Transparency Masters for

Going Beyond An Introduction to Simulation Using GPSS/H

Unit K: Assembly Sets
Unit K
Assembly Sets

- The Concept of Assembly Sets

- The SPLIT Block (Bare Bones)

- Model K1
  A Model to Demonstrate
  Assembly Set id Numbers

- More About Assembly Sets

- The ASSEMBLE Block and Assembly Chains

- Model K2
  A Model for a Bottle-Labeling Operation
  in a Small Winery

- More About the ASSEMBLE Block

- Red Book Case Study 7D
  The Impact of Limited Labor-Pool Size
  on Completion Times in Precedence Networks

- The GATHER Block and Gather Chains

- Model K3
  A Model for a Corking-and-Labeling Operation
  in a Small Winery
Unit K
(continued)

- Synchronizing the Movement of Assembly-Set Members in Distinct Parts of a Model
- The MATCH Block and Matching Chains

  Model K4
  A Model to Display GPSS/H Matching-Chain Printout

- An Example of MATCH-Block Use in Context

  Model K5
  A Model to Display GPSS/H Assembly, Gather, and Matching Chain Printouts for Multiple Chains of Each Type

- More About the MATCH Block

- The Use of GATE Blocks with Assembly Sets

- The SPLIT Block's Optional Serialization Operand and Parameter Operands
The Concept of Assembly Sets

- Each Xact introduced into a model by a GENERATE Block has the potential of creating an Assembly Set.

- An Assembly Set is a collection of Xacts whose movements in a model can be coordinated and synchronized in certain ways through the use of these Blocks:

  ✓ ASSEMBLE
  ✓ GATHER
  ✓ MATCH
  ✓ GATE

- An Xact (which has been introduced into a model by a GENERATE Block) creates an Assembly Set the first time it executes a SPLIT Block.

- SPLIT-Block execution causes additional Xacts to be created.

- The Xact that executes a SPLIT Block can be thought of as a parent, and the Xacts created by the SPLIT-Block execution can be thought of as children or offspring or clones of the parent.

- The parent and its children are members of one and the same Assembly Set.
The Concept of Assembly Sets
(continued)

- Although the children are unique Xacts and have unique Xact id numbers, they are otherwise initially identical to the parent...
  ✓ same Priority Level
  ✓ same Mark Time (!!!)
  ✓ same numbers and types of Parameters
  ✓ same Parameter values

- As the SPLIT Block executes, and as each child is created, one by one ...
  ✓ it is merged immediately into the CEC (as last in its Priority Class, of course)
  ✓ its Current Block is the SPLIT Block
  ✓ its Next Block is set to the value of the SPLIT Block's B-Operand

- These aspects of the bare-bones SPLIT Block are summarized on the next frame
The Bare-Bones SPLIT Block

```
parent in

SPLIT

(B) child(ren) out

A

parent out
```

<table>
<thead>
<tr>
<th>Operand</th>
<th>Significance</th>
<th>Default or Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Split Count; Specifies the number of Xacts (children; offspring; clones) to be created*</td>
<td>Compile-Time Error</td>
</tr>
<tr>
<td>B</td>
<td>Next Block for the children</td>
<td>Compile-Time Error</td>
</tr>
</tbody>
</table>

*: the children are merged into the CEC (as last in their Priority Class, which is identical to that of their parent) at the time of their creation; they are on the CEC behind the point of the scan, so there is no need to turn on the model's Status Change Flag.
Bare-Bones SPLIT-Block Example

Parent In

SPLIT

(BYPASS)

Parent Out

1st Child Out

2nd Child Out

k=4
More About Assembly Sets

- A parent can create more children by executing SPLIT Blocks any number of additional times
  (this does not result in the creation of new Assembly Sets, however, but only increases the population of the already-existing Assembly Set)

- Furthermore, the children can create their own children by executing SPLIT Blocks one or more times, etc. etc.
  (this does not result in the creation of new Assembly Sets, but only increases the population of the Assembly Set to which they belong)

- An example of such possibilities is shown on the next frame
Creation and Propagation of Assembly Sets

**GENERATE**

\[ A, B \]

any meaningful collection of Blocks

parent

**SPLIT**

\[ (B) \] (1st generation, round 1)

child(ren)

parent

any meaningful collection of Blocks

parent

**SPLIT**

\[ (B) \]

child(ren)

(1st generation, round 2)

parent

any meaningful collection of Blocks

child(ren)

parent

**SPLIT**

\[ (B) \]

grandchild(ren)

(2nd generation, round 1)
More About Assembly Sets (continued)

- GPSS/H assigns numbers as Assembly-Set identifiers.

- An Assembly Set's id number is identical to the id number of the Xact which SPLIT to create the Assembly Set in the first place.

- When the attributes of an Xact are printed out (as in "display xact 25" or "display cec" or "PRINT ,,MOV"), the number of the Assembly Set to which an Xact belongs (if any) is shown in the **ASMSET** column.

  (sample Xact printout is shown on Frame K-9)

- The **ASMSET** column is **blank** if the Xact is not a member of an Assembly Set.

- The Standard Numerical Attribute **AN1** takes as its value the **size** of (number of Xacts in) the Assembly Set to which the moving Xact belongs.

  (if the moving Xact does not belong to an Assembly Set, the value of AN1 is 0)

- AN1 is the only SNA provided for Assembly Sets.
*************** Model K1 ***************

SIMULATE A Model to Demonstrate *

Assembly Set id Numbers Displayed in Xact and Chain Printouts *

In this model, the Xact with id #3 creates a child; *
the child then creates its own child *
(grandchild of the original); *
and the child's child then creates a child of its own *
(great grandchild of the original); *
then the four generations are displayed on the CEC *

******************************************************************************

Model Segment 1 (Segment to Use Up Xact id's #1 and #2) *
******************************************************************************

GENERATE 0,,1 initialization Xact has id #1

TERMINATE

GENERATE 0,,1 initialization Xact has id #2

TERMINATE

******************************************************************************

Model Segment 2 (Segment to Provide Xact with id #3, etc.) *
******************************************************************************

