine total tardy hours: " << tardyTime << endl;
}
```

APPENDIX 8: ALGORITHM TO DETERMINE OPTIMAL SCHEDULE (7 ORDERS) CODE

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//      optimal.cpp                     Optimizing Algorithm                                                                  //
//                                                                                                                            //
//      Progammer:              Niem-Trung L. Tra                                                                             //
//      Compiler:               Microsoft Visual C++ ver 5.0                                                                  //
//      Platform:               Pentium II 350, Windows 98                                                                    //
//      Date:                   January 2000                                                                                  //
//                                                                                                                            //
//      This algorithm optimizes any given schedule.                                                                          //
//                                                                                                                            //
//      First, the program reads an external file named "orders.txt" consisting of:                                           //
//              Customer Number                                                                                               //
//              Product Type                                                                                                  //
//              Due Date                                                                                                      //
//              Customer Order Quantity  (in batches)                                                                         //
//      Next, all the permutations of the the customer orders are developed and evaluated for total                           //
//      tardiness.  The schedule with the minimum tardiness is saved as the optimal schedule.                                //
//      Finally, the optimal production schedule is saved to a file named "schedule.txt".                                     //
//                                                                                                                            //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

#include <fstream.h>                    // for file I/O
#include <iomanip.h>                    // allows use of setw and setprecision
#include <stdlib.h>                     // allows use of rand()
#include <time.h>                       // allows use of clock to determine CPU time

        int             index = 0,      // position where item belongs
                        j,
                        i,              // loop 1
                        k,              // loop 2
                        l,              // loop 3
                        w,              // loop 4
                        x,              // loop 5
                        y,              // loop 6
                        z,              // loop 7
                        list=0,         // partial list length
                        count,          // loop control variable
                        end,
                        length = 0,     // length of ordered list
                        change = 0,     // number of changeovers
                        custNumber,     // customer number
                        prodType,       // product type ordered
                        dueDate = 0,    // order due date
                        quantity;       // quantity customer ordered in batches

        double  totTardy = 0,           // total hours of tardy jobs
                        optimal = 100000,       // minimum tardiness
                        scheduleTime,
                        tardyTime,
                        lateTime[100][4],       // customer #, type, time in weeks, tardy
                        schedule[100][4],       // optimal schedule
                        temp[100][4],           // working schedule
```

```
                order[100][4];                  // custNumber, prodType, dueDate, quantity
                                                //          working schedule w/ CO

        clock_t     beginScheduleTime,          // CPU time to find schedule
                    endScheduleTime,
                    beginTardyTime,             // CPU time to find total tardiness time
                    endTardyTime;

        ifstream inorders;                      // input file with customer order information
        ofstream outschedule;                   // output file for production schedule
        ofstream outtardy;                      //
        ofstream outimprovschd;                 //

void main()
{
        beginScheduleTime = clock();
        void reset();                           // reset the order working list
        void shift();                           // subroutine to shift job list down
        void changeover();                      // subroutine to add changeover "jobs"
        void write();                           // subroutine to write schedule to file
        void tardiness();                       // subroutine to calculate tardiness

        inorders.open("book2.txt");             // open input file
        if (!inorders)
        {
                cout << "** Can't open input file **" << endl;
                return;
        }

        outschedule.open("schedule.txt");       // open output file
        if (!outschedule)
        {
                cout << "** Can't open output file**" << endl;
                return;
        }

        outschedule.setf(ios::fixed, ios::floatfield);  // turn on manipulators

        outtardy.open("tardy.txt");             // open output file
        if (!outtardy)
        {
                cout << "** Can't open output file**" << endl;
                return;
        }

        outtardy.setf(ios::fixed, ios::floatfield);     // turn on manipulators

        outimprovschd.open("improvschd.txt");   // open output file
        if (!outimprovschd)
        {
                cout << "** Can't open output file**" << endl;
                return;
        }

        outimprovschd.setf(ios::fixed, ios::floatfield);  // turn on manipulators
```

```
// Read in input file and puts it in a list.
// There is an assumption made in this program that the data file has
// been setup correctly to input the correct data types.

        inorders >> custNumber >> prodType >> dueDate >> quantity;
        while (dueDate!=-999)
        {
                temp[length][0] = custNumber;
                temp[length][1] = prodType;
                temp[length][2] = dueDate;
                temp[length][3] = quantity;
                length++;
                end = length;
                inorders >> custNumber >> prodType >> dueDate >> quantity;
        }

        for (i=1; i<=7; i++)
        {       for (k=1; k<=7; k++)
                {       for (l=1; l<=7; l++)
                        {       for (w=1; w<=7; w++)
                                {       for (x=1; x<=7; x++)
                                        {       for (y=1; y<=7; y++)
                                                {       for (z=1; z<=7; z++)
                                                {if (z!=y && z!=x && z!=w && z!=l && z!=k && z!=i &&
                                                y!=z && y!=x && y!=w && y!=l && y!=k && y!=i &&
                                                x!=y && x!=z && x!=w && x!=l && x!=k && x!=i &&
                                                w!=y && w!=x && w!=z && w!=l && w!=k && w!=i &&
                                                l!=y && l!=x && l!=w && l!=z && l!=k && l!=i &&
                                                k!=y && k!=x && k!=w && k!=l && k!=z && k!=i &&
                                                i!=y && i!=x && i!=w && i!=l && i!=k && i!=z)
                                                {
                                                        order[0][0]=temp[i-1][0];
                                                        order[0][1]=temp[i-1][1];
                                                        order[0][2]=temp[i-1][2];
                                                        order[0][3]=temp[i-1][3];
                                                        order[1][0]=temp[k-1][0];
                                                        order[1][1]=temp[k-1][1];
                                                        order[1][2]=temp[k-1][2];
                                                        order[1][3]=temp[k-1][3];
                                                        order[2][0]=temp[l-1][0];
                                                        order[2][1]=temp[l-1][1];
                                                        order[2][2]=temp[l-1][2];
                                                        order[2][3]=temp[l-1][3];
                                                        order[3][0]=temp[w-1][0];
                                                        order[3][1]=temp[w-1][1];
                                                        order[3][2]=temp[w-1][2];
                                                        order[3][3]=temp[w-1][3];
                                                        order[4][0]=temp[x-1][0];
                                                        order[4][1]=temp[x-1][1];
                                                        order[4][2]=temp[x-1][2];
                                                        order[4][3]=temp[x-1][3];
                                                        order[5][0]=temp[y-1][0];
                                                        order[5][1]=temp[y-1][1];
                                                        order[5][2]=temp[y-1][2];
                                                        order[5][3]=temp[y-1][3];
                                                        order[6][0]=temp[z-1][0];
```

```
                                                order[6][1]=temp[z-1][1];
                                                order[6][2]=temp[z-1][2];
                                                order[6][3]=temp[z-1][3];
                                                length = 7;
                        // Add changeovers
                                                index = 0;
                                                changeover();
                                                length++;

                                                for (index = 1; index < length; index++)
                                                {if (order[index-1][1]!=order[index][1] &&
                                                        order[index-1][1]!=4 && order[index-
                                                        1][1]!=5)
                                                        {
                                                                changeover();
                                                                length++;
                                                        }
                                                }
                                                totTardy = 0;
                                                tardiness();

                                                if (totTardy < optimal)
                                                {
                                                        optimal = totTardy;
                                                        for (index = 0; index < length; index++)
                                                        {
                                                        schedule[index][0] = order[index][0];
                                                        schedule[index][1] = order[index][1];
                                                        schedule[index][2] = order[index][2];
                                                        schedule[index][3] = order[index][3];
                                                        }
                                                        end = length;
                                                }
                                                }
                                                }
                                        }
                                }
                        }
                }
        }

        length = end;
        for (index = 0; index < length; index++)
        {
                order[index][0] = schedule[index][0];
                order[index][1] = schedule[index][1];
                order[index][2] = schedule[index][2];
                order[index][3] = schedule[index][3];
        }

        endScheduleTime = clock();
        scheduleTime = double (endScheduleTime - beginScheduleTime)/CLOCKS_PER_SEC;

// Calculate Completion Date
        beginTardyTime = clock();
```

```
	totTardy = 0;
	tardiness();
	endTardyTime = clock();
	tardyTime = double (endTardyTime - beginTardyTime)/CLOCKS_PER_SEC;

// Writing schedule to output file.

	write();

	inorders.close();                                   // close input and output files
	outschedule.close();
	outtardy.close();
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void shift()                                            // shifts job list down
{
	for (count = length - 1; count >= index; count--)
		{
			order[count+1][0] = order[count][0];
			order[count+1][1] = order[count][1];
			order[count+1][2] = order[count][2];
			order[count+1][3] = order[count][3];
		}
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void changeover()                                       // add changeover 'jobs' to the schedule
{

// Insert first changeover for partial job list
	shift();

	order[index][0] = 0;
	order[index][2] = 1000;
	order[index][3] = 4./5.;

	if(order[index+1][1]==1 || order[index+1][1]==2)
		order[index][1] = 4;
	else
		order[index][1] = 5;
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void tardiness()                    // determines the tardiness of a job based on 6.5 working hours/shift,
                                    //                  3 shifts per day, 7 days per week
{
	const double	T1 = 195./60,       // changeover time (shift-hr) for Type 1
			T3 = 243.75/60,     // changeover time (shift-hr) for Type 3
			P1 = 128.25/60,     // process time / batch of Product 1
```

```
                P2 = 152.02/60,         // process time / batch of Product 2
                P3 = 196.52/60,         // process time / batch of Product 3
                cutter = 15.38/60;      // dehy process time / batch
        double  process = 0.;           // total process time for all jobs

        void tardy(double process);

        for (count = 0; count < length; count++)
        {
                lateTime[count][0] = order[count][0];
                lateTime[count][1] = order[count][1];

                if (order[count][1]==4)
                {
                        process = process + m14Change;
                        lateTime[count][2] = process;
                        lateTime[count][3] = 0;
                }
                else if (order[count][1]==5)
                {
                        process = process + rpChange;
                        lateTime[count][2] = process;
                        lateTime[count][3] = 0;
                }
                else if (order[count][1]==1)
                {
                        if (order[count-1][1]>3)
                                process = process + (mix865+cutter*(order[count][3]-1));
                        else
                                process = process + (cutter*order[count][3]);
                        lateTime[count][2] = process;
                        tardy(process);
                }
                else if (order[count][1]==2)
                {
                        if (order[count-1][1]>3)
                                process = process + (mix831+cutter*(order[count][3]-1));
                        else
                                process = process + (cutter*order[count][3]);
                        lateTime[count][2] = process;
                        tardy(process);
                }
                else if (order[count][1]==3)
                {
                        if (order[count-1][1]>3)
                                process = process + (mix910+cutter*(order[count][3]-1));
                        else
                                process = process + (cutter*order[count][3]);
                        lateTime[count][2] = process;
                        tardy(process);
                }
        }
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
void tardy(double process)
{
	double	late;

	late = process - order[count][2];
	if (late < 0)
		lateTime[count][3] = 0;
	else
		lateTime[count][3] = late;

	totTardy = totTardy + lateTime[count][3];
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void write()	// write schedule to file
{
	for (count = 0; count<length; count++)
	{
		for (index = 0; index < 5.*order[count][3]; index++)
			outschedule << order[count][0] << " "<< order[count][1] << " "
				<< order[count][2] << " "<< order[count][3] << endl;
	}

	for (count = 0; count< length; count++)
	{
		for (index = 0; index < 4; index++)
		{
			outimprovschd<<order[count][index]<<" ";
			outtardy << lateTime[count][index] << "  ";
			cout << lateTime[count][index] << "  ";
		}
		outimprovschd<<endl;
		outtardy <<order[count][2]<<endl;
		cout << endl;
	}
	outimprovschd<<0<<" "<<0<<" "<<0<<" "<<-999;
	cout << "Total Tardiness (Hours): " << totTardy << endl;
	cout << "CPU time to develop schedule: " <<
		setw(10) << setprecision(8) << scheduleTime << endl;
	cout << "CPU time to determine total tardy hours: " << tardyTime << endl;
}
```

APPENDIX 9: SIMULATIONS CODE

```
********************************************************************************
*                                                                              *
*                        Formatted Listing of Model:                           *
*                  C:\My Documents\THESIS\Research\final.MOD                   *
*                                                                              *
********************************************************************************

  Time Units:                       Minutes
  Distance Units:                   Feet


********************************************************************************
*                                 Locations                                    *
********************************************************************************

  Name                    Cap      Units Stats       Rules                 Cost
  ---------------------- -------- ----- ----------- --------------------- ------------
  Dehy_Press1a            1        1     Time Series Oldest, ,
  Dehy_Press1b            1        1     Time Series Oldest, ,
  Dehy_Press2a            1        1     Time Series Oldest, ,
  Dehy_Press2b            1        1     Time Series Oldest, ,
  Conveyor1a              1        1     Time Series Oldest, ,
  Conveyor1b              1        1     Time Series Oldest, ,
  Conveyor2a              1        1     Time Series Oldest, ,
  Conveyor2b              1        1     Time Series Oldest, ,
  Scale1                  1        1     Time Series Oldest, ,
  Scale2                  1        1     Time Series Oldest, ,
  Dehy_Trans1a            5        1     Time Series Oldest, ,
  Dehy_Trans1b            5        1     Time Series Oldest, ,
  Dehy_Trans2a            5        1     Time Series Oldest, ,
  Dehy_Trans2b            5        1     Time Series Oldest, ,
  Dehy_T1a                1        1     Time Series Oldest, ,
  Dehy_T1b                1        1     Time Series Oldest, ,
  Dehy_T2a                1        1     Time Series Oldest, ,
  Dehy_T2b                1        1     Time Series Oldest, ,
  Tub1                    10       2     Time Series Oldest, FIFO, By turn
  Tub1.1                  5        1     Time Series Oldest, FIFO,
  Tub1.2                  5        1     Time Series Oldest, FIFO,
  Tub2                    10       2     Time Series Oldest, FIFO, By turn
  Tub2.1                  5        1     Time Series Oldest, FIFO,
  Tub2.2                  5        1     Time Series Oldest, FIFO,
  Empty_buggies1          1        2     Time Series Oldest, , First
  Empty_buggies1.1        1        1     Time Series Oldest, ,
  Empty_buggies1.2        1        1     Time Series Oldest, ,
  Empty_buggies2          1        2     Time Series Oldest, , First
  Empty_buggies2.1        1        1     Time Series Oldest, ,
  Empty_buggies2.2        1        1     Time Series Oldest, ,
  Empty_buggies           1        17    Time Series Oldest, , First
  Empty_buggies.1         1        1     Time Series Oldest, ,
  Empty_buggies.2         1        1     Time Series Oldest, ,
  Empty_buggies.3         1        1     Time Series Oldest, ,
  Empty_buggies.4         1        1     Time Series Oldest, ,
  Empty_buggies.5         1        1     Time Series Oldest, ,
  Empty_buggies.6         1        1     Time Series Oldest, ,
  Empty_buggies.7         1        1     Time Series Oldest, ,
  Empty_buggies.8         1        1     Time Series Oldest, ,
  Empty_buggies.9         1        1     Time Series Oldest, ,
  Empty_buggies.10        1        1     Time Series Oldest, ,
  Empty_buggies.11        1        1     Time Series Oldest, ,
  Empty_buggies.12        1        1     Time Series Oldest, ,
  Empty_buggies.13        1        1     Time Series Oldest, ,
  Empty_buggies.14        1        1     Time Series Oldest, ,
  Empty_buggies.15        1        1     Time Series Oldest, ,
  Empty_buggies.16        1        1     Time Series Oldest, ,
  Empty_buggies.17        1        1     Time Series Oldest, ,
  Gp_Empty_buggies        2        1     Time Series Oldest, ,
  Ungp_Empty_buggies1     2        1     Time Series Oldest, ,
  Ungp_Empty_buggies2     2        1     Time Series Oldest, ,
  MixerQ                  11       1     Time Series Oldest, FIFO, First
  Mixer1a                 1        1     Time Series Oldest, ,
  Mixer2a                 1        1     Time Series Oldest, ,
  Mixer3a                 1        1     Time Series Oldest, ,
  Mixer1b                 1        1     Time Series Oldest, ,
  Mixer2b                 1        1     Time Series Oldest, ,
  Mixer3b                 1        1     Time Series Oldest, ,
  Mixer_Load1             5        1     Time Series Oldest, ,
  Mixer_Load2             5        1     Time Series Oldest, ,
  Mixer_Load3             5        1     Time Series Oldest, ,
  Mixer_Unload1           1        1     Time Series Oldest, ,
  Mixer_Unload2           1        1     Time Series Oldest, ,
  Mixer_Unload3           1        1     Time Series Oldest, ,
  Mixer_Trans1            1        2     Time Series Oldest, , First
  Mixer_Trans1.1          1        1     Time Series Oldest, ,
  Mixer_Trans1.2          1        1     Time Series Oldest, ,
  BlockerQ                5        1     Time Series Oldest, FIFO,
  Blocker_Control_Center  1        1     Time Series Oldest, ,
  Blocker1                1        1     Time Series Oldest, ,
  Blocker2                1        1     Time Series Oldest, ,
  Blocker_Trans           1        2     Time Series Oldest, , First
  Blocker_Trans.1         1        1     Time Series Oldest, ,
  Blocker_Trans.2         1        1     Time Series Oldest, ,
```

Empty_Block_Bug	1	2	Time Series	Oldest, , First
Empty_Block_Bug.1	1	1	Time Series	Oldest, ,
Empty_Block_Bug.2	1	1	Time Series	Oldest, ,
Buggy_Location	6	1	Time Series	Oldest, ,
Elevator	6	1	Time Series	Oldest, ,
UExt2_Unload_bug	6	1	Time Series	Oldest, ,
UExt2_combine	6	1	Time Series	Oldest, ,
UExt2	1	1	Time Series	Oldest, ,
UExt3_Unload_bug	2	1	Time Series	Oldest, ,
UExt3_combine	2	1	Time Series	Oldest, ,
UExt3	1	1	Time Series	Oldest, ,
UExt4_Unload_bug	2	1	Time Series	Oldest, ,
UExt4_combine	2	1	Time Series	Oldest, ,
UExt4	1	1	Time Series	Oldest, ,
DExt2	1	1	Time Series	Oldest, ,
DExt3	1	1	Time Series	Oldest, ,
DExt4	1	1	Time Series	Oldest, ,
Hen_Nest	1	6	Time Series	Oldest, , First
Hen_Nest.1	1	1	Time Series	Oldest, ,
Hen_Nest.2	1	1	Time Series	Oldest, ,
Hen_Nest.3	1	1	Time Series	Oldest, ,
Hen_Nest.4	1	1	Time Series	Oldest, ,
Hen_Nest.5	1	1	Time Series	Oldest, ,
Hen_Nest.6	1	1	Time Series	Oldest, ,
NIR_SAMPLE_WAIT1	2	1	Time Series	Oldest, ,
NIR_SAMPLE_WAIT2	2	1	Time Series	Oldest, ,
NIR_Pack	10	1	Time Series	Oldest, ,
NIR_Test	2	1	Time Series	Oldest, ,
NIR_Water	1	1	Time Series	Oldest, ,
Buggy_Wait1	1	2	Time Series	Oldest, , First
Buggy_Wait1.1	1	1	Time Series	Oldest, ,
Buggy_Wait1.2	1	1	Time Series	Oldest, ,
Buggy_Wait2	1	2	Time Series	Oldest, , First
Buggy_Wait2.1	1	1	Time Series	Oldest, ,
Buggy_Wait2.2	1	1	Time Series	Oldest, ,
Empty_Nests	6	1	Time Series	Oldest, ,
CutterQ	20	1	Time Series	Oldest, ,
Cutter1	1	1	Time Series	Oldest, ,
Cutter3	1	1	Time Series	Oldest, ,
Cutter4	1	1	Time Series	Oldest, ,
Cutter7	1	1	Time Series	Oldest, ,
Can_Flat_Loc1	1	1	Time Series	Oldest, , First
Can_Flat_Loc2	1	1	Time Series	Oldest, , First
Can_Flat_Loc3	1	1	Time Series	Oldest, ,
Can_Flat_Loc4	1	1	Time Series	Oldest, ,
Can_Flat1	1	1	Time Series	Oldest, ,
Can_Flat3	1	1	Time Series	Oldest, ,
Can_Flat4	1	1	Time Series	Oldest, ,
Can_Flat7	1	1	Time Series	Oldest, ,
Empty_Can_Flats	4	1	Time Series	Oldest, ,
SPC_Data_Entry1	1	1	Time Series	Oldest, ,
SPC_Data_Entry2	1	1	Time Series	Oldest, ,
Badger_Buggy1	1	1	Time Series	Oldest, ,
Badger_Buggy2	1	1	Time Series	Oldest, ,
Badger_Buggy_Scale	1	1	Time Series	Oldest, ,
Loading_Dock	6	1	Time Series	Oldest, , First
Empty_Mixer_Flat	1	5	Time Series	Oldest, , First
Empty_Mixer_Flat.1	1	1	Time Series	Oldest, ,
Empty_Mixer_Flat.2	1	1	Time Series	Oldest, ,
Empty_Mixer_Flat.3	1	1	Time Series	Oldest, ,
Empty_Mixer_Flat.4	1	1	Time Series	Oldest, ,
Empty_Mixer_Flat.5	1	1	Time Series	Oldest, ,
Empty_Badger_Buggy	1	8	Time Series	Oldest, , First
Empty_Badger_Buggy.1	1	1	Time Series	Oldest, ,
Empty_Badger_Buggy.2	1	1	Time Series	Oldest, ,
Empty_Badger_Buggy.3	1	1	Time Series	Oldest, ,
Empty_Badger_Buggy.4	1	1	Time Series	Oldest, ,
Empty_Badger_Buggy.5	1	1	Time Series	Oldest, ,
Empty_Badger_Buggy.6	1	1	Time Series	Oldest, ,
Empty_Badger_Buggy.7	1	1	Time Series	Oldest, ,
Empty_Badger_Buggy.8	1	1	Time Series	Oldest, ,
Jobs	INFINITE	1	Time Series	Oldest, FIFO, First
Job2	1	1	Time Series	Oldest, FIFO,
Changeover_Buggy	3	1	Time Series	Oldest, ,
Changeover_Buggy2	3	1	Time Series	Oldest, ,
Changeover_Buggy3	1	1	Time Series	Oldest, ,
Changeover_Buggy4	1	1	Time Series	Oldest, ,