GENERATE 10,,1 initialization Xact has id #3

PRINT ,,C display the time
PRINT ,,,MOV Xact #3 displays itself on the CEC

SPLIT 1,CHILD create a 1st generation Xact
GATE FU JOE the parent finds itself blocked

CHILD ADVANCE 10 the child pauses for 10 time units,
SPLIT 1,GCHILD then creates a 2nd generation Xact
GATE FU JOE the child finds itself blocked

GCHILD ADVANCE 10 the grandchild pauses 10 time units,
SPLIT 1,GGCHILD then creates a 3rd generation Xact
GATE FU JOE the grandchild finds itself blocked

GGCHILD ADVANCE 10 the great grandchild pauses,
PRINT ,,C then displays the time
PRINT ,,,MOV and displays the 4 generations
on the Current Events Chain
* and finally stops Xact movement

******************************************************************************

Run-Control Statements *
******************************************************************************

START 1 start the simulation

END end of Model-File execution

K-8
<table>
<thead>
<tr>
<th>XACT</th>
<th>CURBLK</th>
<th>NXTBLK</th>
<th>CHAINS</th>
<th>SDPGFT**</th>
<th>MARK-TIME</th>
<th>MOVE-TIME</th>
<th>PRIORITY</th>
<th>PC</th>
<th>ASMSET</th>
<th>ADDR</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>6</td>
<td>7</td>
<td>CEC</td>
<td></td>
<td>10.0000</td>
<td>10.0000</td>
<td>0</td>
<td></td>
<td></td>
<td>00946598</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>XACT</th>
<th>CURBLK</th>
<th>NXTBLK</th>
<th>CHAINS</th>
<th>SDPGFT**</th>
<th>MARK-TIME</th>
<th>MOVE-TIME</th>
<th>PRIORITY</th>
<th>PC</th>
<th>ASMSET</th>
<th>ADDR</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>8</td>
<td>9</td>
<td>CEC</td>
<td>SD</td>
<td>10.0000</td>
<td>10.0000</td>
<td>0</td>
<td>3</td>
<td></td>
<td>00946598</td>
</tr>
<tr>
<td>4</td>
<td>11</td>
<td>12</td>
<td>CEC</td>
<td>SD</td>
<td>10.0000</td>
<td>20.0000</td>
<td>0</td>
<td>3</td>
<td></td>
<td>009465F8</td>
</tr>
<tr>
<td>5</td>
<td>14</td>
<td>15</td>
<td>CEC</td>
<td>SD</td>
<td>10.0000</td>
<td>30.0000</td>
<td>0</td>
<td>3</td>
<td></td>
<td>009466A0</td>
</tr>
<tr>
<td>6</td>
<td>17</td>
<td>18</td>
<td>CEC</td>
<td></td>
<td>10.0000</td>
<td>40.0000</td>
<td>0</td>
<td>3</td>
<td></td>
<td>00946490</td>
</tr>
</tbody>
</table>
A Short Assembly-Set Exercise

- Xacts are to be introduced into a model with exponentially distributed interarrival times having an expected value of 18.5 time units.

  It is required that all such Xacts be members of one and the same Assembly Set.

  Show how to model this situation.

- Here is how **not** to model the situation:

  ![Diagram](attachment:image.png)

- Here is how the situation can be modeled:

  ![Diagram](attachment:image.png)
The ASSEMBLE Block

- TERMINATE Blocks can be used to destroy Xacts which are members of an Assembly Set

- Another Block, the ASSEMBLE Block, can also be used to destroy Xacts which are Assembly Set members

![Diagram showing the process of assembly]

<table>
<thead>
<tr>
<th>Operand</th>
<th>Significance</th>
<th>Default Value or Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Assembly Count; one greater than the number of members of the initiator's Assembly Set which are to be destroyed in the assembly process</td>
<td>Compile-Time Error</td>
</tr>
</tbody>
</table>
When an assembly operation has been completed, there is no moving Xact and a newcomer Xact has been placed on the CEC, so the scan of the CEC is restarted immediately.
Assembly Set/ASSEMBLE Block
Considerations

- For a given ASSEMBLE Block, independent assembly operations for two or more Assembly Sets can be in process concurrently.

- For a given Assembly Set, independent assembly operations at two or more ASSEMBLE Blocks can be in process concurrently.

- Here is an example of various independent Assembly Set/ASSEMBLE Block assembly operations that might be ongoing or not in a given model concurrently:

<table>
<thead>
<tr>
<th>ASMSET #s</th>
<th>Location A</th>
<th>Location B</th>
<th>Location C</th>
<th>etc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>#7</td>
<td>Not Ongoing</td>
<td>Ongoing</td>
<td>Not Ongoing</td>
<td></td>
</tr>
<tr>
<td>#12</td>
<td>Ongoing</td>
<td>Ongoing</td>
<td>Not Ongoing</td>
<td></td>
</tr>
<tr>
<td>#19</td>
<td>Ongoing</td>
<td>Ongoing</td>
<td>Ongoing</td>
<td></td>
</tr>
<tr>
<td>etc.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- After an assembly operation has been completed for a given Assembly Set/ASSEMBLE Block combination, another assembly operation can begin for that Assembly Set at that ASSEMBLE Block.
Some Possible Schemes for SPLIT-ASSEMBLE Use

ANY BLOCK DIAGRAM SEGMENT

SPLIT
9

(NEXT)

ASSEMBLE

SPLIT
10

(BYPAS)

ANY BLOCK DIAGRAM SEGMENT

(Recommended approach if the parent controls one or more facilities at the time of the split...because an error occurs if a XACT tries to release a facility controlled by any other XACT.)
A Model for a Bottle-Labeling Operation in a Small Winery

GENERATE

SEIZE

SPLIT

ADVANCE

RELEASE

ASSEMBLE

TABULATE

TERMINATE

SEED THE SEGMENT

WORKER TAKES NEXT UNLABELED BOTTLE

PROVIDE ANOTHER UNLABELED BOTTLE

PUT A LABEL ON THE BOTTLE

THIS BOTTLE HAS BEEN LABELED

BOTTLE GOES INTO A CASE (12 TO THE CASE)

RECORD THE INTERARRIVAL TIME OF FULL CASES

CASE LEAVES LABELING AREA
SIMULATE A Model for Labeling

Bottles of Wine by Hand in a Small Winery

Basis: Page 464 in the 1974 Red Book

Table Control Statements

IATIME TABLE IA,178,7,8 interarrival time (filled cases)

Model Segment 1 (The Labeling of Wine Bottles)

GENERATE 0,,1 seed the segment
BACK SEIZE WORKER worker takes next unlabeled bottle
ADVANCE 16,3 put a label on the bottle
RELEASE WORKER this bottle has been labeled
SPLIT 1,BACK provide another unlabeled bottle
ASSEMBLE 12 bottle goes into a case
(12 per case)

TABULATE IATIME record the interarrival time
of full cases

TERMINATE case leaves the labeling area

Model Segment 2 (Run-control Transaction)

GENERATE 28800 28800 = 8 hours expressed in seconds
TERMINATE 1 stop simulating after 8 hours

Run Control Statements

START 1,,1 simulate; print chains at the end

END end of model-file processing
RELATIVE CLOCK: 28800.0000  ABSOLUTE CLOCK: 28800.0000

BLOCK CURRENT  TOTAL
1  1
BACK  1797
3  1  1797
4  1796
5  3592
6  1  1796
7  149
8  149
9  1
10  1

ASSEMBLY SET 1

ASSEMBLY IN PROGRESS AT BLOCK 6  REMAINING COUNT = 4

<table>
<thead>
<tr>
<th>XACT</th>
<th>CURBLK</th>
<th>NXTBLK</th>
<th>CHAINS</th>
<th>SDPGET**</th>
<th>MARK-TIME</th>
<th>MOVE-TIME</th>
<th>PRIORITY</th>
<th>PC</th>
<th>ASMSET</th>
<th>ADDR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1790</td>
<td>6</td>
<td>7</td>
<td>ASM</td>
<td>0.0</td>
<td>28675.4443</td>
<td>0</td>
<td>1</td>
<td></td>
<td>009464B0</td>
<td></td>
</tr>
</tbody>
</table>

TABLE  IATIME

ENTRIES IN TABLE  MEAN ARGUMENT  STANDARD DEVIATION  SUM OF ARGUMENTS  NON-WEIGHTED
148.0000  192.3976  6.2259  28474.3406

<table>
<thead>
<tr>
<th>UPPER LIMIT</th>
<th>OBSERVED FREQ</th>
<th>PERCENT OF TOTAL</th>
<th>CUMULATIVE PERCENTAGE</th>
<th>CUMULATIVE REMAINDER</th>
<th>MULTIPLE OF MEAN</th>
<th>DEVIATION FROM MEAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>178.0000</td>
<td>1.0000</td>
<td>0.68</td>
<td>0.68</td>
<td>99.32</td>
<td>0.9252</td>
<td>-2.3125</td>
</tr>
<tr>
<td>185.0000</td>
<td>14.0000</td>
<td>9.46</td>
<td>10.14</td>
<td>89.86</td>
<td>0.9616</td>
<td>-1.1882</td>
</tr>
<tr>
<td>192.0000</td>
<td>54.0000</td>
<td>36.49</td>
<td>46.62</td>
<td>53.38</td>
<td>0.9979</td>
<td>-0.0639</td>
</tr>
<tr>
<td>199.0000</td>
<td>59.0000</td>
<td>39.86</td>
<td>86.49</td>
<td>13.51</td>
<td>1.0343</td>
<td>1.0605</td>
</tr>
<tr>
<td>206.0000</td>
<td>19.0000</td>
<td>12.84</td>
<td>99.32</td>
<td>0.68</td>
<td>1.0707</td>
<td>2.1848</td>
</tr>
<tr>
<td>213.0000</td>
<td>1.0000</td>
<td>0.68</td>
<td>100.00</td>
<td>0.00</td>
<td>1.1071</td>
<td>3.3092</td>
</tr>
</tbody>
</table>

...
More About the ASSEMBLE Block

- In general, the Assembly Count is the value of an arithmetic expression, and so can conceivably take on values of 1 or 0 as well as values greater than 1.

- If an Xact tries to initiate an assembly operation and the Assembly Count is 1, the Xact remains on the CEC and continues to be the moving Xact.

- If an Xact tries to initiate an assembly operation and the Assembly Count is 0 or less, an execution error occurs.

- If the last surviving member of an Assembly Set tries to initiate an assembly operation and the Assembly Count exceeds 1, then an execution error occurs:

  Error 467: entering Xact is the only member of its Assembly Set

- If the last surviving member of an Assembly Set is destroyed at an ASSEMBLE Block without completing the ongoing assembly operation, no execution error results and no warning is issued.
  (this is despite the fact that the initiator of that assembly operation is doomed to remain on the Assembly Chain "forever")
The Impact of Limited Labor-Pool Size on Completion Times in Precedence Networks
(Case Study 7D from the Red Book)
MODEL SEGMENT 1
**Red Book Case Study 7D**

**SIMULATE**

The Impact of Limited Labor-Pool Size on Completion Times in Precedence Networks

Time Unit: Implicit in the Problem Statement

**Compiler Directives (INTEGER)**

**INTEGER** &POOLSIZE

**Control Statements (TABLE)**

...distribution of workers in use (a weighted Table)... INUSE TABLE S(WORKERS),0,1,W20

...project completion times... PCTIMES TABLE M1,25,25,20

**Model Segment 1 (Logic of the Precedence Network)**

**GENERATE** 0 provide an Xact whenever needed
**GATE** LR DONEXT1 wait for end of preceding iteration
**LOGIC** S DONEXT1 close GATE on successor Xact for now

**NODE1**
**SPLIT** 1, SUB13 send child to handle subproject 1-to-3
**SUB12**
**ENTER** WORKERS, 4 get workers for subproject 1-to-2
**ADVANCE** 14, 6 do the subproject
**LEAVE** WORKERS, 4 free the workers

**NODE2**
**SPLIT** 1, SUB24 send child to handle subproject 2-to-4
**SUB25**
**ENTER** WORKERS, 5 get workers for subproject 2-to-5
**ADVANCE** 18, 4 do the subproject
**LEAVE** WORKERS, 5 free the workers