****
* Entities *
****

Name	Speed (fpm)	Stats	Cost
EntA	1	Time Series	
Block	1	Time Series	
Dehy_Buggy	1	Time Series	
Two_Dehy_Buggies	1	Time Series	
NIR_Sample	1	Time Series	
Mixer_Flat	1	Time Series	
Mix	1	Time Series	
Block2	1	Time Series	
Chords	1	Time Series	
Hens_Nest	1	Time Series	

Can_Flat	1	Time Series
Blocker_Buggy	1	Time Series
Propellent	1	Time Series
Badger_Buggy	1	Time Series
SPC_Paperwork	1	Time Series

Path Networks

Name	Type	T/S	From	To	BI	Dist/Time	Speed Factor
NIR	Passing	Speed & Distance	N4	N5	Bi	.001	1
			N4	N6	Bi	.001	1
			N4	N9	Bi	.001	1
			N3	N10	Bi	.001	1
			N4	N3	Bi	M_W_NIR	1
Job	Passing	Speed & Distance	N1	N2	Bi	0	1
			N1	N3	Bi	0	1
Dehy1	Passing	Speed & Distance	N1	N2	Bi	M_PL_dehy/4	1
			N1	N3	Bi	M_PL_dehy/4	1
			N2	N4	Bi	.001	1
			N4	N5	Bi	.001	1
			N3	N6	Bi	.001	1
			N6	N5	Bi	.001	1
			N5	N7	Bi	.001	1
			N5	N8	Bi	.001	1
			N4	N9	Bi	.001	1
			N7	N9	Bi	.001	1
			N7	N10	Bi	.001	1
			N8	N10	Bi	.001	1
			N10	N12	Bi	M_T_NIR/2	1
			N11	N13	Bi	.001	1
			N10	N14	Bi	.001	1
			N14	N11	Bi	M_T_dehy/2	1
Dehy2	Passing	Speed & Distance	N1	N2	Bi	M_PL_dehy/4	1
			N2	N3	Bi	.001	1
			N3	N4	Bi	.001	1
			N4	N5	Bi	.001	1
			N5	N6	Bi	.001	1
			N1	N7	Bi	M_PL_dehy/4	1
			N7	N8	Bi	.001	1
			N8	N4	Bi	.001	1
			N4	N9	Bi	.001	1
			N9	N6	Bi	.001	1
			N6	N10	Bi	M_T_NIR/2	1
			N6	N11	Bi	.001	1
			N11	N12	Bi	M_T_dehy/2	1
			N12	N13	Bi	.001	1
			N9	N14	Bi	.001	1
			N14	N6	Bi	.001	1
Mixer	Passing	Speed & Distance	N2	N3	Bi	.001	1
			N2	N5	Bi	.001	1
			N5	N7	Bi	.001	1
			N1	N6	Bi	M_L_mixer/6	1
			N6	N5	Bi	M_L_mixer/6	1
			N6	N8	Bi	.001	1
			N5	N9	Bi	.001	1
			N9	N10	Bi	.001	1
			N10	N11	Bi	.001	1
			N9	N12	Bi	.001	1
			N12	N13	Bi	.001	1
			N13	N14	Bi	.001	1
Mixer_Trans	Passing	Speed & Distance	N1	N3	Bi	.001	1
			N2	N4	Bi	.001	1
			N4	N1	Uni	M_T_mixer/2	1
			N1	N2	Uni	M_T_mixer/2	1
Blocker	Passing	Speed & Distance	N1	N2	Bi	.001	1
			N2	N3	Bi	.001	1
			N2	N4	Bi	.001	1
			N4	N5	Bi	M_Position_blocker	1
			N4	N6	Bi	.001	1
			N5	N7	Bi	.001	1
			N5	N9	Bi	.001	1
			N5	N10	Bi	.001	1
			N7	N4	Bi	.001	1
			N4	N8	Bi	.001	1
			N8	N12	Bi	M_T_blocker/2	1
			N12	N13	Bi	.001	1
			N12	N14	Bi	.001	1
			N13	N14	Bi	.001	1
Up_Extruder	Passing	Speed & Distance	N2	N3	Bi	.001	1
			N2	N4	Bi	.001	1
			N2	N5	Bi	.001	1
			N3	N6	Bi	.001	1
			N3	N7	Bi	.001	1
			N4	N8	Bi	.001	1
			N6	N9	Bi	.001	1
			N2	N10	Bi	.001	1
			N3	N11	Bi	.001	1
			N11	N12	Bi	.001	1
			N11	N13	Bi	.001	1
			N1	N15	Bi	.001	1
			N15	N2	Bi	.001	1

			N4	N19	Bi	.001	1
			N6	N20	Bi	.001	1
			N12	N21	Bi	.001	1
			N11	N14	Bi	2.14	1
Down_Extruder	Passing	Speed & Distance	N1	N2	Bi	.001	1
			N4	N5	Bi	.001	1
			N5	N2	Bi	.001	1
			N5	N8	Bi	.001	1
			N5	N9	Bi	.001	1
			N9	N10	Bi	.001	1
			N2	N7	Bi	M_T_Dext/3	1
			N7	N15	Bi	M_T_Dext*2/3	1
			N15	N16	Bi	.001	1
Cutter	Passing	Speed & Distance	N1	N2	Bi	.001	1
			N2	N3	Bi	.001	1
			N2	N5	Bi	.001	1
			N2	N6	Bi	.001	1
			N2	N8	Bi	.001	1
			N3	N10	Bi	.001	1
			N3	N14	Bi	M_T_cutter	1
			N14	N15	Bi	.001	1
			N14	N16	Bi	.001	1
			N16	N17	Bi	.001	1
			N16	N18	Bi	.001	1
			N16	N19	Bi	.001	1
			N19	N21	Bi	.001	1
			N1	N22	Bi	.001	1
			N16	N23	Bi	.001	1
			N16	N24	Bi	.001	1
			N16	N25	Bi	.001	1

****
Interfaces
****

Net	Node	Location
NIR	N5	NIR_Test
	N3	NIR_Pack
	N10	NIR_Water
Job	N3	Tub2
	N2	Tub1
	N1	Jobs
Dehy1	N2	Dehy_Press1a
	N5	Scale1
	N3	Dehy_Press1b
	N11	MixerQ
	N4	Conveyor1a
	N6	Conveyor1b
	N1	Tub1
	N12	NIR_Pack
	N10	Buggy_Wait1
	N13	Empty_buggies
	N14	Empty_buggies1
	N8	Dehy_Trans1b
	N8	Dehy_T1b
	N7	Dehy_Trans1a
	N7	Dehy_T1a
	N10	NIR_SAMPLE_WAIT1
	N13	Gp_Empty_buggies
	N14	Ungp_Empty_buggies1
Dehy2	N13	Empty_buggies
	N12	MixerQ
	N6	NIR_SAMPLE_WAIT2
	N11	Empty_buggies2
	N9	Dehy_T2a
	N9	Dehy_Trans2a
	N5	Dehy_T2b
	N5	Dehy_Trans2b
	N4	Scale2
	N3	Conveyor2b
	N8	Conveyor2a
	N7	Dehy_Press2a
	N2	Dehy_Press2b
	N1	Tub2
	N10	NIR_Pack
	N6	Buggy_Wait2
	N13	Gp_Empty_buggies
	N11	Ungp_Empty_buggies2
Mixer	N2	Mixer_Load1
	N3	Mixer1a
	N3	Mixer1b
	N5	Mixer_Trans1
	N8	Empty_buggies
	N1	MixerQ
	N3	Mixer_Unload1
	N10	Mixer_Load2
	N11	Mixer2b
	N11	Mixer_Unload2
	N11	Mixer2a
	N13	Mixer_Load3
	N14	Mixer_Unload3
	N14	Mixer3b
	N14	Mixer3a

```
  Mixer_Trans   N1         Mixer_Trans1
                N2         BlockerQ
                N4         Empty_Mixer_Flat
  Blocker       N7         Blocker_Trans
                N3         Blocker_Control_Center
                N1         BlockerQ
                N5         Buggy_Location
                N9         Blocker1
                N10        Blocker2
                N8         Empty_Mixer_Flat
                N13        Elevator
                N14        Empty_Block_Bug
  Up_Extruder   N4         UExt2_Unload_bug
                N6         UExt3_Unload_bug
                N20        UExt3
                N8         DExt2
                N9         DExt3
                N12        UExt4_Unload_bug
                N21        UExt4
                N1         Elevator
                N19        UExt2
                N14        DExt4
  Down_Extruder N1         DExt2
                N4         DExt3
                N10        DExt4
                N15        CutterQ
                N15        Empty_Nests
                N7         Hen_Nest
  Cutter        N1         CutterQ
                N5         Cutter1
                N6         Cutter3
                N8         Cutter4
                N15        SPC_Data_Entry1
                N21        Loading_Dock
                N10        Cutter7
                N19        Badger_Buggy_Scale
                N1         Empty_Nests
                N17        Can_Flat_Loc1
                N18        Can_Flat_Loc2
                N6         Can_Flat3
                N5         Can_Flat1
                N8         Can_Flat4
                N10        Can_Flat7
                N17        Badger_Buggy1
                N18        Badger_Buggy2
                N23        Can_Flat_Loc3
                N24        Can_Flat_Loc4
                N25        Empty_Badger_Buggy
                N14        Empty_Can_Flats

*****************************************************************************
*                               Resources                                   *
*****************************************************************************

                                  Res     Ent
  Name              Units Stats   Search  Search Path          Motion       Cost
  ----------------- ----- -------- ------- ------ ------------ ------------ ------------
  NIR_Tester        1     By Unit  Closest Oldest NIR          Empty: 1 fpm
                                                  Home: N9     Full: 1 fpm
                                                  (Return)

  Dehy1_Operator    1     By Unit  Closest Oldest Dehy1        Empty: 1 fpm
                                                  Home: N9     Full: 1 fpm
                                                  (Return)

  Dehy2_Operator    1     By Unit  Closest Oldest Dehy2        Empty: 1 fpm
                                                  Home: N14    Full: 1 fpm
                                                  (Return)

  Mixer_Operator    4     By Unit  Closest Oldest Mixer        Empty: 1 fpm
                                                  Home: N7     Full: 1 fpm
                                                  (Return)

  Mixer_Tranporter  1     By Unit  Closest Oldest Mixer_Trans  Empty: 1 fpm
                                                  Home: N3     Full: 1 fpm

  Blocker_Operator1 1     By Unit  Closest Oldest Blocker      Empty: 1 fpm
                                                  Home: N2     Full: 1 fpm
                                                  (Return)

  Blocker_Operator2 1     By Unit  Closest Oldest Blocker      Empty: 1 fpm
                                                  Home: N2     Full: 1 fpm
                                                  (Return)

  UEXtru_Operator   2     By Unit  Closest Oldest Up_Extruder  Empty: 1 fpm
                                                  Home: N10    Full: 1 fpm
                                                  (Return)

  DEXtru_Operator   3     By Unit  Closest Oldest Down_Extruder Empty: 1 fpm
                                                  Home: N8     Full: 1 fpm
                                                  (Return)

  Cutter_Operator   5     By Unit  Closest Oldest Cutter       Empty: 1 fpm
                                                  Home: N22    Full: 1 fpm
```

```
*****************************************************************************
*                              Processing                                   *
*****************************************************************************

                                        Process                                  Routing

 Entity          Location           Operation            Blk  Output          Destination         Rule                        Move Logic
 --------------- ------------------ ------------------   ---- --------------- ------------------- --------------------------  ------------
 Block           Jobs                                    1    Block           Job2                FIRST 1
 Block           Job2               IF A_Mix_Type > 3 THEN
                                    {
                                    Graphic 10
                                    IF V_First_CO = 0 THEN
                                    {
                                    WAIT UNTIL V_Tub1_Space > 5 AND V_Tub2_Space > 5
                                    }
                                    ROUTE 1
                                    }
                                    ELSE
                                    {
                                    INC V_COUNT
                                    IF V_COUNT < 10 THEN
                                    { ROUTE 2   }
                                    ELSE
                                    { IF V_COUNT < 19  THEN
                                      {         ROUTE 3   }
                                      ELSE
                                      {          IF V_COUNT < 20 THEN
                                                 {         V_COUNT = -1
                                                           ROUTE 3   }
                                      }
                                    }
                                    }                   1    Block           Tub1                TURN 1                      DEC V_Tub1_Space
                                                                                                                             inc V_num_processed
                                                             Block           Tub2                TURN                        DEC V_Tub2_Space
                                                                                                                             inc V_num_processed
                                                        2    Block           Tub1                FIRST 1                     DEC V_Tub1_Space
                                                                                                                             inc V_num_processed
                                                        3    Block           Tub2                FIRST 1                     DEC V_Tub2_Space
                                                                                                                             inc V_num_processed
 Block           Tub1               IF A_Cust_Num<> V_Cust_Num AND
                                       A_Cust_Num<> V_Cust_Num2 AND
                                       A_Cust_Num<> V_Cust_Num3 AND
                                       A_Cust_Num<> V_Cust_Num4 AND
                                       A_Cust_Num<> V_Cust_Num5 THEN
                                    {
                                    IF V_Cust_Num=0 THEN
                                    {  V_Cust_Num= A_Cust_Num
                                      V_Tot_Mixes=A_Tot_Mixes }
                                    ELSE { IF V_Cust_Num2=0 THEN
                                      {          V_Cust_Num2= A_Cust_Num
                                                 V_Tot_Mixes2=A_Tot_Mixes }
                                      ELSE { IF V_Cust_Num3=0 THEN
                                                 {          V_Cust_Num3= A_Cust_Num
                                                            V_Tot_Mixes3=A_Tot_Mixes }
                                                 ELSE { IF V_Cust_Num4=0 THEN
                                                            {          V_Cust_Num4= A_Cust_Num
                                                                      V_Tot_Mixes4=A_Tot_Mixes }
                                                            ELSE { IF V_Cust_Num5=0 THEN
                                                                       {          V_Cust_Num5= A_Cust_Num
                                                                                 V_Tot_Mixes5=A_Tot_Mixes }}}}}
                                    }
```

```
                                        1    Block           Dehy_Press1a       FIRST 1                  MOVE WITH Dehy1_Operator,6  THEN FREE
                                                                                                         INC V_Tub1_Space
                                             Block           Dehy_Press1b       ALT                      MOVE WITH Dehy1_Operator,6  THEN FREE
                                                                                                         INC V_Tub1_Space
                                        2*   Dehy_Buggy      Changeover_Buggy   FIRST 1                  GRAPHIC 7
Block           Tub2                IF A_Cust_Num<> V_Cust_Num AND
                                       A_Cust_Num<> V_Cust_Num2 AND
                                       A_Cust_Num<> V_Cust_Num3 AND
                                       A_Cust_Num<> V_Cust_Num4 AND
                                       A_Cust_Num<> V_Cust_Num5 THEN
                                    {
                                    IF V_Cust_Num=0 THEN
                                    {  V_Cust_Num= A_Cust_Num
                                      V_Tot_Mixes=A_Tot_Mixes }
                                    ELSE { IF V_Cust_Num2=0 THEN
                                      {          V_Cust_Num2= A_Cust_Num
                                                 V_Tot_Mixes2=A_Tot_Mixes }
                                      ELSE { IF V_Cust_Num3=0 THEN
                                                 {          V_Cust_Num3= A_Cust_Num
                                                            V_Tot_Mixes3=A_Tot_Mixes }
                                                 ELSE { IF V_Cust_Num4=0 THEN
                                                            {          V_Cust_Num4= A_Cust_Num
                                                                      V_Tot_Mixes4=A_Tot_Mixes }
                                                            ELSE { IF V_Cust_Num5=0 THEN
                                                                       {          V_Cust_Num5= A_Cust_Num
                                                                                 V_Tot_Mixes5=A_Tot_Mixes }}}}}
                                    }                   1    Block           Dehy_Press2a       FIRST 1  MOVE WITH Dehy2_Operator,6  THEN FREE
                                                                                                         INC V_Tub2_Space
                                                        Block           Dehy_Press2b       ALT      MOVE WITH Dehy2_Operator,6 THEN FREE
                                                                                                         INC V_Tub2_Space
                                                        2*   Dehy_Buggy      Changeover_Buggy   FIRST 1  GRAPHIC 7
Dehy_Buggy      Changeover_Buggy    IF A_Mix_Type < 4 THEN
                                    {          ROUTE 2   }
                                    ELSE
                                    {
                                    INC V_NCO_loaded
                                    A_Mix_Type=4

                                    IF V_NCO_loaded  < 4THEN
                                    {
                                    IF V_First_CO = 0 THEN
                                    {
                                    WAIT 30 MIN
                                    }
                                    WAIT UNTIL V_MixerQ = 0
                                    ROUTE 1
                                    }
                                    ELSE
                                    {
                                     V_NCO_loaded=0
                                     ROUTE 2
                                    }}                  1    Dehy_Buggy      MixerQ             FIRST 1  INC V_MixerQ
                                                        2    Dehy_Buggy      EXIT               FIRST 1
Block           Dehy_Press1a        IF A_Mix_Type = 4 THEN
                                    {
                                    V_Ready_4_Changeover=V_Ready_4_Changeover+.25
                                     WAIT UNTIL V_Ready_4_Changeover = 5
                                    V_Process_Time = CLOCK(HR)
                                    WAIT M_Change_P1
                                    V_Ready_4_Changeover = 0
                                    ROUTE 2
                                    }
                                    ELSE
                                    { IF  A_Mix_Type = 5 THEN
                                      {
                                      V_Ready_4_Changeover=V_Ready_4_Changeover+.25
                                       WAIT UNTIL V_Ready_4_Changeover = 5
                                       V_Process_Time = CLOCK(HR)
```

```
                                WAIT M_Change_P3
                                V_Ready_4_Changeover = 0
                                ROUTE 2
                                }
                                ELSE
                                {
                                USE Dehy1_Operator, 8 FOR M_L_dehy
                                WAIT M_R_dehya

                                A_Type =  1
                                ROUTE 1
                                }
                               }                    1    Block            Scale1             FIRST 1                    MOVE WITH Dehy1_Operator,8 THEN FREE

                                                    2    Block            EXIT               FIRST 1
Block           Dehy_Press2a    IF A_Mix_Type = 4 THEN
                               {
                               V_Ready_4_Changeover=V_Ready_4_Changeover+.25
                                WAIT UNTIL V_Ready_4_Changeover = 5
                               V_Process_Time = CLOCK(HR)
                               WAIT M_Change_P1
                               V_Ready_4_Changeover = 0
                               ROUTE 2
                               }
                               ELSE
                               { IF  A_Mix_Type = 5 THEN
                                 {
                                 V_Ready_4_Changeover=V_Ready_4_Changeover+.25
                                  WAIT UNTIL V_Ready_4_Changeover = 5
                                 V_Process_Time = CLOCK(HR)
                                 WAIT M_Change_P3
                                 V_Ready_4_Changeover = 0
                                 ROUTE 2
                                 }
                                 ELSE
                                 {
                               USE Dehy2_Operator, 8 FOR M_L_dehy
                               WAIT M_R_dehya

                               A_Type =  1
                                 ROUTE 1
                               }}                   1    Block            Scale2             FIRST 1                    MOVE WITH Dehy2_Operator,8 THEN FREE

                                                    2    Block            EXIT               FIRST 1
Block           Dehy_Press1b    IF A_Mix_Type = 4 THEN
                               {
                               V_Ready_4_Changeover=V_Ready_4_Changeover+.25
                                WAIT UNTIL V_Ready_4_Changeover = 5
                               V_Process_Time = CLOCK(HR)
                               WAIT M_Change_P1
                               V_Ready_4_Changeover = 0
                               ROUTE 2
                               }
                               ELSE
                               { IF  A_Mix_Type = 5 THEN
                                 {
                                 V_Ready_4_Changeover=V_Ready_4_Changeover+.25
                                  WAIT UNTIL V_Ready_4_Changeover = 5
                                 V_Process_Time = CLOCK(HR)
                                 WAIT M_Change_P3
                                 V_Ready_4_Changeover = 0
                                 ROUTE 2
                                 }
                                 ELSE
                                 {
                               USE Dehy1_Operator, 8 FOR M_L_dehy
                               WAIT M_R_dehyb
```

```
                          A_Type = 2
                            ROUTE 1
                          }}                1    Block           Scale1             FIRST 1                   MOVE WITH Dehy1_Operator,8 THEN FREE

                                            2    Block           EXIT               FIRST 1
Block          Dehy_Press2b       IF A_Mix_Type = 4 THEN
                                  {
                                  V_Ready_4_Changeover=V_Ready_4_Changeover+.25
                                   WAIT UNTIL V_Ready_4_Changeover = 5
                                  V_Process_Time = CLOCK(HR)
                                  WAIT M_Change_P1
                                  V_Ready_4_Changeover = 0
                                  ROUTE 2
                                  }
                                  ELSE
                                  { IF  A_Mix_Type = 5 THEN
                                    {
                                    V_Ready_4_Changeover=V_Ready_4_Changeover+.25
                                     WAIT UNTIL V_Ready_4_Changeover = 5
                                    V_Process_Time = CLOCK(HR)
                                    WAIT M_Change_P3
                                    V_Ready_4_Changeover = 0
                                    ROUTE 2
                                    }
                                    ELSE
                                    {
                                  USE Dehy2_Operator, 8 FOR M_L_dehy
                                  WAIT M_R_dehyb

                                  A_Type = 2
                                    ROUTE 1
                                  }}                1    Block           Scale2             FIRST 1                   MOVE WITH Dehy2_Operator,8 THEN FREE


                                            2    Block           EXIT               FIRST 1
Block          Scale1             WAIT UNTIL  V_Need_Bug_Dehy1 < 3
                                  USE Dehy1_Operator FOR M_U_dehy
                                            1    Block           Dehy_Trans1a       IF A_Type = 1, 1          IF  V_NBlocks1a>2 THEN
                                                                                                              {          WAIT UNTIL V_Buggy_Ready1a=1  }
                                                                                                              MOVE WITH Dehy1_Operator,8 THEN FREE

                                                 Block           Dehy_Trans1b       IF A_Type = 2             IF  V_NBlocks1b>2 THEN
                                                                                                              {          WAIT UNTIL V_Buggy_Ready1b=1  }
                                                                                                              MOVE WITH Dehy1_Operator,8 THEN FREE
Block          Scale2             WAIT UNTIL  V_Need_Bug_Dehy2 < 3
                                  USE Dehy2_Operator FOR M_U_dehy
                                            1    Block           Dehy_Trans2a       IF A_Type = 1, 1          IF  V_NBlocks2a>2 THEN
                                                                                                              {          WAIT UNTIL V_Buggy_Ready2a=1  }
                                                                                                              MOVE WITH Dehy2_Operator,8 THEN FREE

                                                 Block           Dehy_Trans2b       IF A_Type = 2             IF  V_NBlocks2b>2 THEN
                                                                                                              {          WAIT UNTIL V_Buggy_Ready2b=1  }
                                                                                                              MOVE WITH Dehy2_Operator,8 THEN FREE
Block          Dehy_Trans1a       IF V_NBlocks1a  < 5 THEN
                                  {
                                   INC V_NBlocks1a
                                   GRAPHIC (V_NBlocks1a +1)
                                  }
                                   ELSE
                                  {
                                  V_NBlocks1a = 1
                                  GRAPHIC (V_NBlocks1a +1)
                                  }
                                  COMBINE 5
                                            1    Block           Dehy_T1a           LOAD 1
                                            2*   NIR_Sample      NIR_SAMPLE_WAIT1   FIRST 1                   V_Test_finish1 = 0

                                                                                                              // MAKING AN NIR SAMPLE FOR TESTING
```

```
Block           Dehy_Trans2a       IF V_NBlocks2a  < 5 THEN
                                   {
                                    INC V_NBlocks2a
                                    GRAPHIC (V_NBlocks2a +1)
                                   }
                                    ELSE
                                   {
                                   V_NBlocks2a = 1
                                   GRAPHIC (V_NBlocks2a +1)
                                   }
                                   COMBINE 5
                                                      1    Block          Dehy_T2a           LOAD 1
                                                      2*   NIR_Sample     NIR_SAMPLE_WAIT2   FIRST 1                 V_Test_finish2 = 0

                                                                                                                     // MAKING AN NIR SAMPLE FOR TESTING
Block           Dehy_Trans1b       IF V_NBlocks1b  < 5 THEN
                                   {
                                    INC V_NBlocks1b
                                    GRAPHIC (V_NBlocks1b +1)
                                   }
                                   ELSE
                                   {
                                    V_NBlocks1b = 1
                                    GRAPHIC (V_NBlocks1b +1)
                                   }
                                   COMBINE 5          1    Block          Dehy_T1b           LOAD 1
                                                      2*   NIR_Sample     NIR_SAMPLE_WAIT1   FIRST 1                 // MAKING AN NIR SAMPLE FOR TESTING
Block           Dehy_Trans2b       IF V_NBlocks2b  < 5 THEN
                                   {
                                    INC V_NBlocks2b
                                    GRAPHIC (V_NBlocks2b +1)
                                   }
                                   ELSE
                                   {
                                    V_NBlocks2b = 1
                                    GRAPHIC (V_NBlocks2b +1)
                                   }
                                   COMBINE 5          1    Block          Dehy_T2b           LOAD 1
                                                      2*   NIR_Sample     NIR_SAMPLE_WAIT2   FIRST 1                 // MAKING AN NIR SAMPLE FOR TESTING
Block           Dehy_T1a                           1    Block          Buggy_Wait1        FIRST 1                 GRAPHIC 7

Block           Dehy_T2a                           1    Block          Buggy_Wait2        FIRST 1                 GRAPHIC 7

Block           Dehy_T1b                           1    Block          Buggy_Wait1        FIRST 1                 GRAPHIC 7

Block           Dehy_T2b                           1    Block          Buggy_Wait2        FIRST 1                 GRAPHIC 7

Dehy_Buggy      Dehy_T1a           GRAPHIC 2
                                   WAIT UNTIL V_NBlocks1a = 5
                                   LOAD 1
                                   GRAPHIC 5          1    Dehy_Buggy     Buggy_Wait1        FIRST 1                 GRAPHIC 4
                                                                                                                     MOVE WITH Dehy1_Operator THEN FREE
                                                                                                                     V_Buggy_Ready1a=0

                                                      Dehy_Buggy     EXIT               FIRST
Dehy_Buggy      Dehy_T2a           GRAPHIC 2
                                   WAIT UNTIL V_NBlocks2a = 5
                                   LOAD 1
                                   GRAPHIC 5          1    Dehy_Buggy     Buggy_Wait2        FIRST 1                 GRAPHIC 4
                                                                                                                     MOVE WITH Dehy2_Operator THEN FREE
                                                                                                                     V_Buggy_Ready2a=0

                                                      Dehy_Buggy     EXIT               FIRST
Dehy_Buggy      Dehy_T1b           GRAPHIC 3
                                   WAIT UNTIL V_NBlocks1b = 5
                                   LOAD 1
                                   GRAPHIC 5          1    Dehy_Buggy     Buggy_Wait1        FIRST 1                 GRAPHIC 4
                                                                                                                     MOVE WITH Dehy1_Operator THEN FREE
```

```
                                                                                                                    V_Buggy_Ready1b=0
                                                Dehy_Buggy      EXIT              FIRST
Dehy_Buggy      Dehy_T2b        GRAPHIC 3
                                WAIT UNTIL V_NBlocks2b = 5
                                LOAD 1
                                GRAPHIC 5           1   Dehy_Buggy      Buggy_Wait2       FIRST 1           GRAPHIC 4
                                                                                                            MOVE WITH Dehy2_Operator THEN FREE
                                                                                                            V_Buggy_Ready2b=0
                                                Dehy_Buggy      EXIT              FIRST
Block           Buggy_Wait1                     1   Block           MixerQ            FIRST 1
Block           Buggy_Wait2                     1   Block           MixerQ            FIRST 1
Dehy_Buggy      Buggy_Wait1     WAIT UNTIL V_Test_finish1 > 0
                                DEC V_Test_finish1

                                /* ASSIGNING ORDER IN WHICH WORKERS
                                  GET NEXT AVAILABLE EMPTY DEHY BUGGY */

                                IF V_1_in_line_dehy = 0 THEN
                                {V_1_in_line_dehy = 1}
                                ELSE
                                { IF  V_2_in_line_dehy = 0 THEN
                                  {V_2_in_line_dehy = 1}
                                  ELSE
                                  {         IF  V_3_in_line_dehy = 0 THEN
                                            {V_3_in_line_dehy = 1}
                                            ELSE
                                            {         IF  V_4_in_line_dehy = 0 THEN
                                                      {V_4_in_line_dehy = 1}
                                                      ELSE
                                                      {         IF  V_5_in_line_dehy = 0 THEN
                                                                {V_5_in_line_dehy = 1}
                                                                ELSE
                                                                {         IF  V_6_in_line_dehy = 0 THEN
                                                                          {V_6_in_line_dehy = 1}
                                                                          ELSE
                                                                          {         IF  V_7_in_line_dehy = 0 THEN
                                                                                    {V_7_in_line_dehy = 1}
                                ELSE
                                { IF  V_8_in_line_dehy = 0 THEN
                                  {V_8_in_line_dehy = 1}
                                  ELSE
                                  {         IF  V_9_in_line_dehy = 0 THEN
                                            {V_9_in_line_dehy = 1}
                                            ELSE
                                            {         IF  V_10_in_line_dehy = 0 THEN
                                                      {V_10_in_line_dehy = 1}
                                                      ELSE
                                                      {         IF  V_11_in_line_dehy = 0 THEN
                                                                {V_11_in_line_dehy = 1}
                                                                ELSE
                                                                {         IF  V_12_in_line_dehy = 0 THEN
                                                                          {V_12_in_line_dehy = 1}
                                                                          ELSE
                                                                          {         IF  V_13_in_line_dehy = 0 THEN
                                                                                    {V_13_in_line_dehy = 1}
                                ELSE
                                { IF  V_14_in_line_dehy = 0 THEN
                                  {V_14_in_line_dehy = 1}
                                  ELSE
                                  {         IF  V_15_in_line_dehy = 0 THEN
                                            {V_15_in_line_dehy = 1}
                                            ELSE
                                            {         V_16_in_line_dehy = 1}
                                  }
                                }}}}}}}}}}}}}}}
                                        1   Dehy_Buggy      MixerQ            FIRST 1           INC V_MixerQ
                                                                                                MOVE WITH Dehy1_Operator,8
                                                                                                INC V_Need_Bug_Dehy1
```

```
                                                                                                                                IF V_Need_Bug_Dehy1= 4 THEN
                                                                                                                                {
                                                                                                                                V_Ungroup_Dehy1_Buggy = 1
                                                                                                                                WAIT UNTIL V_1_in_line_dehy = 1
                                                                                                                                WAIT UNTIL V_Ungroup_Dehy1_Buggy = 3
                                                                                                                                }

                                                                                                                                FREE Dehy1_Operator
Dehy_Buggy      Buggy_Wait2         WAIT UNTIL V_Test_finish2 > 0
                                    DEC V_Test_finish2

                                    /* ASSIGNING ORDER IN WHICH WORKERS
                                      GET NEXT AVAILABLE EMPTY DEHY BUGGY */

                                    IF V_1_in_line_dehy = 0 THEN
                                    {V_1_in_line_dehy = 2}
                                    ELSE
                                    { IF  V_2_in_line_dehy = 0 THEN
                                      {V_2_in_line_dehy = 2}
                                      ELSE
                                      {         IF  V_3_in_line_dehy = 0 THEN
                                                {V_3_in_line_dehy = 2}
                                                ELSE
                                                {         IF  V_4_in_line_dehy = 0 THEN
                                                          {V_4_in_line_dehy = 2}
                                                          ELSE
                                                          {         IF  V_5_in_line_dehy = 0 THEN
                                                                    {V_5_in_line_dehy = 2}
                                                                    ELSE
                                                                    {         IF  V_6_in_line_dehy = 0 THEN
                                                                              {V_6_in_line_dehy = 2}
                                                                              ELSE
                                                                              {         IF  V_7_in_line_dehy = 0 THEN
                                                                                        {V_7_in_line_dehy = 2}
                                    ELSE
                                    { IF  V_8_in_line_dehy = 0 THEN
                                      {V_8_in_line_dehy = 2}
                                      ELSE
                                      {         IF  V_9_in_line_dehy = 0 THEN
                                                {V_9_in_line_dehy = 2}
                                                ELSE
                                                {         IF  V_10_in_line_dehy = 0 THEN
                                                          {V_10_in_line_dehy = 2}
                                                          ELSE
                                                          {         IF  V_11_in_line_dehy = 0 THEN
                                                                    {V_11_in_line_dehy = 2}
                                                                    ELSE
                                                                    {         IF  V_12_in_line_dehy = 0 THEN
                                                                              {V_12_in_line_dehy = 2}
                                                                              ELSE
                                                                              {         IF  V_13_in_line_dehy = 0 THEN
                                                                                        {V_13_in_line_dehy = 2}
                                    ELSE
                                    { IF  V_14_in_line_dehy = 0 THEN
                                      {V_14_in_line_dehy = 2}
                                      ELSE
                                      {         IF  V_15_in_line_dehy = 0 THEN
                                                {V_15_in_line_dehy = 2}
                                                ELSE
                                                {         V_16_in_line_dehy = 2}
                                      }
                                    }}}}}}}}}}}}}}}
                                              1    Dehy_Buggy       MixerQ              FIRST 1                  INC V_MixerQ
                                                                                                                MOVE WITH Dehy2_Operator,8
                                                                                                                INC V_Need_Bug_Dehy2
```

```
                                                                                                                                 IF V_Need_Bug_Dehy2 =4 THEN
                                                                                                                                 {
                                                                                                                                 V_Ungroup_Dehy2_Buggy = 1
                                                                                                                                 WAIT UNTIL V_1_in_line_dehy = 2
                                                                                                                                 WAIT UNTIL V_Ungroup_Dehy2_Buggy = 3
                                                                                                                                 }

                                                                                                                                 FREE Dehy2_Operator
Dehy_Buggy       Empty_buggies1     GRAPHIC 6          1    Dehy_Buggy       Dehy_T1a           EMPTY 1                      MOVE WITH Dehy1_Operator THEN FREE
                                                                                                                                  V_Buggy_Ready1a=1
                                                            Dehy_Buggy       Dehy_T1b           EMPTY                        MOVE WITH Dehy1_Operator THEN FREE
                                                                                                                                  V_Buggy_Ready1b=1
Two_Dehy_Buggies Ungp_Empty_buggies1 UNGROUP
Dehy_Buggy       Ungp_Empty_buggies1                   1    Dehy_Buggy       Empty_buggies1     EMPTY 1                      MOVE WITH Dehy1_Operator

Dehy_Buggy       Empty_buggies2     GRAPHIC 6          1    Dehy_Buggy       Dehy_T2a           EMPTY 1                      MOVE WITH Dehy2_Operator THEN FREE
                                                                                                                                  V_Buggy_Ready2a=1
                                                            Dehy_Buggy       Dehy_T2b           EMPTY                        MOVE WITH Dehy2_Operator THEN FREE
                                                                                                                                  V_Buggy_Ready2b=1
Two_Dehy_Buggies Ungp_Empty_buggies2 UNGROUP
Dehy_Buggy       Ungp_Empty_buggies2                   1    Dehy_Buggy       Empty_buggies2     EMPTY 1                      MOVE WITH Dehy2_Operator

Dehy_Buggy       MixerQ             /* SETTING UP FIRST DEHY BUGGIES FOR
                                      PROCESSING AT START OF PROGRAM */

                                    WAIT UNTIL V_Mixer_Idle  >  0
                                    DEC V_Mixer_Idle
                                    DEC V_MixerQ
                                                       1    Dehy_Buggy       Mixer_Load1        IF V_MIXER1 = 0, 1           V_MIXER1 = 1
                                                                                                                                 MOVE WITH Mixer_Operator

                                                            Dehy_Buggy       Mixer_Load2        IF V_MIXER2 = 0              V_MIXER2 = 1
                                                                                                                                 MOVE WITH Mixer_Operator

                                                            Dehy_Buggy       Mixer_Load3        IF V_MIXER3 = 0              V_MIXER3 = 1
                                                                                                                                 MOVE WITH Mixer_Operator

                                                       2*   Mixer_Flat       Changeover_Buggy2  FIRST 1                      GRAPHIC 9
Block            MixerQ                                1    Block            Mixer_Load1        FIRST 1
                                                            Block            Mixer_Load2        FIRST
                                                            Block            Mixer_Load3        FIRST
Mixer_Flat       Changeover_Buggy2  IF A_Mix_Type < 4 THEN
                                    { ROUTE 2}
                                    ELSE
                                    {

                                    INC V_NCO_loaded2

                                    A_Mix_Type=4

                                    IF V_NCO_loaded2 =1 THEN
                                    {
                                    WAIT UNTIL V_NCO_loaded1=3
                                    WAIT UNTIL V_TUB_Avail + V_Empty_mixer_flats = 3
                                    ROUTE 1
                                    }
                                    ELSE
                                    { IF V_NCO_loaded2 =2 THEN
                                      {
                                      ROUTE 2
                                      }
                                      ELSE
                                      {          IF V_NCO_loaded2 =3 THEN
                                                 {
                                                 V_NCO_loaded2 = 0
                                                 ROUTE 2
                                                 }
                                      }
```

```
                          }}                  1    Mixer_Flat       BlockerQ          FIRST 1                  INC  V_BlockerQ
                                              2    Mixer_Flat       EXIT              FIRST 1
Block          Mixer_Load1      GRAPHIC 6
                                GET Mixer_Operator
                                              1    Block            Mixer1a           FIRST 1                  MOVE FOR M_L_mixer*2/3 MIN
                                                                                                               FREE Mixer_Operator
Block          Mixer_Load2      GRAPHIC 6
                                GET Mixer_Operator
                                              1    Block            Mixer2a           FIRST 1                  MOVE FOR M_L_mixer*2/3 MIN
                                                                                                               FREE Mixer_Operator
Block          Mixer_Load3      GRAPHIC 6
                                GET Mixer_Operator
                                              1    Block            Mixer3a           FIRST 1                  MOVE FOR M_L_mixer*2/3 MIN
                                                                                                               FREE Mixer_Operator
Dehy_Buggy     Mixer_Load1      IF A_Mix_Type = 4 THEN
                                {
                                FREE ALL
                                V_Ready_4_Changeover = V_Ready_4_Changeover + 1/3
                                INC V_NCO_loaded1
                                 WAIT UNTIL V_Ready_4_Changeover = 5
                                 WAIT UNTIL V_Ready_4_Changeover = 0
                                INC  V_Mixer_Idle
                                V_MIXER1 = 0
                                DEC V_NCO_loaded1
                                ROUTE 2
                                }
                                ELSE
                                {
                                GRAPHIC 2
                                UNLOAD 1
                                WAIT M_L_mixer*2/3 MIN
                                  ROUTE 1
                                }                 1    Dehy_Buggy       Empty_buggies     FIRST 1                  MOVE WITH Mixer_Operator THEN FREE
                                                                                                               INC V_Empty_Buggies
                                              2    Dehy_Buggy       EXIT              FIRST 1
Dehy_Buggy     Mixer_Load2      IF A_Mix_Type = 4 THEN
                                {
                                FREE ALL
                                V_Ready_4_Changeover = V_Ready_4_Changeover + 1/3
                                INC V_NCO_loaded1
                                 WAIT UNTIL V_Ready_4_Changeover = 5
                                 WAIT UNTIL V_Ready_4_Changeover = 0
                                INC  V_Mixer_Idle
                                V_MIXER2 = 0
                                DEC V_NCO_loaded1
                                ROUTE 2
                                }
                                ELSE
                                {
                                GRAPHIC 2
                                UNLOAD 1
                                WAIT M_L_mixer*2/3 MIN
                                  ROUTE 1
                                }                 1    Dehy_Buggy       Empty_buggies     FIRST 1                  MOVE WITH Mixer_Operator  THEN FREE
                                                                                                               INC V_Empty_Buggies
                                              2    Dehy_Buggy       EXIT              FIRST 1
Dehy_Buggy     Mixer_Load3      IF A_Mix_Type = 4 THEN
                                {
                                FREE ALL
                                V_Ready_4_Changeover = V_Ready_4_Changeover + 1/3
                                INC V_NCO_loaded1
                                 WAIT UNTIL V_Ready_4_Changeover = 5
                                 WAIT UNTIL V_Ready_4_Changeover = 0
                                INC  V_Mixer_Idle
                                V_MIXER3 = 0
                                DEC V_NCO_loaded1
                                ROUTE 2
                                }
```

```
                                ELSE
                                {
                                GRAPHIC 2
                                UNLOAD 1
                                WAIT M_L_mixer*2/3 MIN
                                ROUTE 1
                                }                   1    Dehy_Buggy       Empty_buggies       FIRST 1                   MOVE WITH Mixer_Operator THEN FREE
                                                                                                                       INC V_Empty_Buggies
                                                    2    Dehy_Buggy       EXIT                FIRST 1
Dehy_Buggy      Empty_buggies   GRAPHIC 6

                                WAIT UNTIL V_1_in_line_dehy = 1
                                  OR V_1_in_line_dehy = 2

                                /* CHECKING TO SEE IF FIRST WORKER IN LINE
                                  NEEDS MORE THAN ONE BUGGY */

                                IF  V_1_in_line_dehy = 1 THEN
                                { IF V_Ungroup_Dehy1_Buggy = 1
                                            OR V_Ungroup_Dehy1_Buggy = 2 THEN
                                  {         ROUTE 3   }
                                  ELSE
                                  {         // UPDATING WHICH WORKER GETS THE NEXT AVAILABLE BUGGY

                                            V_1_in_line_dehy =  V_2_in_line_dehy
                                            V_2_in_line_dehy =  V_3_in_line_dehy
                                            V_3_in_line_dehy =  V_4_in_line_dehy
                                            V_4_in_line_dehy =  V_5_in_line_dehy
                                            V_5_in_line_dehy =  V_6_in_line_dehy
                                            V_6_in_line_dehy =  V_7_in_line_dehy
                                            V_7_in_line_dehy =  V_8_in_line_dehy
                                            V_8_in_line_dehy =  V_9_in_line_dehy
                                            V_9_in_line_dehy =  V_10_in_line_dehy
                                            V_10_in_line_dehy =  V_11_in_line_dehy
                                            V_11_in_line_dehy =  V_12_in_line_dehy
                                            V_12_in_line_dehy =  V_13_in_line_dehy
                                            V_13_in_line_dehy =  V_14_in_line_dehy
                                            V_14_in_line_dehy =  V_15_in_line_dehy
                                            V_15_in_line_dehy =  V_16_in_line_dehy
                                            V_16_in_line_dehy =  0

                                            DEC V_Need_Bug_Dehy1
                                            GET Dehy1_Operator,8
                                            DEC V_Empty_Buggies
                                            ROUTE 1   }
                                }
                                ELSE
                                { IF  V_1_in_line_dehy = 2 THEN
                                  {         IF V_Ungroup_Dehy2_Buggy = 1
                                                      OR V_Ungroup_Dehy2_Buggy = 2 THEN
                                            {         ROUTE 3   }
                                            ELSE
                                            {         // UPDATING WHICH WORKER GETS THE NEXT AVAILABLE BUGGY

                                                      V_1_in_line_dehy =  V_2_in_line_dehy
                                                      V_2_in_line_dehy =  V_3_in_line_dehy
                                                      V_3_in_line_dehy =  V_4_in_line_dehy
                                                      V_4_in_line_dehy =  V_5_in_line_dehy
                                                      V_5_in_line_dehy =  V_6_in_line_dehy
                                                      V_6_in_line_dehy =  V_7_in_line_dehy
                                                      V_7_in_line_dehy =  V_8_in_line_dehy
                                                      V_8_in_line_dehy =  V_9_in_line_dehy
                                                      V_9_in_line_dehy =  V_10_in_line_dehy
                                                      V_10_in_line_dehy =  V_11_in_line_dehy
                                                      V_11_in_line_dehy =  V_12_in_line_dehy
                                                      V_12_in_line_dehy =  V_13_in_line_dehy
                                                      V_13_in_line_dehy =  V_14_in_line_dehy
                                                      V_14_in_line_dehy =  V_15_in_line_dehy
```

```
                                        V_15_in_line_dehy =  V_16_in_line_dehy
                                        V_16_in_line_dehy =  0

                                        DEC V_Need_Bug_Dehy2
                                        GET Dehy2_Operator,8
                                        DEC V_Empty_Buggies
                                        ROUTE 2   }
                                } }
                                       1    Dehy_Buggy       Empty_buggies1      FIRST 1                    MOVE WITH Dehy1_Operator,8  THEN FREE
                                       2    Dehy_Buggy       Empty_buggies2      FIRST 1                    MOVE WITH Dehy2_Operator,8  THEN FREE
                                       3    Dehy_Buggy       Gp_Empty_buggies    FIRST 1
Dehy_Buggy       Gp_Empty_buggies    /* IF WORKER NEEDS TWO BUGGIES THEN GROUP
                                       TWO BUGGIES TOGETHER */

                                     IF V_Need_Bug_Dehy1 = 4
                                       AND V_1_in_line_dehy = 1 THEN
                                     {
                                     INC V_Ungroup_Dehy1_Buggy
                                     GROUP 2 AS Two_Dehy_Buggies
                                     }
                                     ELSE
                                     {
                                     IF V_Need_Bug_Dehy2 = 4
                                       AND V_1_in_line_dehy = 2 THEN
                                     {
                                     INC V_Ungroup_Dehy2_Buggy
                                     GROUP 2 AS Two_Dehy_Buggies
                                     }
                                     }
Two_Dehy_Buggies Gp_Empty_buggies    // CHECKING TO SEE WHERE THE TWO BUGGIES GO

                                     IF V_Need_Bug_Dehy1 = 4
                                       AND V_1_in_line_dehy = 1
                                       AND V_2_in_line_dehy = 1 THEN
                                     { // UPDATING WHICH WORKER GETS THE NEXT AVAILABLE BUGGY

                                       V_1_in_line_dehy =  V_3_in_line_dehy
                                       V_2_in_line_dehy =  V_4_in_line_dehy
                                       V_3_in_line_dehy =  V_5_in_line_dehy
                                       V_4_in_line_dehy =  V_6_in_line_dehy
                                       V_5_in_line_dehy =  V_7_in_line_dehy
                                       V_6_in_line_dehy =  V_8_in_line_dehy
                                       V_7_in_line_dehy =  V_9_in_line_dehy
                                       V_8_in_line_dehy =  V_10_in_line_dehy
                                       V_9_in_line_dehy =  V_11_in_line_dehy
                                       V_10_in_line_dehy =  V_12_in_line_dehy
                                       V_11_in_line_dehy =  V_13_in_line_dehy
                                       V_12_in_line_dehy =  V_14_in_line_dehy
                                       V_13_in_line_dehy =  V_15_in_line_dehy
                                       V_14_in_line_dehy =  V_16_in_line_dehy
                                       V_15_in_line_dehy =  0
                                       V_16_in_line_dehy =  0

                                       V_Need_Bug_Dehy1 = V_Need_Bug_Dehy1 - 2
                                       GET Dehy1_Operator,9
                                       V_Empty_Buggies = V_Empty_Buggies -2
                                       ROUTE 1   }
                                     ELSE
                                     { IF V_Need_Bug_Dehy2 = 4
                                                 AND V_1_in_line_dehy = 2
                                                 AND V_2_in_line_dehy = 2 THEN
                                       {         // UPDATING WHICH WORKER GETS THE NEXT AVAILABLE BUGGY

                                                 V_1_in_line_dehy =  V_3_in_line_dehy
                                                 V_2_in_line_dehy =  V_4_in_line_dehy
                                                 V_3_in_line_dehy =  V_5_in_line_dehy
                                                 V_4_in_line_dehy =  V_6_in_line_dehy
                                                 V_5_in_line_dehy =  V_7_in_line_dehy
```

```
                                V_6_in_line_dehy =  V_8_in_line_dehy
                                V_7_in_line_dehy =  V_9_in_line_dehy
                                V_8_in_line_dehy =  V_10_in_line_dehy
                                V_9_in_line_dehy =  V_11_in_line_dehy
                                V_10_in_line_dehy =  V_12_in_line_dehy
                                V_11_in_line_dehy =  V_13_in_line_dehy
                                V_12_in_line_dehy =  V_14_in_line_dehy
                                V_13_in_line_dehy =  V_15_in_line_dehy
                                V_14_in_line_dehy =  V_16_in_line_dehy
                                V_15_in_line_dehy =  0
                                V_16_in_line_dehy =  0

                                V_Need_Bug_Dehy2 = V_Need_Bug_Dehy2 - 2
                                GET Dehy2_Operator,9
                                V_Empty_Buggies = V_Empty_Buggies -2
                                ROUTE 2   }
                        }
                                        1    Two_Dehy_Buggies Ungp_Empty_buggies1 FIRST 1                    MOVE WITH Dehy1_Operator
                                        2    Two_Dehy_Buggies Ungp_Empty_buggies2 FIRST 1                    MOVE WITH Dehy2_Operator
Block           Mixer1a         GRAPHIC 8
                                IF A_Mix_Type = 1 THEN
                                { WAIT M_R_mixer_P1   }
                                ELSE
                                { IF A_Mix_Type = 2 THEN
                                  {         WAIT M_R_mixer_P2   }
                                  ELSE
                                  {         IF  A_Mix_Type = 3 THEN
                                            {         WAIT M_R_mixer_P3   }
                                  }
                                }

                                V_Need_Tub_Mix1 = 2
                                V_MIXER = 1

                                // ASSINGNING ORDER OF UNLOADING MIXER

                                IF V_1_in_line_mix = 0 THEN
                                {
                                 V_1_in_line_mix = V_MIXER
                                }
                                ELSE
                                { IF  V_2_in_line_mix = 0 THEN
                                  {
                                   V_2_in_line_mix = V_MIXER
                                  }
                                  ELSE
                                  {                  IF V_3_in_line_mix = 0 THEN
                                                     {
                                                      V_3_in_line_mix = V_MIXER
                                                     }
                                                     ELSE
                                                     {
                                                      V_4_in_line_mix = V_MIXER
                                                     }
                                  }
                                }                   1    Mix             Mixer_Unload1      LOAD 1          GRAPHIC 2
                                                                                                            WAIT M_U_mixer
                                                    2*   Block           Mixer1b            FIRST 1         //CREATING SECOND MIX TO BE UNLOADED
                                                                                                            GRAPHIC 8
Block           Mixer2a         GRAPHIC 8
                                IF A_Mix_Type = 1 THEN
                                { WAIT M_R_mixer_P1   }
                                ELSE
                                { IF A_Mix_Type = 2 THEN
                                  {         WAIT M_R_mixer_P2   }
                                  ELSE
                                  {         IF  A_Mix_Type = 3 THEN
                                            {         WAIT M_R_mixer_P3   }
```

```
                                  }
                                }

                                V_Need_Tub_Mix2 = 2
                                V_MIXER = 2

                                // ASSINGNING ORDER OF UNLOADING MIXER

                                IF V_1_in_line_mix = 0 THEN
                                {
                                 V_1_in_line_mix = V_MIXER
                                }
                                ELSE
                                { IF  V_2_in_line_mix = 0 THEN
                                  {
                                   V_2_in_line_mix = V_MIXER
                                  }
                                  ELSE
                                  {                   IF V_3_in_line_mix = 0 THEN
                                                      {
                                                       V_3_in_line_mix = V_MIXER
                                                      }
                                                      ELSE
                                                      {
                                                       V_4_in_line_mix = V_MIXER
                                                      }
                                  }
                                }                     1    Mix              Mixer_Unload2       LOAD 1                    GRAPHIC 2
                                                                                                                        WAIT M_U_mixer
                                                      2*   Block            Mixer2b             FIRST 1                   //CREATING SECOND MIX TO BE UNLOADED
                                                                                                                        GRAPHIC 8
Block           Mixer3a           GRAPHIC 8
                                  IF A_Mix_Type = 1 THEN
                                  { WAIT M_R_mixer_P1   }
                                  ELSE
                                  { IF A_Mix_Type = 2 THEN
                                    {         WAIT M_R_mixer_P2   }
                                    ELSE
                                    {         IF  A_Mix_Type = 3 THEN
                                              {         WAIT M_R_mixer_P3   }
                                    }
                                  }

                                  V_Need_Tub_Mix3 = 2
                                  V_MIXER = 3

                                  // ASSINGNING ORDER OF UNLOADING MIXER

                                  IF V_1_in_line_mix = 0 THEN
                                  {
                                   V_1_in_line_mix = V_MIXER
                                  }
                                  ELSE
                                  { IF  V_2_in_line_mix = 0 THEN
                                    {
                                     V_2_in_line_mix = V_MIXER
                                    }
                                    ELSE
                                    {                   IF V_3_in_line_mix = 0 THEN
                                                        {
                                                         V_3_in_line_mix = V_MIXER
                                                        }
                                                        ELSE
                                                        {
                                                         V_4_in_line_mix = V_MIXER
                                                        }
                                    }
                                  }                   1    Mix              Mixer_Unload3       LOAD 1                    GRAPHIC 2
```