**NODE5**
**ASSEMBLE** 2 wait for 2-to-5 and 4-to-5 to complete
**SUB57**
**ENTER** WORKERS, 2 get workers for subproject 5-to-7
**ADVANCE** 8, 3 do the subproject
**LEAVE** WORKERS, 2 free the workers
**TRANSFER** , NODE7 go signal completion of 5-to-7

**SUB24**
**ENTER** WORKERS, 3 get workers for subproject 2-to-4
**ADVANCE** 10, 3 do the subproject
**LEAVE** WORKERS, 3 free the workers

**NODE4**
**ASSEMBLE** 2 wait for 2-to-4 and 3-to-4 to complete
**SPLIT** 1, NODE5 send child to signal node 4 completion
**SUB47**
**ENTER** WORKERS, 4 get workers for subproject 4-to-7
**ADVANCE** 15, 5 do the subproject
**LEAVE** WORKERS, 4 free the workers
* NODE7 ASSEMBLE 3 wait for final subprojects to complete
TABULATE PCTIMES tabulate the project completion time
LOGIC R DONEXT1 open GATE for the next iteration
TERMINATE 1 current iteration is finished
*
* SUB13 ENTER WORKERS,3 get workers for subproject 1-to-3
ADVANCE 20,9 do the subproject
LEAVE WORKERS,3 free the workers
*
* NODE3 SPLIT 1,SUB34 send child to handle subproject 3-to-4
SUB36 ENTER WORKERS get a worker for subproject 3-to-6
ADVANCE 25,7 do the subproject
LEAVE WORKERS free the worker
*
* NODE6 ENTER WORKERS,4 get workers for subproject 6-to-7
ADVANCE 10,3 do the subproject
LEAVE WORKERS,4 free the workers
TRANSFER ,NODE7 go signal completion of 3-to-7
*
* SUB34 ENTER WORKERS,2 get workers for subproject 3-to-4
ADVANCE 22,5 do the subproject
LEAVE WORKERS,2 free the workers
TRANSFER ,NODE4 go signal completion of 3-to-4
*

**********************************************************************
* Model Segment 2 (Xact to Make Weighted-Table Recordings) *
**********************************************************************
*
GENERATE 0,,1,5,1PL bring in one high-priority Xact
CYCLE ASSIGN TIME,AC1,PL note the current time
TEST G AC1,PL(TIME) wait until after next clock update
TABULATE INUSE, \[AC1-PL(TIME)\] record current number of workers
* in use, weighted by the duration of that use
TRANSFER ,CYCLE play it again, Sam
*
**********************************************************************
* Run-Control Statements *
**********************************************************************
*
DO &POOLSIZE=5,14 loop on from 5 to 14 workers
*
WORKERS STORAGE &POOLSIZE operationalize current worker pool
* size in terms of Storage capacity
START 250 replicate 250 times
CLEAR clear for next pool size
ENDDO call for the next pool size
*
END that's all she wrote

K-23
<table>
<thead>
<tr>
<th>CLASS</th>
<th>MAXIMUM CONTENTS</th>
<th>CURRENT CONTENTS</th>
<th>AVERAGE CONTENTS</th>
<th>CAPACITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>RELATIVE CLOCK: 14670.2995</td>
<td>11</td>
<td>7.195</td>
<td>7.195</td>
<td>11</td>
</tr>
</tbody>
</table>

---

### Table 1: Entries in Table

<table>
<thead>
<tr>
<th>TABLE Entries</th>
<th>Mean Argument</th>
<th>Mean Argument</th>
<th>Standard Deviation</th>
<th>Standard Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>250,000</td>
<td>7.3400</td>
<td>7.1949</td>
<td>2.5711</td>
<td>2.6571</td>
</tr>
</tbody>
</table>

### Table 2: Non-Weighted Cumulative Deviation

<table>
<thead>
<tr>
<th>TABLE Entries</th>
<th>Mean Argument</th>
<th>Mean Argument</th>
<th>Standard Deviation</th>
<th>Standard Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>250,000</td>
<td>7.3400</td>
<td>7.1949</td>
<td>2.5711</td>
<td>2.6571</td>
</tr>
</tbody>
</table>

---

### Table 3: Upper Observed Limit of Total Percentage Remainder

<table>
<thead>
<tr>
<th>TABLE Entries</th>
<th>Mean Argument</th>
<th>Mean Argument</th>
<th>Standard Deviation</th>
<th>Standard Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>250,000</td>
<td>7.3400</td>
<td>7.1949</td>
<td>2.5711</td>
<td>2.6571</td>
</tr>
</tbody>
</table>

---

### Table 4: Non-Weighted Cumulative Deviation

<table>
<thead>
<tr>
<th>TABLE Entries</th>
<th>Mean Argument</th>
<th>Mean Argument</th>
<th>Standard Deviation</th>
<th>Standard Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>250,000</td>
<td>7.3400</td>
<td>7.1949</td>
<td>2.5711</td>
<td>2.6571</td>
</tr>
</tbody>
</table>

---

### Table 5: Upper Observed Limit of Total Percentage Remainder

<table>
<thead>
<tr>
<th>TABLE Entries</th>
<th>Mean Argument</th>
<th>Mean Argument</th>
<th>Standard Deviation</th>
<th>Standard Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>250,000</td>
<td>7.3400</td>
<td>7.1949</td>
<td>2.5711</td>
<td>2.6571</td>
</tr>
</tbody>
</table>

---

### Table 6: Non-Weighted Cumulative Deviation

<table>
<thead>
<tr>
<th>TABLE Entries</th>
<th>Mean Argument</th>
<th>Mean Argument</th>
<th>Standard Deviation</th>
<th>Standard Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>250,000</td>
<td>7.3400</td>
<td>7.1949</td>
<td>2.5711</td>
<td>2.6571</td>
</tr>
</tbody>
</table>

---

### Table 7: Upper Observed Limit of Total Percentage Remainder

<table>
<thead>
<tr>
<th>TABLE Entries</th>
<th>Mean Argument</th>
<th>Mean Argument</th>
<th>Standard Deviation</th>
<th>Standard Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>250,000</td>
<td>7.3400</td>
<td>7.1949</td>
<td>2.5711</td>
<td>2.6571</td>
</tr>
</tbody>
</table>
# Summary of Results for Red Book Case 7D