							WAIT M_U_mixer
			2*	Block	Mixer3b	FIRST 1	//CREATING SECOND MIX TO BE UNLOADED GRAPHIC 8
Block	Mixer1b		1	Mix	Mixer_Unload1	LOAD 1	GRAPHIC 2 WAIT M_U_mixer
Block	Mixer2b		1	Mix	Mixer_Unload2	LOAD 1	GRAPHIC 2 WAIT M_U_mixer
Block	Mixer3b		1	Mix	Mixer_Unload3	LOAD 1	GRAPHIC 2 WAIT M_U_mixer

Mixer_Flat	Mixer_Trans1	GRAPHIC 5
		WAIT UNTIL V_Need_Tub_Mix1>0
		OR V_Need_Tub_Mix2>0
		OR V_Need_Tub_Mix3>0
		OR V_Need_Tub_Mix4>0
		
		WAIT UNTIL V_TUB_Avail > 0
		
		// ASSIGNING WHERE AVAILABLE MIXER FLATS SHOULD GO
		
		IF V_Need_Tub_Mix1 >0 AND V_1_in_line_mix = 1 THEN
		{
		WAIT UNTIL V_UL1_Idle = 0
		DEC V_Need_Tub_Mix1
		V_UL1_Idle = 1
		ROUTE 1
		}
		ELSE
		{
		IF V_Need_Tub_Mix2>0 AND V_1_in_line_mix = 2 THEN
		{
		WAIT UNTIL V_UL2_Idle = 0
		DEC V_Need_Tub_Mix2
		V_UL2_Idle = 1
		ROUTE 2
		}
		ELSE
		{
		IF V_Need_Tub_Mix3>0 AND V_1_in_line_mix = 3 THEN
		{
		WAIT UNTIL V_UL3_Idle = 0
		DEC V_Need_Tub_Mix3
		V_UL3_Idle = 1
		ROUTE 3
		}
		}
		}
			1	Mixer_Flat	Mixer_Unload1	FIRST 1	DEC V_TUB_Avail
							
							// UPDATING ORDER OF UNLOADING MIXER
							
							IF V_Need_Tub_Mix1 = 0 THEN
							{
							V_1_in_line_mix = V_2_in_line_mix
							V_2_in_line_mix = V_3_in_line_mix
							V_3_in_line_mix = V_4_in_line_mix
							V_4_in_line_mix = 0
							}
							