## Project Completion Times When Subproject Times Are Probabilistic

(Basis: 250 Iterations)

<table>
<thead>
<tr>
<th>Size of Labor Pool</th>
<th>Project Completion Times</th>
<th>Number of Busy Workers on Average</th>
<th>Utilization of Labor Pool</th>
<th>Maximum Number of Workers in Use Concurrently</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>117.5 8.9</td>
<td>3.6</td>
<td>0.720</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>98.1 8.0</td>
<td>4.3</td>
<td>0.715</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>81.9 6.2</td>
<td>5.1</td>
<td>0.734</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>62.9 4.8</td>
<td>6.7</td>
<td>0.840</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>66.6 5.4</td>
<td>9.7</td>
<td>0.702</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>65.0 5.6</td>
<td>6.5</td>
<td>0.649</td>
<td>10</td>
</tr>
<tr>
<td>11</td>
<td>58.5 6.3</td>
<td>7.2</td>
<td>0.655</td>
<td>11</td>
</tr>
<tr>
<td>12</td>
<td>59.1 5.8</td>
<td>7.2</td>
<td>0.596</td>
<td>11</td>
</tr>
<tr>
<td>13</td>
<td>59.1 5.9</td>
<td>7.2</td>
<td>0.552</td>
<td>13</td>
</tr>
<tr>
<td>14</td>
<td>59.4 6.3</td>
<td>7.2</td>
<td>0.511</td>
<td>13</td>
</tr>
</tbody>
</table>

## Project Completion Times When Subproject Times Are Deterministic

<table>
<thead>
<tr>
<th>Size of Labor Pool</th>
<th>Project Completion Time</th>
<th>Number of Busy Workers on Average</th>
<th>Utilization of Labor Pool</th>
<th>Maximum Number of Workers in Use Concurrently</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>117</td>
<td>3.6</td>
<td>0.720</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>99</td>
<td>4.3</td>
<td>0.709</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>85</td>
<td>5.0</td>
<td>0.708</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>60</td>
<td>6.8</td>
<td>0.877</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>67</td>
<td>6.3</td>
<td>0.698</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>67</td>
<td>6.3</td>
<td>0.628</td>
<td>10</td>
</tr>
<tr>
<td>11</td>
<td>57</td>
<td>7.4</td>
<td>0.671</td>
<td>11</td>
</tr>
<tr>
<td>12</td>
<td>57</td>
<td>7.4</td>
<td>0.615</td>
<td>11</td>
</tr>
<tr>
<td>13</td>
<td>57</td>
<td>7.4</td>
<td>0.568</td>
<td>11</td>
</tr>
<tr>
<td>14</td>
<td>57</td>
<td>7.4</td>
<td>0.528</td>
<td>11</td>
</tr>
</tbody>
</table>
The **GATHER** Block

- The **GATHER** Block gathers together a specified number of members of an Assembly Set before letting them all go on on their way.

```
A
GATHER
```

The initiator of the gathering process and the additional gatherees go in.

```
the initiator of the gathering process and the additional gatherees come out at the conclusion of the gathering process.
```

<table>
<thead>
<tr>
<th>Operand</th>
<th>Significance</th>
<th>Default Value or Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Gather Count; the number of Assembly-Set members which are to be gathered at the GATHER Block</td>
<td>Compile-Time Error</td>
</tr>
</tbody>
</table>
An Example of Gather-Chain Use

Clock

Gather Chain

Remaining Gather Count

- When a gathering operation has been completed, the scan of the CEC is restarted immediately
A Model for a Corking-and-Labeling Operation in a Small Winery

SEED THE SEGMENT

WORKER TAKES NEXT UNCORKED BOTTLE

PUT A CORK IN THE BOTTLE

PROVIDE ANOTHER UNCORKED BOTTLE

THIS BOTTLE HAS BEEN CORKED

BOOST PRIORITY FOR EVENTUAL LABELING OPERATION

LABELING WON'T START UNTIL 24 BOTTLES HAVE BEEN CORKED

TERMINATE

WORKER TAKES NEXT UNLABELED BOTTLE

PUT A LABEL ON THE BOTTLE

THIS BOTTLE HAS BEEN LABELED

BOTTLE GOES INTO A CASE (12 TO THE CASE)

RECORD THE INTERARRIVAL TIME OF FULL CASES

CASE LEAVES CORKING-AND-LABELING AREA
SIMULATE  A Model for Corking and Labeling *
* Bottles of Wine by Hand in a Small Winery *
* (Basis: Page 476 in the 1974 Red Book) *
* 
IATIME  TABLE  IA,200,100,7  interarrival time (filled cases)
* 
** Model Segment 1 (The Labeling and Corking of Wine Bottles) **
** 
| GENERATE  | BACK  | 0,,1 | seed the segment |
| SEIZE     | SEIZE | WORKER | worker takes next uncorked bottle |
| ADVANCE   | ADVANCE | 8,3 | put a cork in the bottle |
| RELEASE   | RELEASE | WORKER | this bottle has been corked |
| SPLIT     | SPLIT | 1,BACK | provide another uncorked bottle |
| PRIORITY   | PRIORITY | 1 | boost priority for eventual |
| GATHER    | GATHER | 24 | labeling operation |
| SEIZE     | SEIZE | WORKER | labeling won't start until |
| ADVANCE   | ADVANCE | 14,5 | 24 bottles have been corked |
| RELEASE   | RELEASE | WORKER | worker takes next unlabeled bottle |
| ASSEMBLE  | ASSEMBLE | 12 | put a label on the bottle |
| TABULATE  | TABULATE | IATIME | this bottle has been labeled |
| TERMINATE | TERMINATE | 0 | bottle goes into a case |
|           |           |     | 12 per case |
|           |           |     | record full-case interarrival time |
|           |           |     | case leaves the labeling area |

** Model Segment 2 (Run-control Transaction) **

| GENERATE  | 28800 | 28800 = 8 hours expressed in seconds |
| TERMINATE | 1 | stop simulating after 8 hours |

** Run Control Statements **

| START     | 1,,1 | simulate; print chains at the end |
| END       |     | end of Model-File processing |
RELATIVE CLOCK: 28800.000  ABSOLUTE CLOCK: 28800.000

<table>
<thead>
<tr>
<th>BLOCK CURRENT</th>
<th>TOTAL</th>
<th>BLOCK CURRENT</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>11</td>
<td>1296</td>
</tr>
<tr>
<td>2</td>
<td>1317</td>
<td>12</td>
<td>108</td>
</tr>
<tr>
<td>3</td>
<td>1317</td>
<td>13</td>
<td>108</td>
</tr>
<tr>
<td>4</td>
<td>1316</td>
<td>14</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>2632</td>
<td>15</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>1316</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>20</td>
<td>1316</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1296</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>1296</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>1296</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

ASSEMBLY SET 1

GATHER IN PROGRESS AT BLOCK 7
REMAINING COUNT = 4

<table>
<thead>
<tr>
<th>XACT</th>
<th>CURBLK</th>
<th>NXTBLK</th>
<th>CHAINS</th>
<th>SDPGFT**</th>
<th>MARK-TIME</th>
<th>MOVE-TIME</th>
<th>PRIORITY</th>
<th>PC</th>
<th>ASMSET</th>
<th>ADDR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1298</td>
<td>7</td>
<td>8</td>
<td>GTH</td>
<td>0.0</td>
<td>28649.6264</td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td>00945F78</td>
</tr>
</tbody>
</table>

(to save space, all Xacts on the Gather Chain except the first and the last have been deleted here)

<table>
<thead>
<tr>
<th>XACT</th>
<th>CURBLK</th>
<th>NXTBLK</th>
<th>CHAINS</th>
<th>SDPGFT**</th>
<th>MARK-TIME</th>
<th>MOVE-TIME</th>
<th>PRIORITY</th>
<th>PC</th>
<th>ASMSET</th>
<th>ADDR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1317</td>
<td>7</td>
<td>8</td>
<td>GTH</td>
<td>0.0</td>
<td>28797.2147</td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td>00945D38</td>
</tr>
</tbody>
</table>

TABLE IATIME

<table>
<thead>
<tr>
<th>ENTRIES IN TABLE</th>
<th>MEAN ARGUMENT</th>
<th>STANDARD DEVIATION</th>
<th>SUM OF ARGUMENTS</th>
<th>NON-WEIGHTED</th>
</tr>
</thead>
<tbody>
<tr>
<td>107.0000</td>
<td>264.1391</td>
<td>97.0296</td>
<td>28262.8843</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>UPPER LIMIT</th>
<th>OBSERVED FREQUENCY</th>
<th>PERCENT OF TOTAL</th>
<th>CUMULATIVE PERCENTAGE</th>
<th>CUMULATIVE REMAINDER</th>
<th>MULTIPLE OF MEAN</th>
<th>DEVIATION FROM MEAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>200.0000</td>
<td>54.0000</td>
<td>50.47</td>
<td>50.47</td>
<td>49.53</td>
<td>0.7572</td>
<td>-0.6610</td>
</tr>
<tr>
<td>400.0000</td>
<td>53.0000</td>
<td>49.53</td>
<td>100.00</td>
<td>0.00</td>
<td>1.5144</td>
<td>1.4002</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
More About the GATHER Block

- In general, the Gather Count is the value of an arithmetic expression, and so can conceivably take on values of 1 or 0 (or even negative values) as well as values greater than 1.

- If an Xact tries to initiate a gathering operation and the Gather Count is 1, the Xact remains on the CEC and continues to be the moving Xact.

- If an Xact tries to initiate a gathering operation and the Gather Count is 0 or less, an execution error occurs.

- If the last surviving member of an Assembly Set tries to initiate a gathering operation and the Gather Count exceeds 1, then an execution error occurs:

  Error 467: entering Xact is the only member of its Assembly Set

- If the last surviving member of an Assembly Set (or if the last member not currently on an Assembly Chain or a Gather Chain ... or on a Matching Chain, to be discussed later) executes a GATHER Block without completing the ongoing gathering operation, no execution error results and no warning is issued.

  (this is despite the fact that the gatherers are doomed to remain on the Gather Chain "forever".)
Synchronizing the Movement of Assembly-Set Members in Distinct Parts of a Model

- Suppose that no (additional) members of an Assembly Set are to proceed past one point in a model until a(nother) member of the Assembly Set has reached some other point in the model, as suggested in this figure:
The synchronization objective illustrated in the preceding frame can be achieved by using a pair of MATCH Blocks, each of which takes the other as its conjugate:
The MATCH Block

When the moving Xact enters this Block, the conjugate MATCH Block is checked to see if a match has been made.

If no match has been made, the moving Xact is put onto a Matching Chain; if a match is made, the moving Xact continues to move and the matched Xact comes out of the conjugate MATCH Block (in order of last-in, first-out at the conjugate Block) at the same simulated time.

<table>
<thead>
<tr>
<th>Operand</th>
<th>Significance</th>
<th>Default Value or Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Label attached to the conjugate MATCH Block</td>
<td>Compile-Time Error</td>
</tr>
</tbody>
</table>
An Example of Matching-Chain Use
SIMULATE A Model to Display GPSS/H
Matching-Chain Printout

This model also demonstrates that matches are made on a last-in, first-out (LIFO) basis

Model Segment 1 (Path Containing the BLOCKA MATCH Block)

GENERATE 0,1 provide a parent (Xact id #1)
SPLIT 1,PAT2 send a child (Xact id #2) to PATH2
SPLIT 1,BLOCKA send a child (Xact id #3) to BLOCKA
ADVANCE 10 Xact id #1 pauses 10 time units, then joins its second child (Xact id #3) which is already in the BLOCKA MATCH

BLOCKA MATCH BLOCKB wait for Xact id #2 to make a match at BLOCKB

Model Segment 2 (Path Containing the BLOCKB MATCH Block)

PATH2 ADVANCE 25 Xact id #2 pauses 25 time units
BLOCKB MATCH BLOCKA Xact id #2 makes a match at BLOCKA, springing loose its parent (Xact id #1)
PRINT ,,MOV Xact id #2 displays the CEC
PRINT ,,MAT and displays the Matching Chain,
TERMINATE 1 then stops the Xact-Movement Phase