							MOVE WITH Mixer_Operator, 9

			2	Mixer_Flat	Mixer_Unload2	FIRST 1	DEC V_TUB_Avail
							
							// UPDATING ORDER OF UNLOADING MIXER
							
							IF V_Need_Tub_Mix2 = 0 THEN
							{
							V_1_in_line_mix = V_2_in_line_mix
							V_2_in_line_mix = V_3_in_line_mix
							V_3_in_line_mix = V_4_in_line_mix
							V_4_in_line_mix = 0

```
                                                                                          }

                                                                                          MOVE WITH Mixer_Operator, 9
                                           3   Mixer_Flat     Mixer_Unload3    FIRST 1    DEC V_TUB_Avail

                                                                                          // UPDATING ORDER OF UNLOADING MIXER

                                                                                          IF  V_Need_Tub_Mix3 = 0 THEN
                                                                                          {
                                                                                          V_1_in_line_mix = V_2_in_line_mix
                                                                                          V_2_in_line_mix = V_3_in_line_mix
                                                                                          V_3_in_line_mix = V_4_in_line_mix
                                                                                          V_4_in_line_mix = 0
                                                                                          }

                                                                                          MOVE WITH Mixer_Operator, 9
 Mixer_Flat     Mixer_Unload1     GET Mixer_Operator, 9
                                  LOAD 1
                                  GRAPHIC 1
                                  FREE Mixer_Operator
                                           1   Mixer_Flat     Mixer_Trans1     FIRST 1    MOVE WITH Mixer_Operator THEN FREE
                                                                                          INC V_NMix1

                                                                                          // KEEPING TRACK OF NUMBER OF MIXES UNLOADED FROM
MIXER

                                                                                          IF V_NMix1 = 2 THEN
                                                                                          {
                                                                                          INC V_Mixer_Idle
                                                                                          V_MIXER1 = 0
                                                                                          V_NMix1= 0
                                                                                          }
                                                                                          V_UL1_Idle = 0
 Mixer_Flat     Mixer_Unload2     GET Mixer_Operator, 9
                                  LOAD 1
                                  GRAPHIC 1
                                  FREE Mixer_Operator
                                           1   Mixer_Flat     Mixer_Trans1     FIRST 1    MOVE WITH Mixer_Operator THEN FREE
                                                                                          INC V_NMix2

                                                                                          // KEEPING TRACK OF NUMBER OF MIXES UNLOADED FROM
MIXER

                                                                                          IF V_NMix2=2 THEN
                                                                                          {
                                                                                          INC V_Mixer_Idle
                                                                                          V_MIXER2 = 0
                                                                                          V_NMix2 = 0
                                                                                          }
                                                                                          V_UL2_Idle = 0
 Mixer_Flat     Mixer_Unload3     GET Mixer_Operator, 9
                                  LOAD 1
                                  GRAPHIC 1
                                  FREE Mixer_Operator
                                           1   Mixer_Flat     Mixer_Trans1     FIRST 1    MOVE WITH Mixer_Operator THEN FREE
                                                                                          INC V_NMix3

                                                                                          // KEEPING TRACK OF NUMBER OF MIXES UNLOADED FROM
MIXER

                                                                                          IF V_NMix3=2 THEN
                                                                                          {
                                                                                          INC V_Mixer_Idle
                                                                                          V_MIXER3 = 0
                                                                                          V_NMix3 = 0
                                                                                          }
                                                                                          V_UL3_Idle =0
 Mix            Mixer_Unload1                1   Mix            Mixer_Trans1     FIRST 1
```

```
Mix             Mixer_Unload2                          1    Mix             Mixer_Trans1       FIRST 1
Mix             Mixer_Unload3                          1    Mix             Mixer_Trans1       FIRST 1
Mixer_Flat      Mixer_Trans1       GRAPHIC 2           1    Mixer_Flat      BlockerQ           FIRST 1                INC V_Need_TUB_Mix
                                                                                                                       MOVE WITH Mixer_Tranporter
                                                                                                                       INC V_BlockerQ

                                                                                                                        WAIT UNTIL V_Empty_mixer_flats > 0
                                                                                                                       FREE Mixer_Tranporter
                                                                                                                       V_Need_Mixtrans = 1

                                                           Mixer_Flat      EXIT               FIRST
Mix             Mixer_Trans1                           1    Mix             BlockerQ           FIRST 1
Mixer_Flat      BlockerQ           // SETTING UP FIRST MIX FLAT FOR PROCESSING AT START OF PROGRAM

                                   IF V_BlockerQ > 0 THEN
                                   {
                                   WAIT UNTIL V_Blocker_Status = 0
                                   }                   1    Mixer_Flat      Buggy_Location     FIRST 1                DEC V_BlockerQ
                                                                                                                       V_Blocker_Status = 1
                                                                                                                       MOVE WITH Blocker_Operator1 THEN FREE

                                                          2*   Blocker_Buggy   Changeover_Buggy3  FIRST 1
Mix             BlockerQ                               1    Mix             Buggy_Location     FIRST 1
Blocker_Buggy   Changeover_Buggy3  IF A_Mix_Type <4 THEN
                                   { ROUTE 2   }
                                   ELSE
                                   {
                                   A_Mix_Type = 4
                                   GRAPHIC 12
                                   WAIT UNTIL V_TUB_Avail + V_Empty_mixer_flats = 3
                                   IF V_First_CO = 0 THEN
                                   {
                                   WAIT M_R_blocker
                                   WAIT M_U_blocker
                                   WAIT M_T_blocker
                                   WAIT M_Elevator_time
                                   WAIT 30 SEC
                                   WAIT UNTIL V_Need_Bug_Blocker = 0
                                   WAIT UNTIL V_ELEVATORQ  = 0
                                   }
                                   ROUTE 1
                                   }

                                                          1    Blocker_Buggy   Elevator           FIRST 1                INC V_ELEVATORQ
                                                                                                                       GRAPHIC 12
                                                          2    Blocker_Buggy   EXIT               FIRST 1
Mixer_Flat      Buggy_Location     GRAPHIC 1

                                   IF A_Mix_Type = 4 THEN
                                   {
                                   GRAPHIC 9
                                   V_Blocker_Status = 0
                                   INC V_Ready_4_Changeover
                                    WAIT UNTIL V_Ready_4_Changeover = 5
                                    WAIT UNTIL V_Ready_4_Changeover = 0
                                   ROUTE 2
                                   }
                                   ELSE
                                   {
                                   UNLOAD 1

                                   // CHANGING GRAPHIC AS BLOCKS ARE BEING PRCOCESSED

                                   WAIT UNTIL  V_Number_blocks_1 = 2
                                   GRAPHIC 3

                                   WAIT UNTIL  V_Number_blocks_1 = 4
                                   GRAPHIC 4
```

```
                                WAIT UNTIL  V_Number_blocks_1 = 5
                                GET Blocker_Operator1

                                WAIT UNTIL  V_Number_blocks_1 = 6
                                GRAPHIC 5
                                  ROUTE 1
                                }
                                                1    Mixer_Flat       Empty_Mixer_Flat    FIRST 1                  GRAPHIC 7
                                                                                                                   V_Blocker_Status = 0
                                                                                                                   MOVE WITH Blocker_Operator1 THEN FREE
                                                                                                                   V_Number_blocks_1 = 0
                                                                                                                   INC V_Empty_mixer_flats

                                                2    Mixer_Flat       EXIT                FIRST 1
Mixer_Flat      Empty_Mixer_Flat    GRAPHIC 7
                                WAIT UNTIL V_Need_Mixtrans = 1
                                 V_Need_Mixtrans = 0
                                GET  Mixer_Tranporter,10
                                DEC V_Need_TUB_Mix
                                                1    Mixer_Flat       Mixer_Trans1        FIRST 1                  MOVE WITH Mixer_Tranporter THEN FREE
                                                                                                                   DEC V_Empty_mixer_flats
                                                                                                                   INC V_TUB_Avail

Mix             Buggy_Location      GRAPHIC 6           1    Mix              Blocker1            FIRST 6          GRAPHIC 2
                                                             Mix              Blocker2            ALT              GRAPHIC 2
Mix             Blocker1            GET ( Blocker_Operator1 OR  Blocker_Operator2)
                                    WAIT M_L_blocker
                                    FREE ALL
                                    INC V_Number_blocks_1

                                    /* WAIT UNTL BOTH BLOCKERS HAVE BEEN LOADED
                                      AND THEN RUN AND UNLOAD */

                                    WAIT UNTIL V_Number_blocks_1 = 2
                                      OR V_Number_blocks_1 = 4
                                      OR V_Number_blocks_1 = 6
                                    WAIT M_R_blocker
                                    GET ( Blocker_Operator1 OR  Blocker_Operator2)
                                    WAIT  M_U_blocker
                                    FREE ALL            1    Block2           Blocker_Trans       LOAD 1           // CHANGING GRAPHICS TO MATCH  NUMBER OF BLOCKS
PROCESSED

                                                                                                                   IF  V_Number_blocks_1 = 2 THEN
                                                                                                                   {
                                                                                                                   GRAPHIC 2
                                                                                                                   }
                                                                                                                   IF  V_Number_blocks_1 = 4 THEN
                                                                                                                   {
                                                                                                                   GRAPHIC 3
                                                                                                                   }
                                                                                                                   IF  V_Number_blocks_1 = 6 THEN
                                                                                                                   {
                                                                                                                   GRAPHIC 4
                                                                                                                   }

Mix             Blocker2            GET ( Blocker_Operator1 OR  Blocker_Operator2)
                                    WAIT M_L_blocker
                                    FREE ALL
                                    INC V_Number_blocks_1

                                    /* WAIT UNTL BOTH BLOCKERS HAVE BEEN LOADED
                                      AND THEN RUN AND UNLOAD */

                                    WAIT UNTIL V_Number_blocks_1 = 2
                                      OR V_Number_blocks_1 = 4
                                      OR V_Number_blocks_1 = 6
                                    WAIT M_R_blocker
```

```
                          GET ( Blocker_Operator1 OR  Blocker_Operator2)
                          WAIT M_U_blocker
                          FREE ALL             1    Block2           Blocker_Trans       LOAD 1                    // CHANGING GRAPHICS TO MATCH  NUMBER OF BLOCKS
PROCESSED

                                                                                                                     IF  V_Number_blocks_1 = 2 THEN
                                                                                                                     {
                                                                                                                     GRAPHIC 2
                                                                                                                     }
                                                                                                                     IF  V_Number_blocks_1 = 4 THEN
                                                                                                                     {
                                                                                                                     GRAPHIC 3
                                                                                                                     }
                                                                                                                     IF  V_Number_blocks_1 = 6 THEN
                                                                                                                     {
                                                                                                                     GRAPHIC 4
                                                                                                                     }
Blocker_Buggy     Blocker_Trans       GRAPHIC 1
                          LOAD 6
                          GRAPHIC 5            1    Blocker_Buggy    Elevator            FIRST 1                   MOVE WITH Blocker_Operator2
                                                                                                                     INC V_ELEVATORQ
                                                                                                                     WAIT UNTIL V_Elevator_Status = 0
                                                                                                                     FREE Blocker_Operator2
                                                                                                                     V_Need_Bug_Blocker = 1
                                                    Blocker_Buggy    EXIT                FIRST
Block2            Blocker_Trans                     1    Block2           Elevator            FIRST 1
Blocker_Buggy     Elevator            // WAIT FOR ELEVATOR TO BE FREE FOR USE

                          GRAPHIC 10

                          IF A_Mix_Type = 4 THEN
                          { GRAPHIC 12          }

                          WAIT UNTIL V_Elevator_Status = 0
                          GET  UEXtru_Operator
                          V_Elevator_Status = 1

                          IF A_Mix_Type< 4 THEN
                          {
                          WAIT M_Elevator_time
                          }

                          WAIT UNTIL V_UPext2_status = 0
                            OR V_UPext3_status = 0
                            OR V_UPext4_status = 0

                          DEC V_ELEVATORQ      1    Blocker_Buggy    UExt2_Unload_bug    IF V_UPext2_status = 0, 1   MOVE WITH UEXtru_Operator
                                                    Blocker_Buggy    UExt3_Unload_bug    IF V_UPext3_status = 0      MOVE WITH UEXtru_Operator
                                                    Blocker_Buggy    UExt4_Unload_bug    IF V_UPext4_status = 0      MOVE WITH UEXtru_Operator
                                               2*   Hens_Nest        Changeover_Buggy4   FIRST 1
Block2            Elevator                          1    Block2           UExt2_Unload_bug    FIRST 1
                                                         Block2           UExt3_Unload_bug    FIRST
                                                         Block2           UExt4_Unload_bug    FIRST

Hens_Nest         Changeover_Buggy4   IF A_Mix_Type < 4 then
                          { ROUTE 2   }
                          ELSE
                          {
                          A_Mix_Type=4
                          IF V_First_CO = 0 THEN
                          {
                          WAIT M_L_UPext
                          WAIT M_R_UPext
                          WAIT M_T_Dext
                          WAIT M_U_Dext
                          WAIT UNTIL V_CUTTERQ  = 0
                          WAIT UNTIL  V_CUT3_READY=0
                            AND V_CUT1_READY=0
                            AND V_CUT4_READY=0
```

```
                                  AND V_CUT7_READY=0
                                WAIT UNTIL V_Can_flats_in_use = 0
                                }
                                V_First_CO = 0
                                ROUTE 1
                                }                   1    Hens_Nest       CutterQ          FIRST 1                  INC V_CUTTERQ
                                                                                                                   GRAPHIC 7
                                                    2    Hens_Nest       EXIT             FIRST 1
NIR_Sample      NIR_SAMPLE_WAIT1  INC V_NIR_sample1
                                WAIT UNTIL V_NIR_sample1 = 2
                                GROUP 2 AS NIR_Sample
                                A_Type = 1
                                GRAPHIC 2
                                V_NIR_sample1 = 0
                                                    1    NIR_Sample      NIR_Pack         FIRST 1                  MOVE WITH Dehy1_Operator THEN FREE
                                                                                                                   INC V_NIRQ

NIR_Sample      NIR_SAMPLE_WAIT2  INC V_NIR_sample2
                                WAIT UNTIL V_NIR_sample2 = 2
                                GROUP 2 AS NIR_Sample
                                A_Type = 2
                                GRAPHIC 2
                                V_NIR_sample2 = 0
                                                    1    NIR_Sample      NIR_Pack         FIRST 1                  MOVE WITH Dehy2_Operator  THEN FREE
                                                                                                                   INC V_NIRQ
NIR_Sample      NIR_Pack          DEC V_NIRQ
                                USE NIR_Tester FOR M_L_NIR
                                GRAPHIC 3           1    NIR_Sample      NIR_Test         FIRST 1                  MOVE WITH NIR_Tester

NIR_Sample      NIR_Test          WAIT M_R_NIR
                                WAIT M_D_NIR
                                                    1    NIR_Sample      NIR_Water        FIRST 1                  MOVE WITH NIR_Tester
NIR_Sample      NIR_Water         WAIT M_U_NIR      1    NIR_Sample      EXIT             IF A_Type = 1, 1         V_Test_finish1 = 2
                                                                                                                   FREE NIR_Tester
                                                         NIR_Sample      EXIT             IF A_Type = 2            V_Test_finish2 = 2
                                                                                                                   FREE NIR_Tester
Blocker_Buggy   UExt2_Unload_bug  IF A_Mix_Type = 4 THEN
                                {
                                FREE ALL
                                WAIT UNTIL  V_UPext3_status = 0 AND V_UPext4_status = 0
                                INC  V_Ready_4_Changeover
                                 WAIT UNTIL V_Ready_4_Changeover = 5
                                 WAIT UNTIL V_Ready_4_Changeover = 0
                                FREE ALL
                                V_Elevator_Status = 0
                                ROUTE 3
                                }
                                ELSE
                                {
                                V_UPext2_status = 1
                                GRAPHIC 11
                                USE  UEXtru_Operator FOR M_L_UPext/3 MIN
                                UNLOAD 2
                                GRAPHIC 1
                                INC V_UNLOADED_BLOCKS

                                // CHECK TO SEE IF MORE BLOCKS NEED TO BE PROCESSED

                                IF V_UNLOADED_BLOCKS < 3 THEN
                                {
                                WAIT UNTIL V_UPext3_status = 0
                                  OR V_UPext4_status = 0
                                ROUTE 1
                                }
                                ELSE
                                {
                                V_UNLOADED_BLOCKS = 0
                                ROUTE 2
```

```
                                }
                                }                   1    Blocker_Buggy    UExt3_Unload_bug    IF V_UPext3_status  = 0, 1  MOVE WITH UExtru_Operator THEN FREE

                                                         Blocker_Buggy    UExt4_Unload_bug    IF V_UPext4_status= 0       MOVE WITH UExtru_Operator THEN FREE

                                                    2    Blocker_Buggy    Elevator            FIRST 1                     MOVE WITH UExtru_Operator

                                                    3    Blocker_Buggy    EXIT                FIRST 1
Block2          UExt2_Unload_bug    GRAPHIC 2           1    Block2           UExt2_combine       FIRST 1
Block2          UExt2_combine       COMBINE 2           1    Block2           UExt2               FIRST 1
Block2          UExt2               USE UExtru_Operator, 10 FOR M_L_UPext*2/3 MIN
                                    GRAPHIC 5
                                    V_UExt2_Ready = 1
                                                    1    Chords           DExt2               LOAD 1                      GRAPHIC 3
Hens_Nest       DExt2               GRAPHIC 1
                                    WAIT UNTIL V_UExt2_Ready = 1
                                    USE DExtru_Operator FOR M_L_Dext
                                    LOAD 1
                                    GRAPHIC 4
                                    WAIT M_R_UPext
                                    GET DExtru_Operator
                                    WAIT M_U_Dext
                                    V_Need_Nest2 = 1    1    Hens_Nest        CutterQ             FIRST 1                     GRAPHIC 5
                                                                                                                             V_UExt2_Ready = 0
                                                                                                                             V_UPext2_status = 0
                                                                                                                             MOVE WITH DExtru_Operator THEN FREE
                                                                                                                             INC V_CUTTERQ
Chords          DExt2               GRAPHIC 2           1    Chords           CutterQ             FIRST 1
Blocker_Buggy   UExt3_Unload_bug    IF A_Mix_Type = 4 THEN
                                    {
                                    WAIT UNTIL  V_UPext3_status = 0 AND V_UPext4_status = 0
                                    Get  UExtru_Operator
                                    INC V_Ready_4_Changeover
                                     WAIT UNTIL V_Ready_4_Changeover = 5
                                     WAIT UNTIL V_Ready_4_Changeover = 0
                                    FREE ALL
                                    V_Elevator_Status = 0
                                    ROUTE 3
                                    }
                                    ELSE
                                    {
                                    V_UPext3_status = 1
                                    GRAPHIC 11
                                    USE UExtru_Operator FOR M_L_UPext/3 MIN
                                    UNLOAD 2
                                    GRAPHIC 1
                                    INC V_UNLOADED_BLOCKS

                                    // CHECK TO SEE IF MORE BLOCKS NEED TO BE PROCESSED

                                    IF V_UNLOADED_BLOCKS < 3 THEN
                                    {
                                    WAIT UNTIL V_UPext4_status = 0
                                      OR V_UPext2_status = 0
                                    ROUTE 1
                                    }
                                    ELSE
                                    {
                                    V_UNLOADED_BLOCKS = 0
                                    ROUTE 2
                                    }
                                    }                   1    Blocker_Buggy    UExt4_Unload_bug    IF V_UPext4_status = 0, 1   MOVE WITH UExtru_Operator  THEN FREE
                                                             Blocker_Buggy    UExt2_Unload_bug    IF V_UPext2_status = 0      MOVE WITH UExtru_Operator THEN FREE
                                                    2    Blocker_Buggy    Elevator            FIRST 1                     MOVE WITH UExtru_Operator

                                                    3    Blocker_Buggy    EXIT                FIRST 1
Block2          UExt3_Unload_bug    GRAPHIC 2           1    Block2           UExt3_combine       FIRST 1
```

```
Block2          UExt3_combine       COMBINE 2           1    Block2          UExt3              FIRST 1
Block2          UExt3               USE UEXtru_Operator, 10 FOR M_L_UPext*2/3 MIN
                                    GRAPHIC 5
                                    V_UExt3_Ready = 1
                                                        1    Chords          DExt3              LOAD 1                     GRAPHIC 3
Hens_Nest       DExt3               GRAPHIC 1
                                    WAIT UNTIL V_UExt3_Ready = 1
                                    USE DEXtru_Operator FOR M_L_Dext
                                    LOAD 1
                                    GRAPHIC 4
                                    WAIT M_R_UPext
                                    GET DEXtru_Operator
                                    WAIT M_U_Dext
                                    V_Need_Nest3 = 1    1    Hens_Nest       CutterQ            FIRST 1                    GRAPHIC 5
                                                                                                                           V_UExt3_Ready = 0
                                                                                                                           V_UPext3_status = 0
                                                                                                                           MOVE WITH DEXtru_Operator  THEN FREE
                                                                                                                           INC V_CUTTERQ
Chords          DExt3               GRAPHIC 2           1    Chords          CutterQ            FIRST 1
Blocker_Buggy   UExt4_Unload_bug    IF A_Mix_Type = 4 THEN
                                    {
                                    WAIT UNTIL  V_UPext3_status = 0 AND V_UPext2_status = 0
                                    INC V_Ready_4_Changeover
                                     WAIT UNTIL V_Ready_4_Changeover = 5
                                     WAIT UNTIL V_Ready_4_Changeover = 0
                                    FREE ALL
                                    V_Elevator_Status = 0
                                    ROUTE 3
                                    }
                                    ELSE
                                    {
                                    V_UPext4_status = 1
                                    GRAPHIC 11
                                    USE  UEXtru_Operator FOR M_L_UPext/3 MIN
                                    UNLOAD 2
                                    GRAPHIC 1
                                    INC V_UNLOADED_BLOCKS

                                    // CHECK TO SEE IF MORE BLOCKS NEED TO BE PROCESSED

                                    IF V_UNLOADED_BLOCKS < 3 THEN
                                    {
                                    WAIT UNTIL V_UPext3_status = 0
                                      OR V_UPext2_status = 0
                                    ROUTE 1
                                    }
                                    ELSE
                                    {
                                    V_UNLOADED_BLOCKS = 0
                                    ROUTE 2
                                    }
                                    }                   1    Blocker_Buggy   UExt3_Unload_bug   IF V_UPext3_status = 0, 1  MOVE WITH UEXtru_Operator THEN FREE
                                                             Blocker_Buggy   UExt2_Unload_bug   IF V_UPext2_status = 0      MOVE WITH UEXtru_Operator THEN FREE
                                                        2    Blocker_Buggy   Elevator           FIRST 1                    MOVE WITH UEXtru_Operator

                                                        3    Blocker_Buggy   EXIT               FIRST 1
Block2          UExt4_Unload_bug    GRAPHIC 2           1    Block2          UExt4_combine      FIRST 1
Block2          UExt4_combine       COMBINE 2           1    Block2          UExt4              FIRST 1
Block2          UExt4               USE UEXtru_Operator,10  FOR M_L_UPext*2/3 MIN
                                    GRAPHIC 5
                                    V_UExt4_Ready = 1
                                                        1    Chords          DExt4              LOAD 1                     GRAPHIC 3
Hens_Nest       DExt4               GRAPHIC 1
                                    WAIT UNTIL V_UExt4_Ready = 1
                                    USE DEXtru_Operator FOR M_L_Dext
                                    LOAD 1
                                    GRAPHIC 4
                                    WAIT M_R_UPext
```

```
                         GET DEXtru_Operator
                         WAIT M_U_Dext
                         V_Need_Nest4 = 1     1    Hens_Nest        CutterQ            FIRST 1                  GRAPHIC 5
                                                                                                                 V_UExt4_Ready = 0
                                                                                                                 V_UPext4_status = 0
                                                                                                                 MOVE WITH DEXtru_Operator THEN FREE
                                                                                                                 INC V_CUTTERQ
Chords           DExt4               GRAPHIC 2            1    Chords           CutterQ            FIRST 1
Blocker_Buggy    Elevator            WAIT M_Elevator_time
                                     V_Elevator_Status = 0
                                     FREE  UEXtru_Operator
                                     1    Blocker_Buggy    Empty_Block_Bug    FIRST 1
Blocker_Buggy    Empty_Block_Bug     GRAPHIC 6
                                     WAIT UNTIL V_Need_Bug_Blocker = 1
                                     V_Need_Bug_Blocker = 0
                                     1    Blocker_Buggy    Blocker_Trans      FIRST 1                  GRAPHIC 7
                                                                                                       MOVE WITH Blocker_Operator2,8 THEN FREE
SPC_Paperwork    SPC_Data_Entry1     GRAPHIC 1
                                     USE Cutter_Operator FOR  2 HR
                                     1    SPC_Paperwork    SPC_Data_Entry2    FIRST 1
                                          SPC_Paperwork    EXIT               FIRST
SPC_Paperwork    SPC_Data_Entry2     GRAPHIC 2
                                     WAIT 2 HR
                                     1    SPC_Paperwork    SPC_Data_Entry1    FIRST 1
Hens_Nest        CutterQ             IF A_Mix_Type = 4 THEN
                                     {
                                     GRAPHIC 7
                                     INC V_Ready_4_Changeover
                                      WAIT UNTIL V_Ready_4_Changeover = 5
                                      WAIT UNTIL V_Ready_4_Changeover = 0
                                     //WRITELINE(PROCESS_TIME,V_Process_Time,5,2)
                                     DEC V_CUTTERQ
                                     ROUTE 2
                                     }
                                     ELSE
                                     {
                                     GRAPHIC 6
                                     WAIT UNTIL V_CUT3_READY=0
                                       OR V_CUT1_READY=0
                                       OR V_CUT4_READY=0
                                       OR V_CUT7_READY=0
                                     DEC V_CUTTERQ
                                       ROUTE 1
                                     }                    1    Hens_Nest        Cutter3            IF V_CUT3_READY=0, 1     V_CUT3_READY=1
                                                                                                                 INC V_Can_flats_in_use
                                                                                                                 MOVE WITH Cutter_Operator,10
                                          Hens_Nest        Cutter1            IF V_CUT1_READY=0        V_CUT1_READY=1
                                                                                                                 INC V_Can_flats_in_use
                                                                                                                 MOVE WITH Cutter_Operator,10
                                          Hens_Nest        Cutter4            IF V_CUT4_READY=0        V_CUT4_READY=1
                                                                                                                 INC V_Can_flats_in_use
                                                                                                                 MOVE WITH Cutter_Operator,10
                                          Hens_Nest        Cutter7            IF V_CUT7_READY=0        V_CUT7_READY=1
                                                                                                                 INC V_Can_flats_in_use
                                                                                                                 MOVE WITH Cutter_Operator,10
                                     2    Hens_Nest        EXIT               FIRST 1
Chords           CutterQ             1    Chords           Cutter3            FIRST 1
                                          Chords           Cutter1            FIRST
                                          Chords           Cutter4            FIRST
                                          Chords           Cutter7            FIRST
Hens_Nest        Cutter3             WAIT M_L_cutter MIN
                                     GRAPHIC 5
                                     UNLOAD 1
                                     GRAPHIC 2
                                     WAIT UNTIL V_Cut3_Status=1
                                     1    Hens_Nest        Empty_Nests        FIRST 1                  GRAPHIC 3
                                                                                                       MOVE WITH  Cutter_Operator THEN FREE
                                                                                                       V_Cut3_Status=0
```

```
                                                Hens_Nest       EXIT              FIRST
Hens_Nest       Cutter1         WAIT M_L_cutter MIN
                                GRAPHIC 5
                                UNLOAD 1
                                GRAPHIC 2

                                WAIT UNTIL V_Cut1_Status=1
                                        1       Hens_Nest       Empty_Nests       FIRST 1         GRAPHIC 3
                                                                                                  MOVE WITH  Cutter_Operator THEN FREE
                                                                                                  V_Cut1_Status=0
                                                Hens_Nest       EXIT              FIRST
Hens_Nest       Cutter4         WAIT M_L_cutter MIN
                                GRAPHIC 5
                                UNLOAD 1
                                GRAPHIC 2
                                WAIT UNTIL V_Cut4_Status=1
                                        1       Hens_Nest       Empty_Nests       FIRST 1         GRAPHIC 3
                                                                                                  MOVE WITH  Cutter_Operator THEN FREE
                                                                                                  V_Cut4_Status=0
                                                Hens_Nest       EXIT              FIRST
Hens_Nest       Cutter7         WAIT M_L_cutter MIN
                                GRAPHIC 5
                                UNLOAD 1
                                GRAPHIC 2

                                WAIT UNTIL V_Cut7_Status=1
                                        1       Hens_Nest       Empty_Nests       FIRST 1         GRAPHIC 3
                                                                                                  MOVE WITH  Cutter_Operator THEN FREE
                                                                                                  V_Cut7_Status=0
                                                Hens_Nest       EXIT              FIRST
Hens_Nest       Empty_Nests     DEC V_Nests_in_use
                                GET DEXtru_Operator
                                        1       Hens_Nest       Hen_Nest          FIRST 1         MOVE WITH DEXtru_Operator
                                                                                                  IF V_Need_Nest2 = 0
                                                                                                  AND V_Need_Nest3 = 0
                                                                                                  AND V_Need_Nest4 = 0 THEN
                                                                                                  {
                                                                                                  FREE DEXtru_Operator
                                                                                                  }
Hens_Nest       Hen_Nest        GRAPHIC 1
                                WAIT UNTIL V_Need_Nest2 = 1
                                           OR V_Need_Nest3 = 1
                                           OR V_Need_Nest4 = 1
                                        1       Hens_Nest       DExt2             LU 1            V_Need_Nest2 = 0
                                                                                                  INC V_Nests_in_use
                                                                                                  MOVE WITH DEXtru_Operator  THEN FREE
                                                Hens_Nest       DExt3             LU              V_Need_Nest3 = 0
                                                                                                  INC V_Nests_in_use
                                                                                                  MOVE WITH DEXtru_Operator THEN FREE
                                                Hens_Nest       DExt4             LU              V_Need_Nest4 = 0
                                                                                                  INC V_Nests_in_use
                                                                                                  MOVE WITH DEXtru_Operator THEN FREE
Chords          Cutter3         GRAPHIC 2
                                WAIT M_R_cutter
                                V_Cut3_Status=1         1       Propellent        Can_Flat3       LOAD 1          INC V_TOT_CUT
Chords          Cutter1         GRAPHIC 2
                                WAIT M_R_cutter
                                V_Cut1_Status =1        1       Propellent        Can_Flat1       LOAD 1          INC V_TOT_CUT
Chords          Cutter4         GRAPHIC 2
                                WAIT M_R_cutter
                                V_Cut4_Status=1         1       Propellent        Can_Flat4       LOAD 1          INC V_TOT_CUT
Chords          Cutter7         GRAPHIC 2
                                WAIT M_R_cutter
                                V_Cut7_Status=1         1       Propellent        Can_Flat7       LOAD 1          INC V_TOT_CUT
Can_Flat        Can_Flat3       GRAPHIC 2
                                WAIT UNTIL V_CUT3_READY=1

                                LOAD 1
```

```
GRAPHIC 1
INC V_CANQ

/* CHECK TO SEE WHICH BADGER BUGGY TO MOVE TO AND TO SEE IF
  WORKER CAN MOVE FILLED CAN FLAT THERE TO EMPTY */

WAIT UNTIL V_N_Prop_Load1 < 9
  OR V_N_Prop_Load2 <  9

IF  V_N_Prop_Load1 >= 9
  AND   V_N_Prop_Load2 < 9THEN
{ V_WHICH_ROUTE = 2   }
ELSE
{ IF  V_N_Prop_Load1 <  9
           AND   V_N_Prop_Load2 >= 9THEN
  {        V_WHICH_ROUTE = 1   }
}

IF  V_WHICH_ROUTE = 1 THEN
{ IF  V_N_Prop_Load1 < 9 THEN
  {        IF V_Can_Flat_Loc1_Status = 0
                    OR V_Can_Flat_Loc3_Status = 0 THEN
           {         V_WHICH_ROUTE = 2
                     INC V_N_Prop_Load1
                     ROUTE 1   }
           ELSE
           {         IF V_N_Prop_Load2 < 9 THEN
                     {         IF V_Can_Flat_Loc2_Status = 0
                                         OR V_Can_Flat_Loc4_Status = 0 THEN
                               {         V_WHICH_ROUTE = 1
                                         INC V_N_Prop_Load2
                                         ROUTE     2         }
                               ELSE
                               {         V_WHICH_ROUTE = 2
                                         FREE ALL
                                         INC V_N_Prop_Load1
                                         WAIT UNTIL  V_Can_Flat_Loc1_Status = 0
                                                   OR V_Can_Flat_Loc3_Status = 0
                                         GET  Cutter_Operator
                                         ROUTE 1
                     }         }
                     ELSE
                     {         V_WHICH_ROUTE = 2
                               FREE ALL
                               INC V_N_Prop_Load1
                               WAIT UNTIL  V_Can_Flat_Loc1_Status = 0
                                         OR V_Can_Flat_Loc3_Status = 0
                               GET  Cutter_Operator
                               ROUTE 1
  }        }         }
  ELSE
  {        IF V_N_Prop_Load2 < 9 THEN
           {         IF V_Can_Flat_Loc2_Status = 0
                               OR V_Can_Flat_Loc4_Status = 0 THEN
                     {         V_WHICH_ROUTE = 1
                               INC V_N_Prop_Load2
                               ROUTE     2         }
                     ELSE
                     {         V_WHICH_ROUTE = 1
                               FREE ALL
                               INC V_N_Prop_Load2
                               WAIT UNTIL  V_Can_Flat_Loc2_Status = 0
                                         OR V_Can_Flat_Loc4_Status = 0
                               GET  Cutter_Operator
                               ROUTE 2
           }         }
           ELSE
           {         V_WHICH_ROUTE = 2
```

```
                    FREE ALL
                    WAIT UNTIL V_N_Prop_Load1 = 0
                    GET  Cutter_Operator
                    INC V_N_Prop_Load1
                    ROUTE 1
} }         }
ELSE
{ IF V_N_Prop_Load2 < 9 THEN
  {         IF V_Can_Flat_Loc2_Status = 0
                    OR V_Can_Flat_Loc4_Status = 0 THEN
            {         V_WHICH_ROUTE = 1
                      INC V_N_Prop_Load2
                      ROUTE     2         }
            ELSE
            {         IF  V_N_Prop_Load1 < 9 THEN
                      {         IF V_Can_Flat_Loc1_Status = 0
                                          OR V_Can_Flat_Loc3_Status = 0 THEN
                                {         V_WHICH_ROUTE = 2
                                          INC V_N_Prop_Load1
                                          ROUTE 1   }
                                ELSE
                                {         V_WHICH_ROUTE = 1
                                          FREE ALL
                                          INC V_N_Prop_Load2
                                          WAIT UNTIL  V_Can_Flat_Loc2_Status = 0
                                                    OR V_Can_Flat_Loc4_Status = 0
                                          GET  Cutter_Operator
                                          ROUTE 2
                      }         }
                      ELSE
                      {         V_WHICH_ROUTE = 1
                                FREE ALL
                                INC V_N_Prop_Load2
                                WAIT UNTIL  V_Can_Flat_Loc2_Status = 0
                                          OR V_Can_Flat_Loc4_Status = 0
                                GET  Cutter_Operator
                                ROUTE 2
  }         }         }
  ELSE
  {         IF  V_N_Prop_Load1 < 9 THEN
            {         IF V_Can_Flat_Loc1_Status = 0
                                OR V_Can_Flat_Loc3_Status = 0 THEN
                      {         V_WHICH_ROUTE = 2
                                INC V_N_Prop_Load1
                                ROUTE 1   }
                      ELSE
                      {         V_WHICH_ROUTE = 2
                                FREE ALL
                                INC V_N_Prop_Load1
                                WAIT UNTIL  V_Can_Flat_Loc1_Status = 0
                                                    OR V_Can_Flat_Loc2_Status = 0
                                GET  Cutter_Operator
                                ROUTE 1
            }         }
            ELSE
            {         V_WHICH_ROUTE = 1
                      FREE ALL
                      WAIT UNTIL V_N_Prop_Load2 = 0
                      GET  Cutter_Operator
                      INC V_N_Prop_Load2
                      ROUTE 2
} }         }               1    Can_Flat         Can_Flat_Loc1       FIRST 1                 INC V_BB_unload_in_use
                                                                                                  MOVE WITH Cutter_Operator,7
                                                                                                  V_Can_Flat_Loc1_Status = 1

                            Can_Flat         Can_Flat_Loc3       ALT                           INC V_BB_unload_in_use
                                                                                                  MOVE WITH Cutter_Operator,7
                                                                                                  V_Can_Flat_Loc3_Status = 1
```

```
                                        2    Can_Flat         Can_Flat_Loc2       FIRST 1                    INC V_BB_unload_in_use
                                                                                                             MOVE WITH Cutter_Operator,7
                                                                                                             V_Can_Flat_Loc2_Status = 1
                                             Can_Flat         Can_Flat_Loc4       ALT                        INC V_BB_unload_in_use
                                                                                                             MOVE WITH Cutter_Operator,7
                                                                                                             V_Can_Flat_Loc4_Status = 1
Can_Flat         Can_Flat1           GRAPHIC 2
                                     WAIT UNTIL V_CUT1_READY=1

                                     LOAD 1
                                     GRAPHIC 1
                                     INC V_CANQ

                                     /* CHECK TO SEE WHIC BADGER BUGGY TO MOVE TO AND TO SEE IF
                                       WORKER CAN MOVE FILLED CAN FLAT THERE TO EMPTY */

                                     WAIT UNTIL V_N_Prop_Load1 < 9
                                       OR V_N_Prop_Load2 <  9

                                     IF  V_N_Prop_Load1 >= 9
                                       AND   V_N_Prop_Load2 < 9THEN
                                     { V_WHICH_ROUTE = 2   }
                                     ELSE
                                     { IF  V_N_Prop_Load1 <  9
                                                AND   V_N_Prop_Load2 >= 9THEN
                                       {        V_WHICH_ROUTE = 1   }
                                     }

                                     IF  V_WHICH_ROUTE = 1 THEN
                                     { IF  V_N_Prop_Load1 < 9 THEN
                                       {        IF V_Can_Flat_Loc1_Status = 0
                                                          OR V_Can_Flat_Loc3_Status = 0 THEN
                                                {         V_WHICH_ROUTE = 2
                                                          INC V_N_Prop_Load1
                                                          ROUTE 1   }
                                                ELSE
                                                {         IF V_N_Prop_Load2 < 9 THEN
                                                          {         IF V_Can_Flat_Loc2_Status = 0
                                                                              OR V_Can_Flat_Loc4_Status = 0 THEN
                                                                    {         V_WHICH_ROUTE = 1
                                                                              INC V_N_Prop_Load2
                                                                              ROUTE     2         }
                                                                    ELSE
                                                                    {         V_WHICH_ROUTE = 2
                                                                              FREE ALL
                                                                              INC V_N_Prop_Load1
                                                                              WAIT UNTIL  V_Can_Flat_Loc1_Status = 0
                                                                                        OR V_Can_Flat_Loc3_Status = 0
                                                                              GET  Cutter_Operator
                                                                              ROUTE 1
                                                          }         }
                                                          ELSE
                                                          {         V_WHICH_ROUTE = 2
                                                                    FREE ALL
                                                                    INC V_N_Prop_Load1
                                                                    WAIT UNTIL  V_Can_Flat_Loc1_Status = 0
                                                                              OR V_Can_Flat_Loc3_Status = 0
                                                                    GET  Cutter_Operator
                                                                    ROUTE 1
                                       }        }         }
                                       ELSE
                                       {        IF V_N_Prop_Load2 < 9 THEN
                                                {         IF V_Can_Flat_Loc2_Status = 0
                                                                    OR V_Can_Flat_Loc4_Status = 0 THEN
                                                          {         V_WHICH_ROUTE = 1
                                                                    INC V_N_Prop_Load2
                                                                    ROUTE     2         }
```

```
                    ELSE
                    {         V_WHICH_ROUTE = 1
                              FREE ALL
                              INC V_N_Prop_Load2
                              WAIT UNTIL  V_Can_Flat_Loc2_Status = 0
                                        OR V_Can_Flat_Loc4_Status = 0
                              GET  Cutter_Operator
                              ROUTE 2
          }         }
          ELSE
          {         V_WHICH_ROUTE = 2
                    FREE ALL
                    WAIT UNTIL V_N_Prop_Load1 = 0
                    GET  Cutter_Operator
                    INC V_N_Prop_Load1
                    ROUTE 1
} }       }
ELSE
{ IF V_N_Prop_Load2 < 9 THEN
  {         IF V_Can_Flat_Loc2_Status = 0
                    OR V_Can_Flat_Loc4_Status = 0 THEN
            {         V_WHICH_ROUTE = 1
                      INC V_N_Prop_Load2
                      ROUTE     2         }
            ELSE
            {         IF  V_N_Prop_Load1 < 9 THEN
                      {         IF V_Can_Flat_Loc1_Status = 0
                                        OR V_Can_Flat_Loc3_Status = 0 THEN
                                {         V_WHICH_ROUTE = 2
                                          INC V_N_Prop_Load1
                                          ROUTE 1   }
                                ELSE
                                {         V_WHICH_ROUTE = 1
                                          FREE ALL
                                          INC V_N_Prop_Load2
                                          WAIT UNTIL  V_Can_Flat_Loc2_Status = 0
                                                    OR V_Can_Flat_Loc4_Status = 0
                                          GET  Cutter_Operator
                                          ROUTE 2
                      }         }
                      ELSE
                      {         V_WHICH_ROUTE = 1
                                FREE ALL
                                INC V_N_Prop_Load2
                                WAIT UNTIL  V_Can_Flat_Loc2_Status = 0
                                          OR V_Can_Flat_Loc4_Status = 0
                                GET  Cutter_Operator
                                ROUTE 2
  }         }         }
  ELSE
  {         IF  V_N_Prop_Load1 < 9 THEN
            {         IF V_Can_Flat_Loc1_Status = 0
                              OR V_Can_Flat_Loc3_Status = 0 THEN
                      {         V_WHICH_ROUTE = 2
                                INC V_N_Prop_Load1
                                ROUTE 1   }
                      ELSE
                      {         V_WHICH_ROUTE = 2
                                FREE ALL
                                INC V_N_Prop_Load1
                                WAIT UNTIL  V_Can_Flat_Loc1_Status = 0
                                                          OR V_Can_Flat_Loc2_Status = 0
                                GET  Cutter_Operator
                                ROUTE 1
            }         }
            ELSE
            {         V_WHICH_ROUTE = 1
                      FREE ALL
```

```
                                        WAIT UNTIL V_N_Prop_Load2 = 0
                                        GET  Cutter_Operator
                                        INC V_N_Prop_Load2
                                        ROUTE 2
                                } }         }               1    Can_Flat         Can_Flat_Loc1       FIRST 1                    INC V_BB_unload_in_use
                                                                                                                                MOVE WITH Cutter_Operator,7