Control Statement Section

START 1 start the simulation
END end of Model-File processing
CURRENT EVENTS CHAIN

<table>
<thead>
<tr>
<th>XACT</th>
<th>CURBLK</th>
<th>NXTBLK</th>
<th>CHAINS</th>
<th>SDPGFT**</th>
<th>MARK-TIME</th>
<th>MOVE-TIME</th>
<th>PRIORITY</th>
<th>PC</th>
<th>ASMSET</th>
<th>ADDR</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>BLOCKB</td>
<td>8</td>
<td>CEC</td>
<td></td>
<td>0.</td>
<td>25.0000</td>
<td>0</td>
<td>1</td>
<td></td>
<td>00946550</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>XACT</th>
<th>CURBLK</th>
<th>NXTBLK</th>
<th>CHAINS</th>
<th>SDPGFT**</th>
<th>MARK-TIME</th>
<th>MOVE-TIME</th>
<th>PRIORITY</th>
<th>PC</th>
<th>ASMSET</th>
<th>ADDR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>BLOCKA</td>
<td>PATH2</td>
<td>CEC</td>
<td></td>
<td>0.</td>
<td>10.0000</td>
<td>0</td>
<td>1</td>
<td></td>
<td>009466A0</td>
</tr>
</tbody>
</table>

ASSEMBLY SET 1

MATCH IN PROGRESS AT BLOCK BLOCKA

<table>
<thead>
<tr>
<th>XACT</th>
<th>CURBLK</th>
<th>NXTBLK</th>
<th>CHAINS</th>
<th>SDPGFT**</th>
<th>MARK-TIME</th>
<th>MOVE-TIME</th>
<th>PRIORITY</th>
<th>PC</th>
<th>ASMSET</th>
<th>ADDR</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>BLOCKA</td>
<td>PATH2</td>
<td>MCH</td>
<td></td>
<td>0.</td>
<td>0.</td>
<td>0</td>
<td>1</td>
<td></td>
<td>009464F0</td>
</tr>
</tbody>
</table>
Block Diagram for a MATCH-Block Application
(taken from Chapter 7 in the Red Book)

GENERATE
300,200

SPLIT
(RUTE2)
1

SPLIT EACH UNIT INTO TWO PARTS

SEIZE
MANT

CAPTURE FIRST WORKER

ADVANCE
100,20

(BLOK1)

MATCH
BLOK2

COORDINATE TO CHECK TOLERANCES

ADVANCE
50,5

(MERGE)

2

ASSEMBLE

STEP 3 CANNOT START UNTIL BOTH STEP 2'S ARE DONE

ADVANCE
25,5

RELEASE
MANT

RELEASE FIRST WORKER

TABULATE

RECORD THE INTERARRIVAL TIME

TERMINATE

UNITS OF WORK ARRIVE

SEIZE

MAN2

CAPTURE SECOND WORKER

ADVANCE
110,25

(BLOK2)

MATCH
BLOK1

ADVANCE
70,10

RELEASE
MAN2

RELEASE SECOND WORKER

TRANSFER

PART 2 GOES TO STEP 3

FINISHED UNIT LEAVES THE MODEL

K-38
More About the MATCH Block

- If the last surviving member of an Assembly Set executes a MATCH Block, then an execution error occurs:

  Error 467:
  entering Xact is the only member of its Assembly Set

- In general, a MATCH Block's conjugate does not have to be a MATCH Block, but can be any type of Block

- For example, a MATCH Block can take a GENERATE Block as its conjugate, but if it does so, then no match can ever be made
  (there is no way for an Xact to be in matching status at a GENERATE Block)
and so this MATCH Block will be a sink for entering Xacts
  (no Warning Messages or Error Messages are issued in cases like this;
   note that some other MATCH Block could take this MATCH as its conjugate,
   however, so these Xacts are not necessarily doomed to remain on their Matching Chain forever)

- Perhaps the easiest way to work with MATCH Blocks is to use them as conjugate pairs
Model K5

SIMULATE A Model to Display GPSS/H *
Assembly, Gather, and Matching Chain Printouts *
for Multiple Chains of Each Type *

(Note that all members of Assembly Set #1 wind up on *
Matching, Assembly, or Gather Chains in this model. *
They will be on these chains "forever," because there *
aren't one or more other members of their Assembly Set *
who are "on the loose" to provide the potential for *
eventually completing the ongoing Assembly or Gather *
operations, or making a Match, for Assembly Set #1. *
No Warning or Error Messages are issued by the software.) *

Model Segment 1 (Members of Assembly Set #1) *

NOTSMART GENERATE 0,,1 provide a parent *

SPLIT 1,MAKEMORE sends a child to a SPLIT sequence *
SPLIT 1,ASSEMP1 sends a child to an ASSEMBLE *
SPLIT 1,GATHRPT1 sends a child to a GATHER *
SPLIT 1,MATCHPT1 sends a child to a MATCH *

the parent...
then goes into an ASSEMBLE itself
assembly point 1
gathering point 1

ASSEMP1 ASSEMBLE 3 assembly point 2
GATHRPT1 GATHER 3 gathering point 2
MATCHPT1 MATCH NOTSMART matching point 1

the child...
then goes into an ASSEMBLE itself
assembly point 2

MAKEMORE SPLIT 1,ASSEMP2 sends 1 grandchild to an ASSEMBLE *
SPLIT 1,GATHRPT2 sends 1 grandchild to a GATHER *
SPLIT 1,MATCHPT2 sends 1 grandchild to a MATCH *

ASSEMP2 ASSEMBLE 3 assembly point 2
GATHRPT2 GATHER 3 gathering point 2
MATCHPT2 MATCH NOTSMART matching point 2

Model Segment 2 (Control Transaction) *

GENERATE 10,,1 create a control Transaction *
PRINT ,,MAT display the MAG Chains *
TERMINATE 1 end Xact Movement *