                                                                                                                                V_Can_Flat_Loc1_Status = 1
                                                            Can_Flat         Can_Flat_Loc3       ALT                            INC V_BB_unload_in_use
                                                                                                                                MOVE WITH Cutter_Operator,7
                                                                                                                                V_Can_Flat_Loc3_Status = 1
                                                          2    Can_Flat         Can_Flat_Loc2       FIRST 1                    INC V_BB_unload_in_use
                                                                                                                                MOVE WITH Cutter_Operator,7
                                                                                                                                V_Can_Flat_Loc2_Status = 1
                                                            Can_Flat         Can_Flat_Loc4       ALT                            INC V_BB_unload_in_use
                                                                                                                                MOVE WITH Cutter_Operator,7
                                                                                                                                V_Can_Flat_Loc4_Status = 1
Can_Flat         Can_Flat4          GRAPHIC 6
                                    WAIT UNTIL V_CUT4_READY=1

                                    LOAD 1
                                    GRAPHIC 1
                                    INC V_CANQ

                                    /* CHECK TO SEE WHIC BADGER BUGGY TO MOVE TO AND TO SEE IF
                                      WORKER CAN MOVE FILLED CAN FLAT THERE TO EMPTY */

                                    WAIT UNTIL V_N_Prop_Load1 < 9
                                      OR V_N_Prop_Load2 <  9

                                    IF  V_N_Prop_Load1 >= 9
                                      AND   V_N_Prop_Load2 < 9THEN
                                    { V_WHICH_ROUTE = 2   }
                                    ELSE
                                    { IF  V_N_Prop_Load1 <  9
                                                AND   V_N_Prop_Load2 >= 9THEN
                                      {         V_WHICH_ROUTE = 1   }
                                    }

                                    IF  V_WHICH_ROUTE = 1 THEN
                                    { IF  V_N_Prop_Load1 < 9 THEN
                                      {         IF V_Can_Flat_Loc1_Status = 0
                                                        OR V_Can_Flat_Loc3_Status = 0 THEN
                                                {          V_WHICH_ROUTE = 2
                                                        INC V_N_Prop_Load1
                                                        ROUTE 1   }
                                                ELSE
                                                {         IF V_N_Prop_Load2 < 9 THEN
                                                          {         IF V_Can_Flat_Loc2_Status = 0
                                                                            OR V_Can_Flat_Loc4_Status = 0 THEN
                                                                    {         V_WHICH_ROUTE = 1
                                                                            INC V_N_Prop_Load2
                                                                            ROUTE     2         }
                                                                    ELSE
                                                                    {         V_WHICH_ROUTE = 2
                                                                            FREE ALL
                                                                            INC V_N_Prop_Load1
                                                                            WAIT UNTIL  V_Can_Flat_Loc1_Status = 0
                                                                                      OR V_Can_Flat_Loc3_Status = 0
                                                                            GET  Cutter_Operator
                                                                            ROUTE 1
                                                          }         }
                                                          ELSE
                                                          {         V_WHICH_ROUTE = 2
                                                                    FREE ALL
                                                                    INC V_N_Prop_Load1
                                                                    WAIT UNTIL  V_Can_Flat_Loc1_Status = 0
                                                                            OR V_Can_Flat_Loc3_Status = 0
                                                                    GET  Cutter_Operator
```

```
                                ROUTE 1
  }         }         }
  ELSE
  {         IF V_N_Prop_Load2 < 9 THEN
            {         IF V_Can_Flat_Loc2_Status = 0
                                OR V_Can_Flat_Loc4_Status = 0 THEN
                      {         V_WHICH_ROUTE = 1
                                INC V_N_Prop_Load2
                                ROUTE     2         }
                      ELSE
                      {         V_WHICH_ROUTE = 1
                                FREE ALL
                                INC V_N_Prop_Load2
                                WAIT UNTIL  V_Can_Flat_Loc2_Status = 0
                                          OR V_Can_Flat_Loc4_Status = 0
                                GET  Cutter_Operator
                                ROUTE 2
            }         }
            ELSE
            {         V_WHICH_ROUTE = 2
                      FREE ALL
                      WAIT UNTIL V_N_Prop_Load1 = 0
                      GET  Cutter_Operator
                      INC V_N_Prop_Load1
                      ROUTE 1
} }         }
ELSE
{ IF V_N_Prop_Load2 < 9 THEN
  {         IF V_Can_Flat_Loc2_Status = 0
                      OR V_Can_Flat_Loc4_Status = 0 THEN
            {         V_WHICH_ROUTE = 1
                      INC V_N_Prop_Load2
                      ROUTE     2         }
            ELSE
            {         IF  V_N_Prop_Load1 < 9 THEN
                      {         IF V_Can_Flat_Loc1_Status = 0
                                          OR V_Can_Flat_Loc3_Status = 0 THEN
                                {         V_WHICH_ROUTE = 2
                                          INC V_N_Prop_Load1
                                          ROUTE 1   }
                                ELSE
                                {         V_WHICH_ROUTE = 1
                                          FREE ALL
                                          INC V_N_Prop_Load2
                                          WAIT UNTIL  V_Can_Flat_Loc2_Status = 0
                                                    OR V_Can_Flat_Loc4_Status = 0
                                          GET  Cutter_Operator
                                          ROUTE 2
                      }         }
                      ELSE
                      {         V_WHICH_ROUTE = 1
                                FREE ALL
                                INC V_N_Prop_Load2
                                WAIT UNTIL  V_Can_Flat_Loc2_Status = 0
                                          OR V_Can_Flat_Loc4_Status = 0
                                GET  Cutter_Operator
                                ROUTE 2
  }         }         }
  ELSE
  {         IF  V_N_Prop_Load1 < 9 THEN
            {         IF V_Can_Flat_Loc1_Status = 0
                                OR V_Can_Flat_Loc3_Status = 0 THEN
                      {         V_WHICH_ROUTE = 2
                                INC V_N_Prop_Load1
                                ROUTE 1   }
                      ELSE
                      {         V_WHICH_ROUTE = 2
                                FREE ALL
```

```
                                        INC V_N_Prop_Load1
                                        WAIT UNTIL  V_Can_Flat_Loc1_Status = 0
                                                                OR V_Can_Flat_Loc2_Status = 0
                                        GET  Cutter_Operator
                                        ROUTE 1
                              }         }
                              ELSE
                              {         V_WHICH_ROUTE = 1
                                        FREE ALL
                                        WAIT UNTIL V_N_Prop_Load2 = 0
                                        GET  Cutter_Operator
                                        INC V_N_Prop_Load2
                                        ROUTE 2
                    } }       }               1    Can_Flat         Can_Flat_Loc1       FIRST 1                   INC V_BB_unload_in_use
                                                                                                                  MOVE WITH Cutter_Operator,7
                                                                                                                  V_Can_Flat_Loc1_Status = 1
                                        Can_Flat         Can_Flat_Loc3       ALT                               INC V_BB_unload_in_use
                                                                                                                  MOVE WITH Cutter_Operator,7
                                                                                                                  V_Can_Flat_Loc3_Status = 1
                                    2   Can_Flat         Can_Flat_Loc2       FIRST 1                           INC V_BB_unload_in_use
                                                                                                                  MOVE WITH Cutter_Operator,7
                                                                                                                  V_Can_Flat_Loc2_Status = 1
                                        Can_Flat         Can_Flat_Loc4       ALT                               INC V_BB_unload_in_use
                                                                                                                  MOVE WITH Cutter_Operator,7
                                                                                                                  V_Can_Flat_Loc4_Status = 1
Can_Flat        Can_Flat7       GRAPHIC 2
                                WAIT UNTIL V_CUT7_READY=1

                                LOAD 1
                                GRAPHIC 1
                                INC V_CANQ

                                /* CHECK TO SEE WHIC BADGER BUGGY TO MOVE TO AND TO SEE IF
                                  WORKER CAN MOVE FILLED CAN FLAT THERE TO EMPTY */

                                WAIT UNTIL V_N_Prop_Load1 < 9
                                  OR V_N_Prop_Load2 <  9

                                IF  V_N_Prop_Load1 >= 9
                                  AND   V_N_Prop_Load2 < 9THEN
                                { V_WHICH_ROUTE = 2   }
                                ELSE
                                { IF  V_N_Prop_Load1 <  9
                                          AND   V_N_Prop_Load2 >= 9THEN
                                  {         V_WHICH_ROUTE = 1   }
                                }

                                IF  V_WHICH_ROUTE = 1 THEN
                                { IF  V_N_Prop_Load1 < 9 THEN
                                  {         IF V_Can_Flat_Loc1_Status = 0
                                                    OR V_Can_Flat_Loc3_Status = 0 THEN
                                            {         V_WHICH_ROUTE = 2
                                                    INC V_N_Prop_Load1
                                                    ROUTE 1   }
                                            ELSE
                                            {         IF V_N_Prop_Load2 < 9 THEN
                                                      {         IF V_Can_Flat_Loc2_Status = 0
                                                                        OR V_Can_Flat_Loc4_Status = 0 THEN
                                                                {         V_WHICH_ROUTE = 1
                                                                        INC V_N_Prop_Load2
                                                                        ROUTE     2         }
                                                                ELSE
                                                                {         V_WHICH_ROUTE = 2
                                                                        FREE ALL
                                                                        INC V_N_Prop_Load1
                                                                        WAIT UNTIL  V_Can_Flat_Loc1_Status = 0
                                                                                  OR V_Can_Flat_Loc3_Status = 0
                                                                        GET  Cutter_Operator
```

```
                                        ROUTE 1
                        }       }
                        ELSE
                        {       V_WHICH_ROUTE = 2
                                FREE ALL
                                INC V_N_Prop_Load1
                                WAIT UNTIL  V_Can_Flat_Loc1_Status = 0
                                        OR V_Can_Flat_Loc3_Status = 0
                                GET  Cutter_Operator
                                ROUTE 1
  }         }       }
  ELSE
  {         IF V_N_Prop_Load2 < 9 THEN
            {       IF V_Can_Flat_Loc2_Status = 0
                                OR V_Can_Flat_Loc4_Status = 0 THEN
                    {       V_WHICH_ROUTE = 1
                            INC V_N_Prop_Load2
                            ROUTE     2         }
                    ELSE
                    {       V_WHICH_ROUTE = 1
                            FREE ALL
                            INC V_N_Prop_Load2
                            WAIT UNTIL  V_Can_Flat_Loc2_Status = 0
                                    OR V_Can_Flat_Loc4_Status = 0
                            GET  Cutter_Operator
                            ROUTE 2
            }       }
            ELSE
            {       V_WHICH_ROUTE = 2
                    FREE ALL
                    WAIT UNTIL V_N_Prop_Load1 = 0
                    GET  Cutter_Operator
                    INC V_N_Prop_Load1
                    ROUTE 1
} }         }
ELSE
{ IF V_N_Prop_Load2 < 9 THEN
  {         IF V_Can_Flat_Loc2_Status = 0
                    OR V_Can_Flat_Loc4_Status = 0 THEN
            {       V_WHICH_ROUTE = 1
                    INC V_N_Prop_Load2
                    ROUTE     2         }
            ELSE
            {       IF  V_N_Prop_Load1 < 9 THEN
                    {       IF V_Can_Flat_Loc1_Status = 0
                                    OR V_Can_Flat_Loc3_Status = 0 THEN
                            {       V_WHICH_ROUTE = 2
                                    INC V_N_Prop_Load1
                                    ROUTE 1   }
                            ELSE
                            {       V_WHICH_ROUTE = 1
                                    FREE ALL
                                    INC V_N_Prop_Load2
                                    WAIT UNTIL  V_Can_Flat_Loc2_Status = 0
                                            OR V_Can_Flat_Loc4_Status = 0
                                    GET  Cutter_Operator
                                    ROUTE 2
                    }       }
                    ELSE
                    {       V_WHICH_ROUTE = 1
                            FREE ALL
                            INC V_N_Prop_Load2
                            WAIT UNTIL  V_Can_Flat_Loc2_Status = 0
                                    OR V_Can_Flat_Loc4_Status = 0
                            GET  Cutter_Operator
                            ROUTE 2
  }         }       }
  ELSE
```

```
                                  {         IF  V_N_Prop_Load1 < 9 THEN
                                            {         IF V_Can_Flat_Loc1_Status = 0
                                                               OR V_Can_Flat_Loc3_Status = 0 THEN
                                                      {         V_WHICH_ROUTE = 2
                                                                INC V_N_Prop_Load1
                                                                ROUTE 1   }
                                                      ELSE
                                                      {         V_WHICH_ROUTE = 2
                                                                FREE ALL
                                                                INC V_N_Prop_Load1
                                                                WAIT UNTIL  V_Can_Flat_Loc1_Status = 0
                                                                                                    OR V_Can_Flat_Loc2_Status = 0
                                                                GET  Cutter_Operator
                                                                ROUTE 1
                                            }         }
                                            ELSE
                                            {         V_WHICH_ROUTE = 1
                                                      FREE ALL
                                                      WAIT UNTIL V_N_Prop_Load2 = 0
                                                      GET  Cutter_Operator
                                                      INC V_N_Prop_Load2
                                                      ROUTE 2
                                  } }     }                1    Can_Flat         Can_Flat_Loc1       FIRST 1                  INC V_BB_unload_in_use
                                                                                                                             MOVE WITH Cutter_Operator,7
                                                                                                                             V_Can_Flat_Loc1_Status = 1
                                                          Can_Flat         Can_Flat_Loc3       ALT                      INC V_BB_unload_in_use
                                                                                                                             MOVE WITH Cutter_Operator,7
                                                                                                                             V_Can_Flat_Loc3_Status = 1
                                                        2    Can_Flat         Can_Flat_Loc2       FIRST 1                  INC V_BB_unload_in_use
                                                                                                                             MOVE WITH Cutter_Operator,7
                                                                                                                             V_Can_Flat_Loc2_Status = 1
                                                          Can_Flat         Can_Flat_Loc4       ALT                      INC V_BB_unload_in_use
                                                                                                                             MOVE WITH Cutter_Operator,7
                                                                                                                             V_Can_Flat_Loc4_Status = 1
 Can_Flat         Can_Flat_Loc1       A_Can_Loc=1
                                      GRAPHIC 3
                                      DEC V_CANQ

                                      WAIT M_U_cutter MIN
                                      UNLOAD 1            1    Can_Flat         Empty_Can_Flats     FIRST 1                  GRAPHIC 7
                                                                                                                             MOVE WITH Cutter_Operator
                                                                                                                             V_Can_Flat_Loc1_Status = 0

                                                                                                                             /* CHECKING TO SEE IF WORKER IS NEED FOR
                                                                                                                              WEIGHING BADGER BUGGY */

                                                                                                                             IF V_N_Prop_Load1= 9 OR V_N_Prop_Load2 = 9 THEN
                                                                                                                             {          FREE Cutter_Operator          }
                                                                                                                             ELSE
                                                                                                                             {          IF V_CUT3_READY=0 AND V_CUT1_READY=0
                                                                                                                                        AND V_CUT4_READY=0 AND  V_CUT7_READY=0
THEN
                                                                                                                              {         FREE Cutter_Operator          }
                                                                                                                             }
 Can_Flat         Can_Flat_Loc3       A_Can_Loc=3
                                      GRAPHIC 3
                                      DEC V_CANQ

                                      WAIT M_U_cutter MIN
                                      UNLOAD 1            1    Can_Flat         Empty_Can_Flats     FIRST 1                  GRAPHIC 7

                                                                                                                             MOVE WITH Cutter_Operator
                                                                                                                             V_Can_Flat_Loc3_Status = 0

                                                                                                                             /* CHECKING TO SEE IF WORKER IS NEED FOR
                                                                                                                              WEIGHING BADGER BUGGY */

                                                                                                                             IF V_N_Prop_Load1= 9 OR V_N_Prop_Load2 = 9 THEN
```

```
                                                                                          {          FREE Cutter_Operator          }
                                                                                          ELSE
                                                                                          {          IF V_CUT3_READY=0 AND V_CUT1_READY=0
                                                                                                     AND V_CUT4_READY=0 AND  V_CUT7_READY=0
THEN
                                                                                           {         FREE Cutter_Operator          }
                                                                                          }
 Can_Flat     Can_Flat_Loc2    A_Can_Loc=2
                               GRAPHIC 3
                               DEC V_CANQ

                               WAIT M_U_cutter MIN
                               UNLOAD 1
                                          1   Can_Flat    Empty_Can_Flats   FIRST 1       GRAPHIC 7

                                                                                          MOVE WITH Cutter_Operator
                                                                                          V_Can_Flat_Loc2_Status = 0

                                                                                          /* CHECKING TO SEE IF WORKER IS NEED FOR
                                                                                           WEIGHING BADGER BUGGY */

                                                                                          IF V_N_Prop_Load1= 9 OR V_N_Prop_Load2 = 9 THEN
                                                                                          {          FREE Cutter_Operator          }
                                                                                          ELSE
                                                                                          {          IF V_CUT3_READY=0 AND V_CUT1_READY=0
                                                                                                     AND V_CUT4_READY=0 AND  V_CUT7_READY=0
THEN
                                                                                           {         FREE Cutter_Operator          }
                                                                                          }
 Can_Flat     Can_Flat_Loc4    A_Can_Loc=4
                               GRAPHIC 3
                               DEC V_CANQ


                               WAIT M_U_cutter MIN
                               UNLOAD 1   1   Can_Flat    Empty_Can_Flats   FIRST 1       GRAPHIC 7

                                                                                          MOVE WITH Cutter_Operator
                                                                                          V_Can_Flat_Loc4_Status = 0

                                                                                          /* CHECKING TO SEE IF WORKER IS NEED FOR
                                                                                           WEIGHING BADGER BUGGY */

                                                                                          IF V_N_Prop_Load1= 9 OR V_N_Prop_Load2 = 9 THEN
                                                                                          {          FREE Cutter_Operator          }
                                                                                          ELSE
                                                                                          {          IF V_CUT3_READY=0 AND V_CUT1_READY=0
                                                                                                     AND V_CUT4_READY=0 AND  V_CUT7_READY=0
THEN
                                                                                           {          FREE Cutter_Operator         }
                                                                                          }
 Can_Flat     Empty_Can_Flats  GRAPHIC 6
                               DEC V_Can_flats_in_use
                               DEC V_BB_unload_in_use
                                          1   Can_Flat    Can_Flat3         FIRST 1       MOVE WITH Cutter_Operator THEN FREE
                                                                                          V_CUT3_READY=0
                                              Can_Flat    Can_Flat1         FIRST         MOVE WITH Cutter_Operator THEN FREE
                                                                                          V_CUT1_READY=0
                                              Can_Flat    Can_Flat4         FIRST         MOVE WITH Cutter_Operator THEN FREE
                                                                                          V_CUT4_READY=0
                                              Can_Flat    Can_Flat7         FIRST         MOVE WITH Cutter_Operator THEN FREE
                                                                                          V_CUT7_READY=0
 Propellent   Can_Flat_Loc1    GRAPHIC 8  1   Propellent  Badger_Buggy1     LOAD 1        GRAPHIC 7
                                                                                          IF V_Cust_Num=A_Cust_Num THEN
                                                                                          {          V_Num_Mixes=V_Num_Mixes+(1/6) }
                                                                                          ELSE
                                                                                          {
                                                                                           IF V_Cust_Num2=A_Cust_Num THEN
```

```
          }


          }


          }



V_N_Prop_Load2+V_N_Prop_Load1)/9/60
```

```
 Propellent       Can_Flat_Loc3       GRAPHIC 8           1    Propellent       Badger_Buggy1       LOAD 1
```

```
          }


          }


          }
```

```
 {         V_Num_Mixes2=V_Num_Mixes2+(1/6)

ELSE
{
 IF V_Cust_Num3=A_Cust_Num THEN
 {         V_Num_Mixes3=V_Num_Mixes3+(1/6)

ELSE
{
 IF V_Cust_Num4=A_Cust_Num THEN
 {         V_Num_Mixes4=V_Num_Mixes4+(1/6)

}
}
}

IF V_Num_Mixes=V_Tot_Mixes THEN
{          V_Process_Time=CLOCK(HR)+M_W_cutter*(

 WRITELINE(PROCESS_TIME,V_Process_Time,5,2)
 V_Tardy_Time=V_Process_Time-A_Due_Date
 IF V_Tardy_Time>0 THEN
 {         V_Tot_Tardy=V_Tot_Tardy+V_Tardy_Time
           WRITELINE(TARDY_TIME,V_Tardy_Time,5,2)
           WRITELINE(TOT_TARDY,V_Tot_Tardy,5,2)
 }
 ELSE
 {         V_Tardy_Time=0
           WRITELINE(TARDY_TIME,V_Tardy_Time,5,2)
           WRITELINE(TOT_TARDY,V_Tot_Tardy,5,2)
 }
 V_Num_Mixes=V_Num_Mixes2
 V_Num_Mixes2 = V_Num_Mixes3
 V_Num_Mixes3=V_Num_Mixes4
 V_Num_Mixes4=0
  V_Cust_Num= V_Cust_Num2
  V_Cust_Num2= V_Cust_Num3
  V_Cust_Num3= V_Cust_Num4
  V_Cust_Num4= V_Cust_Num5
  V_Cust_Num5=0
 V_Tot_Mixes= V_Tot_Mixes2
 V_Tot_Mixes2=V_Tot_Mixes3
 V_Tot_Mixes3=V_Tot_Mixes4