Run-Control Statements *

START 1 start Xact movement *

END end of Model-File processing
ASSEMBLY SET 1

MATCH IN PROGRESS AT BLOCK MATCHPT2

<table>
<thead>
<tr>
<th>XACT</th>
<th>CURBLK</th>
<th>NXTBLK</th>
<th>CHAINS</th>
<th>SDPGFT**</th>
<th>MARK-TIME</th>
<th>MOVE-TIME</th>
<th>PRIORITY</th>
<th>PC</th>
<th>ASMSET</th>
<th>ADDR</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>MATCHPT2</td>
<td>15</td>
<td>MCH</td>
<td>0.</td>
<td>0.</td>
<td>0</td>
<td>1</td>
<td>00946250</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

MATCH IN PROGRESS AT BLOCK MATCHPT1

<table>
<thead>
<tr>
<th>XACT</th>
<th>CURBLK</th>
<th>NXTBLK</th>
<th>CHAINS</th>
<th>SDPGFT**</th>
<th>MARK-TIME</th>
<th>MOVE-TIME</th>
<th>PRIORITY</th>
<th>PC</th>
<th>ASMSET</th>
<th>ADDR</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>MATCHPT1</td>
<td>MAKENORE</td>
<td>MCH</td>
<td>0.</td>
<td>0.</td>
<td>0</td>
<td>1</td>
<td>009463D0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

ASSEMBLY IN PROGRESS AT BLOCK ASSEMPT2  REMAINING COUNT = 1

<table>
<thead>
<tr>
<th>XACT</th>
<th>CURBLK</th>
<th>NXTBLK</th>
<th>CHAINS</th>
<th>SDPGFT**</th>
<th>MARK-TIME</th>
<th>MOVE-TIME</th>
<th>PRIORITY</th>
<th>PC</th>
<th>ASMSET</th>
<th>ADDR</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>ASSEMPT2</td>
<td>GATHRPT2</td>
<td>ASM</td>
<td>0.</td>
<td>0.</td>
<td>0</td>
<td>1</td>
<td>009464F0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

ASSEMBLY IN PROGRESS AT BLOCK ASSEMPT1  REMAINING COUNT = 1

<table>
<thead>
<tr>
<th>XACT</th>
<th>CURBLK</th>
<th>NXTBLK</th>
<th>CHAINS</th>
<th>SDPGFT**</th>
<th>MARK-TIME</th>
<th>MOVE-TIME</th>
<th>PRIORITY</th>
<th>PC</th>
<th>ASMSET</th>
<th>ADDR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ASSEMPT1</td>
<td>GATHRPT1</td>
<td>ASM</td>
<td>0.</td>
<td>0.</td>
<td>0</td>
<td>1</td>
<td>009466A0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

GATHER IN PROGRESS AT BLOCK GATHRPT2  REMAINING COUNT = 2

<table>
<thead>
<tr>
<th>XACT</th>
<th>CURBLK</th>
<th>NXTBLK</th>
<th>CHAINS</th>
<th>SDPGFT**</th>
<th>MARK-TIME</th>
<th>MOVE-TIME</th>
<th>PRIORITY</th>
<th>PC</th>
<th>ASMSET</th>
<th>ADDR</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>GATHRPT2</td>
<td>MATCHPT2</td>
<td>GTH</td>
<td>0.</td>
<td>0.</td>
<td>0</td>
<td>1</td>
<td>00946280</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

GATHER IN PROGRESS AT BLOCK GATHRPT1  REMAINING COUNT = 2

<table>
<thead>
<tr>
<th>XACT</th>
<th>CURBLK</th>
<th>NXTBLK</th>
<th>CHAINS</th>
<th>SDPGFT**</th>
<th>MARK-TIME</th>
<th>MOVE-TIME</th>
<th>PRIORITY</th>
<th>PC</th>
<th>ASMSET</th>
<th>ADDR</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>GATHRPT1</td>
<td>MATCHPT1</td>
<td>GTH</td>
<td>0.</td>
<td>0.</td>
<td>0</td>
<td>1</td>
<td>00946430</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The Use of GATE Blocks with Assembly Sets

- Xacts can use GATE Blocks to determine whether one or more members of their Assembly Set are on a MAG Chain

  (MAG Chain: Matching, Assembly or Gather Chain)

  associated with a specified MATCH, ASSEMBLE, or GATHER Block

- The GATE Block Logical Operators M and NM are used for this purpose, as summarized below:

![Diagram of GATE Block with operands A and B](image)

(Use of the B Operand is optional)

<table>
<thead>
<tr>
<th>Operand</th>
<th>Significance</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Label attached to a Block</td>
<td>No Default</td>
</tr>
<tr>
<td>LogOp</td>
<td>Logical Operator; takes either of these two forms: M (Matching Status) NM (No Matching Status)</td>
<td>No Default</td>
</tr>
<tr>
<td>B</td>
<td>Pointer to a labeled Block (a copy of that Block's Label)</td>
<td>No B-Op means wait-until mode, not path-selection mode</td>
</tr>
</tbody>
</table>
The Use of GATE Blocks with Assembly Sets
(continued)

- Here are examples of wait-until and path-selection mode GATE Blocks using M and NM as Logical Operators

- Blocking at a wait-until GATE M or GATE NM Block
  (like blocking at a wait-until GATE FS
   or GATE FNS Block)
  is non-unique

  (in contrast, blocking at wait-until GATE Blocks using any of the other Logical Operators is unique)
The SPLIT Block's Optional Serialization Operand (C Operand) and Parameter Operands (D, E, F and G Operands)

Operands | Significance | Default Value |
--- | --- | --- |
A | Split Count | No Default |
B | Next Block for the children | No Default |
C | Specifies the identifier for and the type of a Parameter in which the parent and its children are to be **serially numbered** | No serial numbering occurs |
D,...,G | Specify the numbers and types of Parameters each child is to have | Children match their parent in numbers and types of Parameters |

**Examples**

`SPLIT 5,ROUTE66,(SEQUENCE)PF`

`SPLIT FN(LOT_SIZE),SALT,MINE,(SERIAL_ID)PF,10PF,5PL`
An Example Showing Use of the SPLIT Block's Serialization Option

assume KEY EQU 3,PF

Parent In

SPLIT

(KEY)PF 2

(BYPASS)

Parent Out

1st Child Out

2nd Child Out

Xact id PR

32 20

18 12

24 12

25 12

39 7

Xact id PR

32 20

18 12

24 12

25 12

52 12

53 12

39 7

CEC

CEC

CEC