 V_Tot_Mixes4= V_Tot_Mixes5
 V_Tot_Mixes5=0}
GRAPHIC 7
IF V_Cust_Num=A_Cust_Num THEN
{          V_Num_Mixes=V_Num_Mixes+(1/6) }
ELSE
{
 IF V_Cust_Num2=A_Cust_Num THEN
 {         V_Num_Mixes2=V_Num_Mixes2+(1/6)

ELSE
{
 IF V_Cust_Num3=A_Cust_Num THEN
 {         V_Num_Mixes3=V_Num_Mixes3+(1/6)

ELSE
{
 IF V_Cust_Num4=A_Cust_Num THEN
 {         V_Num_Mixes4=V_Num_Mixes4+(1/6)

}
}
}

IF V_Num_Mixes=V_Tot_Mixes THEN
```

```
V_N_Prop_Load2+V_N_Prop_Load1)/9/60
```

```
 Propellent       Can_Flat_Loc2       GRAPHIC 8           1    Propellent       Badger_Buggy2       LOAD 1
```

```
          }


          }


          }
```

```
V_N_Prop_Load2+V_N_Prop_Load1)/9/60
```

```
{          V_Process_Time=CLOCK(HR)+M_W_cutter*(

 WRITELINE(PROCESS_TIME,V_Process_Time,5,2)
 V_Tardy_Time=V_Process_Time-A_Due_Date
 IF V_Tardy_Time>0 THEN
 {          V_Tot_Tardy=V_Tot_Tardy+V_Tardy_Time
            WRITELINE(TARDY_TIME,V_Tardy_Time,5,2)
            WRITELINE(TOT_TARDY,V_Tot_Tardy,5,2)
 }
 ELSE
 {          V_Tardy_Time=0
            WRITELINE(TARDY_TIME,V_Tardy_Time,5,2)
            WRITELINE(TOT_TARDY,V_Tot_Tardy,5,2)
 }
 V_Num_Mixes=V_Num_Mixes2
 V_Num_Mixes2 = V_Num_Mixes3
 V_Num_Mixes3=V_Num_Mixes4
 V_Num_Mixes4=0
  V_Cust_Num= V_Cust_Num2
  V_Cust_Num2= V_Cust_Num3
  V_Cust_Num3= V_Cust_Num4
  V_Cust_Num4= V_Cust_Num5
  V_Cust_Num5=0
 V_Tot_Mixes= V_Tot_Mixes2
 V_Tot_Mixes2=V_Tot_Mixes3
 V_Tot_Mixes3=V_Tot_Mixes4
 V_Tot_Mixes4= V_Tot_Mixes5
 V_Tot_Mixes5=0}
GRAPHIC 7
IF V_Cust_Num=A_Cust_Num THEN
{          V_Num_Mixes=V_Num_Mixes+(1/6) }
ELSE
{
 IF V_Cust_Num2=A_Cust_Num THEN
 {          V_Num_Mixes2=V_Num_Mixes2+(1/6)

ELSE
{
 IF V_Cust_Num3=A_Cust_Num THEN
 {          V_Num_Mixes3=V_Num_Mixes3+(1/6)

ELSE
{
 IF V_Cust_Num4=A_Cust_Num THEN
 {          V_Num_Mixes4=V_Num_Mixes4+(1/6)

}
}
}

IF V_Num_Mixes=V_Tot_Mixes THEN
{          V_Process_Time=CLOCK(HR)+M_W_cutter*(

 WRITELINE(PROCESS_TIME,V_Process_Time,5,2)
 V_Tardy_Time=V_Process_Time-A_Due_Date
 IF V_Tardy_Time>0 THEN
 {          V_Tot_Tardy=V_Tot_Tardy+V_Tardy_Time
            WRITELINE(TARDY_TIME,V_Tardy_Time,5,2)
            WRITELINE(TOT_TARDY,V_Tot_Tardy,5,2)
 }
 ELSE
 {          V_Tardy_Time=0
            WRITELINE(TARDY_TIME,V_Tardy_Time,5,2)
            WRITELINE(TOT_TARDY,V_Tot_Tardy,5,2)
 }
 V_Num_Mixes=V_Num_Mixes2
 V_Num_Mixes2 = V_Num_Mixes3
 V_Num_Mixes3=V_Num_Mixes4
```

```
 Propellent       Can_Flat_Loc4       GRAPHIC 8           1    Propellent       Badger_Buggy2       LOAD 1


         }


         }


         }



V_N_Prop_Load2+V_N_Prop_Load1)/9/60
```

```
 Badger_Buggy     Badger_Buggy1       LOAD 1
                                      GRAPHIC 2
                                      LOAD 8
                                      INC V_BB_unload_in_use,2
                                      GET 2 Cutter_Operator,10
                                        WAIT UNTIL V_Can_Flat_Loc1_Status = 0
```

```
 V_Num_Mixes4=0
  V_Cust_Num= V_Cust_Num2
  V_Cust_Num2= V_Cust_Num3
  V_Cust_Num3= V_Cust_Num4
  V_Cust_Num4= V_Cust_Num5
  V_Cust_Num5=0
 V_Tot_Mixes= V_Tot_Mixes2
 V_Tot_Mixes2=V_Tot_Mixes3
 V_Tot_Mixes3=V_Tot_Mixes4
 V_Tot_Mixes4= V_Tot_Mixes5
 V_Tot_Mixes5=0}
GRAPHIC 7
IF V_Cust_Num=A_Cust_Num THEN
{          V_Num_Mixes=V_Num_Mixes+(1/6) }
ELSE
{
 IF V_Cust_Num2=A_Cust_Num THEN
 {         V_Num_Mixes2=V_Num_Mixes2+(1/6)

ELSE
{
 IF V_Cust_Num3=A_Cust_Num THEN
 {         V_Num_Mixes3=V_Num_Mixes3+(1/6)

ELSE
{
 IF V_Cust_Num4=A_Cust_Num THEN
 {         V_Num_Mixes4=V_Num_Mixes4+(1/6)

}
}
}

IF V_Num_Mixes=V_Tot_Mixes THEN
{          V_Process_Time=CLOCK(HR)+M_W_cutter*(

 WRITELINE(PROCESS_TIME,V_Process_Time,5,2)
 V_Tardy_Time=V_Process_Time-A_Due_Date
 IF V_Tardy_Time>0 THEN
 {         V_Tot_Tardy=V_Tot_Tardy+V_Tardy_Time
           WRITELINE(TARDY_TIME,V_Tardy_Time,5,2)
           WRITELINE(TOT_TARDY,V_Tot_Tardy,5,2)
 }
 ELSE
 {         V_Tardy_Time=0
           WRITELINE(TARDY_TIME,V_Tardy_Time,5,2)
           WRITELINE(TOT_TARDY,V_Tot_Tardy,5,2)
 }
 V_Num_Mixes=V_Num_Mixes2
 V_Num_Mixes2 = V_Num_Mixes3
 V_Num_Mixes3=V_Num_Mixes4
 V_Num_Mixes4=0
  V_Cust_Num= V_Cust_Num2
  V_Cust_Num2= V_Cust_Num3
  V_Cust_Num3= V_Cust_Num4
  V_Cust_Num4= V_Cust_Num5
  V_Cust_Num5=0
 V_Tot_Mixes= V_Tot_Mixes2
 V_Tot_Mixes2=V_Tot_Mixes3
 V_Tot_Mixes3=V_Tot_Mixes4
 V_Tot_Mixes4= V_Tot_Mixes5
 V_Tot_Mixes5=0}
```

```
                                        AND V_Can_Flat_Loc3_Status = 0

                              WAIT UNTIL V_Badger_Scale_Status = 0
                              V_Badger_Scale_Status = 1
                                        1    Badger_Buggy     Badger_Buggy_Scale  FIRST 1     INC V_TOT_BUGS
                                                                                              MOVE WITH  Cutter_Operator
                                             Badger_Buggy     EXIT                FIRST
Propellent       Badger_Buggy1             1    Propellent       Badger_Buggy_Scale  FIRST 1
Badger_Buggy     Badger_Buggy2        LOAD 1
                                      GRAPHIC 2
                                      LOAD 8
                                      INC V_BB_unload_in_use,2
                                      GET 2 Cutter_Operator,10
                                        WAIT UNTIL V_Can_Flat_Loc2_Status = 0
                                                    AND V_Can_Flat_Loc4_Status = 0

                              WAIT UNTIL V_Badger_Scale_Status = 0
                              V_Badger_Scale_Status = 1
                                        1    Badger_Buggy     Badger_Buggy_Scale  FIRST 1     INC V_TOT_BUGS
                                                                                              MOVE WITH  Cutter_Operator
                                             Badger_Buggy     EXIT                FIRST
Propellent       Badger_Buggy2             1    Propellent       Badger_Buggy_Scale  FIRST 1
Badger_Buggy     Badger_Buggy_Scale   WAIT M_W_cutter
                                      V_Badger_Scale_Status = 0
                                        1    Badger_Buggy     Loading_Dock        FIRST 1     MOVE WITH Cutter_Operator
                                                                                              FREE ALL
Propellent       Badger_Buggy_Scale        1    Propellent       Loading_Dock        FIRST 1
Badger_Buggy     Loading_Dock         INC V_TOT_N_BADGER_FILLED
                                      WAIT 5 MIN          1    Badger_Buggy     EXIT                FIRST 1
Propellent       Loading_Dock              1    Propellent       EXIT                FIRST 1
Badger_Buggy     Empty_Badger_Buggy        1    Badger_Buggy     Badger_Buggy1       EMPTY 1     MOVE WITH Cutter_Operator THEN FREE
                                                                                              V_N_Prop_Load1 = V_N_Prop_Load1 - 9
                                                                                              DEC V_BB_unload_in_use,2
                                             Badger_Buggy     Badger_Buggy2       EMPTY       MOVE WITH Cutter_Operator,9  THEN FREE
                                                                                              V_N_Prop_Load2 = V_N_Prop_Load2 - 9
                                                                                              DEC V_BB_unload_in_use,2
```

```
*****************************************************************************
*                                Arrivals                                   *
*****************************************************************************

  Entity        Location          Qty each   First Time Occurrences Frequency  Logic
  ------------- ----------------- ---------- ---------- ----------- ---------- ------------
  EntA          Job2              1          0          1           1
  Block         Jobs              1          0          V_N_JOBS-4  2 MIN      READ SCHEDULE,A_Cust_Num
                                                                              READ SCHEDULE,A_Mix_Type
                                                                              READ SCHEDULE,A_Due_Date
                                                                              READ SCHEDULE,A_Tot_Mixes

  Block         Jobs              1          0          4                      V_First_CO = 1
                                                                              READ SCHEDULE,A_Cust_Num
                                                                              READ SCHEDULE,A_Mix_Type
                                                                              READ SCHEDULE,A_Due_Date
                                                                              READ SCHEDULE,A_Tot_Mixes
  Dehy_Buggy    Dehy_T1a          1          0          1           1
  Dehy_Buggy    Dehy_T1b          1          0          1           1
  Dehy_Buggy    Empty_buggies     1          0          4           .5 MIN
  Dehy_Buggy    Empty_buggies1    2          0          1           1
  Mixer_Flat    Mixer_Trans1      2          0          1           1
  Mixer_Flat    Empty_Mixer_Flat  1          0          1           1
  Blocker_Buggy Blocker_Trans     2          0          1           1
  Blocker_Buggy Empty_Block_Bug   2          0          1           1
  Hens_Nest     Hen_Nest          5          0          1           1
  Hens_Nest     DExt2             1          0          1           1
  Hens_Nest     DExt3             1          0          1           1
  Hens_Nest     DExt4             1          0          1           1
  Can_Flat      Can_Flat1         1          0          1           1
```

```
  Can_Flat      Can_Flat3          1          0          1          1
  Can_Flat      Can_Flat4          1          0          1          1
  Can_Flat      Can_Flat7          1          0          1          1
  Can_Flat      Empty_Can_Flats    1          0          1          1
  Badger_Buggy  Badger_Buggy1      1          0          1          1
  Badger_Buggy  Badger_Buggy2      1          0          1          1
  Badger_Buggy  Empty_Badger_Buggy 6          0          INF        60 MIN
  Dehy_Buggy    Dehy_T2a           1          0          1          1
  Dehy_Buggy    Dehy_T2b           1          0          1          1
  Dehy_Buggy    Empty_buggies2     2          0          1          1
  SPC_Paperwork SPC_Data_Entry1    1          0          1          1


*****************************************************************************
*                                Attributes                                 *
*****************************************************************************

  ID              Type         Classification
  --------------- ------------ --------------
  A_Cust_Num      Integer      Entity
  A_Tot_Mixes     Integer      Entity
  A_Due_Date      Integer      Entity
#
#1=M14, 2=M831, 3=RP910, 4=Changeover M14, 5=Changeover RP910
  A_Mix_Type      Integer      Entity
#
#1 = press 1 2 = press 2
  A_Type          Integer      Entity
#
#0 = idle 1 = busy
  A_MIXER1        Integer      Entity
  A_MIXER2        Integer      Entity
  A_MIXER3        Integer      Entity
  A_MIXER4        Integer      Entity
#
## corresponds to bay that sample came from
  A_NIR_SAMPLE    Integer      Entity
  A_UPext1_status Integer      Location
  A_UPext2_status Integer      Location
  A_UPext3_status Integer      Location
  A_UPext4_status Integer      Location
#
#1,2,3,4
  A_Can_Loc       Integer      Entity


*****************************************************************************
*                              Variables (global)                           *
*****************************************************************************

  ID                     Type         Initial value Stats
  ---------------------- ------------ ------------- -----------
  V_num_processed        Integer      0             Time Series
#
#job tardiness
  V_Tardy_Time           Real         0             Time Series
#
#total tardy job time
  V_Tot_Tardy            Real         0             Time Series
  V_Num_Mixes4           Integer      0             Time Series
  V_Num_Mixes3           Integer      0             Time Series
  V_Num_Mixes2           Real         0             Time Series
  V_Num_Mixes            Real         0             Time Series
  V_Tot_Mixes5           Integer      0             Time Series
  V_Tot_Mixes4           Integer      0             Time Series
  V_Tot_Mixes3           Integer      0             Time Series
  V_Tot_Mixes2           Integer      0             Time Series
```

```
  V_Tot_Mixes            Integer      0             Time Series
  V_Cust_Num5            Integer      0             Time Series
  V_Cust_Num4            Integer      0             Time Series
  V_Cust_Num3            Integer      0             Time Series
  V_Cust_Num2            Integer      0             Time Series
  V_Cust_Num             Integer      0             Time Series
#
#process time for a product type
  V_Process_Time         Real         0             None
#
#indicates initial changeover
  V_First_CO             Integer      1             Time Series
#
#keeps track of the number of cotton assigned to each bay
  V_COUNT                Integer      -1            Time Series
#
#availabl e space for more  powder
  V_Tub1_Space           Integer      10            Time Series
#
#availabl e space for more  powder
  V_Tub2_Space           Integer      10            Time Series
#
#nomber of changeover carts loaded
  V_NCO_loaded3          Integer      0             Time Series
#
#nomber of changeover MIXER FLATS loaded
  V_NCO_loaded2          Integer      0             Time Series
#
#nomber of MIXERS ready for changeover
  V_NCO_loaded1          Integer      0             Time Series
#
#nomber of changeover carts loaded
  V_NCO_loaded           Integer      0             Time Series
#
## of areas ready for change over
  V_Ready_4_Changeover   Real         0             Time Series
#
## of customer orders
  V_N_JOBS               Integer      1300          Time Series
#
## of blocks produced from deyh1
  V_NBlocks1a            Integer      0             Time Series
#
## of blocks produced from deyh2
  V_NBlocks1b            Integer      0             Time Series
#
## of blocks produced from deyh3
  V_NBlocks2a            Integer      0             Time Series
#
## of blocks produced from deyh4
  V_NBlocks2b            Integer      0             Time Series
#
## of blocks produced from deyh5
  V_NBlocks3a            Integer      0             Time Series
#
## of blocks produced from deyh6
  V_NBlocks3b            Integer      0             Time Series
#
## of blocks produced from deyh7
  V_NBlocks4a            Integer      0             Time Series
#
## of blocks produced from deyh8
  V_NBlocks4b            Integer      0             Time Series
#
#indicates if dehy buggy is ready for loading
  V_Buggy_Ready1a        Integer      1             Time Series
#
#indicates if dehy buggy is ready for loading
```

```
  V_Buggy_Ready2a        Integer      1             Time Series
#
#indicates if dehy buggy is ready for loading
  V_Buggy_Ready2b        Integer      1             Time Series
#
#indicates if dehy buggy is ready for loading
  V_Buggy_Ready1b        Integer      1             Time Series
  V_Need_Mixtrans        Integer      0             Time Series
#
## of buggies in queue for mixers
  V_MixerQ               Integer      0             Time Series
#
#0 = idle 1 = busy; mixer status
  V_MIXER1               Integer      0             Time Series
#
#0 = idle 1 = busy; mixer status
  V_MIXER2               Integer      0             Time Series
#
#0 = idle 1 = busy; mixer status
  V_MIXER3               Integer      0             Time Series
#
#1 or 2; # mix unloaded
  V_NMix1                Integer      0             Time Series
#
#1 or 2; # mix unloaded
  V_NMix2                Integer      0             Time Series
#
#1 or 2; # mix unloaded
  V_NMix3                Integer      0             Time Series
#
#1 or 2; # mix unloaded
  V_NMix4                Integer      0             Time Series
#
#total # mixers idle
  V_Mixer_Idle           Integer      3             Time Series
#
#mixer1 unload area idle = 0 busy=1
  V_UL1_Idle             Integer      0             Time Series
#
#mixer2 unload area  idle = 0 busy=1
  V_UL2_Idle             Integer      0             Time Series
#
#mixer3 unload area  idle = 0 busy=1
  V_UL3_Idle             Integer      0             Time Series
#
#mixer4 unload area  idle = 0 busy=1
  V_UL4_Idle             Integer      0             Time Series
#
## of empty tubs
  V_Empty_mixer_flats    Integer      1             Time Series
#
## of times both presses have been loaded
  V_Number_blocks_1      Integer      0             Time Series
#
## of times both presses have been loaded
  V_Number_blocks_2      Integer      0             Time Series
#
## of times both presses have been loaded
  V_Number_blocks_3      Integer      0             Time Series
#
## of times both presses have been loaded
  V_Number_blocks_4      Integer      0             Time Series
#
#1 finish 0 testing
  V_Test_finish2         Integer      0             Time Series
#
#1 finish 0 testing
  V_Test_finish1         Integer      0             Time Series
```

```
#
## of samples ready for testing
  V_NIR_sample1          Integer      0             Time Series
#
## of samples ready for testing
  V_NIR_sample2          Integer      0             Time Series
  V_Empty_Buggies        Integer      4             Time Series
#
#1 = need empty buggy for dehy1 area
  V_Need_Bug_Dehy1       Integer      0             Time Series
#
#1 = need empty buggy for dehy2 area
  V_Need_Bug_Dehy2       Integer      0             Time Series
#
#1 = need empty tub for mixer 1
  V_Need_Tub_Mix1        Integer      0             Time Series
#
#1 = need empty tub for mixer 2
  V_Need_Tub_Mix2        Integer      0             Time Series
#
#1 = need empty tub for mixer 3
  V_Need_Tub_Mix3        Integer      0             Time Series
#
#1 = need empty tub for mixer 4
  V_Need_Tub_Mix4        Integer      0             Time Series
#
#1 = need empty tub for mix area
  V_Need_TUB_Mix         Integer      0             Time Series
  V_TUB_Avail            Integer      2             Time Series
#
#1 = need empty buggy for block house
  V_Need_Bug_Blocker     Integer      0             Time Series
#
## of buggies in queue for blockers
  V_BlockerQ             Integer      0             Time Series
#
#0 = idle 1 = busy
  V_Blocker_Status       Integer      0             Time Series
#
#1 = need empty hens nest
  V_Need_Nest2           Integer      0             Time Series
#
#1 = need empty hens nest
  V_Need_Nest3           Integer      0             Time Series
#
#1 = need empty hens nest
  V_Need_Nest4           Integer      0             Time Series
#
#upstairs extruder1 ready
  V_UExt1_Ready          Integer      0             Time Series
#
#upstairs extruder2 ready
  V_UExt2_Ready          Integer      0             Time Series
#
#upstairs extruder3 ready
  V_UExt3_Ready          Integer      0             Time Series
#
#upstairs extruder4 ready
  V_UExt4_Ready          Integer      0             Time Series
  V_UPext1_status        Integer      0             Time Series
  V_UPext2_status        Integer      0             Time Series
  V_UPext3_status        Integer      0             Time Series
  V_UPext4_status        Integer      0             Time Series
#
## of carts of propellent loaded into dehydrator1
  V_N_Prop_Load1         Integer      0             Time Series
#
## of carts of propellent loaded into dehydrator2
```

```
  V_N_Prop_Load2         Integer      0            Time Series
#
#0 = idle 1 = busy
  V_Elevator_Status      Integer      0            Time Series
#
#0 = not waiting 1 = waiting
  V_Wait_Dehy1           Integer      0            Time Series
#
#0 = not waiting 1 = waiting
  V_Wait_Dehy2           Integer      1            Time Series
#
#1st in line for empty mix buggy
  V_1_in_line_mix        Integer      0            Time Series
#
#2st in line for empty mix buggy
  V_2_in_line_mix        Integer      0            Time Series
#
#3st in line for empty mix buggy
  V_3_in_line_mix        Integer      0            Time Series
#
#4st in line for empty mix buggy
  V_4_in_line_mix        Integer      0            Time Series
  V_MIXER                Integer      0            Time Series
  V_1_in_line_dehy       Integer      0            Time Series
  V_2_in_line_dehy       Integer      0            Time Series
  V_3_in_line_dehy       Integer      0            Time Series
  V_4_in_line_dehy       Integer      0            Time Series
  V_5_in_line_dehy       Integer      0            Time Series
  V_6_in_line_dehy       Integer      0            Time Series
  V_7_in_line_dehy       Integer      0            Time Series
  V_8_in_line_dehy       Integer      0            Time Series
  V_9_in_line_dehy       Integer      0            Time Series
  V_10_in_line_dehy      Integer      0            Time Series
  V_11_in_line_dehy      Integer      0            Time Series
  V_12_in_line_dehy      Integer      0            Time Series
  V_13_in_line_dehy      Integer      0            Time Series
  V_14_in_line_dehy      Integer      0            Time Series
  V_15_in_line_dehy      Integer      0            Time Series
  V_16_in_line_dehy      Integer      0            Time Series
#
#0 = idle 1 = busy
  V_Badger_Scale_Status  Integer      0            Time Series
#
#0 = don't need 1= need
  V_Block_bug            Integer      0            Time Series
#
#0 = don't need to 1 = need to
  V_Ungroup_Dehy1_Buggy  Integer      0            Time Series
  V_Ungroup_Dehy2_Buggy  Integer      0            Time Series
#
#cutter 1 has empty can-flat ready
  V_CUT1_READY           Integer      0            Time Series
#
#cutter 3 has empty can-flat ready
  V_CUT3_READY           Integer      0            Time Series
#
#cutter 4 has empty can-flat ready
  V_CUT4_READY           Integer      0            Time Series
#
#cutter 7 has empty can-flat ready
  V_CUT7_READY           Integer      0            Time Series
  V_Cut1_Status          Integer      0            Time Series
  V_Cut3_Status          Integer      0            Time Series
  V_Cut4_Status          Integer      0            Time Series
  V_Cut7_Status          Integer      0            Time Series
#
## times block buggy has been unloaded
  V_UNLOADED_BLOCKS      Integer      0            Time Series
```

```
#
#status of whether or not propellent has finished unloading
  V_Can_Flat_Loc1_Status Integer      0             Time Series
#
#status of whether or not propellent has finished unloading
  V_Can_Flat_Loc2_Status Integer      0             Time Series
#
#status of whether or not propellent has finished unloading
  V_Can_Flat_Loc3_Status Integer      0             Time Series
#
#status of whether or not propellent has finished unloading
  V_Can_Flat_Loc4_Status Integer      0             Time Series
  V_TOT_CUT             Integer      0             Time Series
  V_TOT_BUGS            Integer      0             Time Series
#
#which badger buggy to unload into
  V_WHICH_ROUTE         Integer      1             Time Series
  V_TOT_N_BADGER_FILLED  Integer      0             Time Series
  V_ELEVATORQ           Integer      0             Time Series
  V_Nests_in_use        Integer      3             Time Series
  V_Can_flats_in_use    Integer      1             Time Series
  V_BB_unload_in_use    Integer      1             Time Series
  V_CUTTERQ             Integer      0             Time Series
  V_CANQ                Integer      0             Time Series
  V_NIRQ                Integer      0             Time Series


*****************************************************************************
*                                  Macros                                   *
*****************************************************************************

  ID                     Text
  ---------------------- ------------
  M_Change_P1            195 min
  M_Change_P3            243.75 min
  M_PL_dehy              1.69*(1./((1./U(0.5,0.5,1))-1.))**(1./4.84)                    // time to get two barrels for processing
  M_R_dehya              6+W(4.37, 1.5) MIN                 // dehy press processing time
  M_L_dehy               0.88*(1./((1./U(0.5,0.5,6))-1.))**(1./8.54) MIN                // time to load a press
  M_U_dehy               P5(3.93, 1.91,7)  MIN              // time to unload a press
  M_R_dehyb              6+W(4.37, 1.5) MIN                 // dehy press processing time
  M_T_dehy               T(2, 2.98, 3.63,12)// time to transport full dehy buggy to mixing area
  M_T_NIR                W(1.62, 2.82,13)                   // time to transport full dehy buggy to NIR area
  M_L_NIR                0.852*(1./((1./U(0.5,0.5,12))-1.))**(1./6.28) MIN              // time to pack 2 samples
  M_W_NIR                0.167*(1./((1./U(0.5,0.5,15))-1.))**(1./5.69)                  // time to walk from packing area to testing are and vice versa
  M_R_NIR                1+2.37*(1./((1./U(0.5,0.5,16))-1.))**(1./5.25)  MIN            // time to run the test for 2 samples
  M_D_NIR                P6(0.879, 353, 989,17) MIN                  // time to complete data entry
  M_U_NIR                0.457*(1./((1./U(0.5,0.5,18))-1.))**(1./4.84) MIN              // time to unload the samples
  M_L_mixer              W(3.59, 3.65,19)                   // time to load a mixer--includes travel time from mix queue area
  M_R_mixer_P3           50 MIN                   //mixer run time for RP910
  M_R_mixer_P2           28 MIN                   //mixer run time for M831
  M_R_mixer_P1           18 MIN                  // mixer run time for M865
  M_U_mixer              1+P5(3.9, 7.99,24) + P5(1.82, 1.61,24)/4 MIN                   // unload a mix from mixer
  M_T_mixer              T(2, 6.18, 8.37,25)                // time to transport full mixer flat to blocking area
  M_Position_blocker     G(1.25, 0.29,26)                   // time to position full mixer flat inbetween blockers
  M_L_blocker            W(5.61, 0.368,27) MIN              // time to load a blocker
  M_R_blocker            1+W(4.63, 0.282) MIN               // time to run both blockers
  M_U_blocker            P5(5.25, 1.01,28) MIN              // time to unload a blocker
  M_T_blocker            2+W(1.95, 1.11) // time to transport full blocker buggy to extruding area
  M_L_cutter             P5(5.92, 5.41) // time to load cutter
  M_R_cutter             1+0.831*(1./((1./U(0.5,0.5))-1.))**(1./5.39) MIN // time to run cutter
  M_T_cutter             G(6.74, 0.24) // time to transport cutter flat to badger buggy
  M_U_cutter             1.18*(1./((1./U(0.5,0.5))-1.))**(1./5.81)+IG(1.56, 1.55)//time to unload and clean cutters
  M_W_cutter             1+P6(4.22, 768, 243,35)// time to weigh badger buggy

  M_Elevator_time        .667 MIN               // time for elevator to trave up and vice versa
  M_L_Dext               G(9.79, 0.0757) MIN// time to postition hen's nest
  M_T_Dext               B(2.25, 1.95, 0, 0.83,38) // time to transport filled hen's nest to cutting area
  M_U_Dext               W(6.46, 0.593,39)  MIN// time to stop extruder
```

```
  M_R_UPext               2+W(3.2, 1.75) MIN// time to run the extruder (w/ delays)
  M_U_UPext               W(2.56, 1.2,42)  MIN // time to unload extruder
  M_L_UPext               1.04*(1./((1./U(0.5,0.5))-1.))**(1./5.48) // time to load extruder w/ 2 blocks


*****************************************************************************
*                              External Files                               *
*****************************************************************************

  ID           Type             File Name                                    Prompt
  ------------ ---------------- -------------------------------------------- ----------
  SCHEDULE     General Read     C:\My Documents\THESIS\MuseirEvans\schedule.txt
  PROCESS_TIME General Write    C:\My Documents\THESIS\Research\process.txt
  TARDY_TIME   General Write    C:\My Documents\THESIS\Research\jobtardy.txt
  TOT_TARDY    General Write    C:\My Documents\THESIS\Research\tottardy.txt


*****************************************************************************
*                                 Streams                                   *
*****************************************************************************

  Stream #     Seed #       Reset
  ------------ ------------ ------------
  1            13           No
  2            14           No
  3            15           No
  4            16           No
  5            17           No
  6            18           No
  7            19           No
  8            20           No
  9            21           No
  10           22           No
  11           23           No
  12           24           No
  13           25           No
  14           26           No
  15           27           No
  16           96           No
  17           95           No
  18           94           No
  19           93           No
  20           6            No
  21           5            No
  22           4            No
  23           3            No
  24           2            No
  25           1            No
  26           100          No
  27           99           No
  28           98           No
  29           97           No
  30           96           No
  32           95           No
  33           94           No
  34           93           No
  35           92           No
  36           91           No
  37           90           No
  38           89           No
  39           88           No
  40           87           No
  41           86           No
  42           85           No
  43           84           No
```