The Lehigh University IBM 360 simulator.

Leonard Ira Horey

Follow this and additional works at: http://preserve.lehigh.edu/etd

Part of the Electrical and Computer Engineering Commons

Recommended Citation
THE LEHIGH UNIVERSITY IBM 360 SIMULATOR

by
Leonard Ira Horey

A Thesis
Presented to the Graduate Committee
of Lehigh University
in Candidacy for the Degree of
Master of Science
in
Electrical Engineering

Lehigh University
1975
ProQuest Number: EP76031

All rights reserved

INFORMATION TO ALL USERS
The quality of this reproduction is dependent upon the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript
and there are missing pages, these will be noted. Also, if material had to be removed,
a note will indicate the deletion.

ProQuest

ProQuest EP76031
Published by ProQuest LLC (2015). Copyright of the Dissertation is held by the Author.

All rights reserved.
This work is protected against unauthorized copying under Title 17, United States Code
Microform Edition © ProQuest LLC.

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 - 1346
This thesis is accepted and approved in partial fulfillment of the requirements for the degree of Master of Science.

April 3, 1975
(date)

Professor in Charge

Chairman of Department
Acknowledgment

The author would like to express his gratitude to his advisor, Professor Peggy A. Ota. Her comments and suggestions were a valuable aid in preparing this thesis.
# Table of Contents

Abstract .................................................................................................................. 1

I Introduction ............................................................................................................. 2

1.1 Background ....................................................................................................... 2

1.2 Contents of the Thesis .................................................................................... 6

II User's Guide to LUIS .......................................................................................... 7

2.1 Introduction ....................................................................................................... 7

2.2 Restricted Addresses ....................................................................................... 8

2.3 The Instruction Set .......................................................................................... 9

2.4 Use of the Simulator ....................................................................................... 11

III The Routines of LUIS ....................................................................................... 20

3.1 Introduction ..................................................................................................... 20

3.2 Description of the Routines ............................................................................ 21

3.2.1 The Main Routine ................................................................................... 21

3.2.2 Subroutine BOMB .................................................................................. 21

3.2.3 Subroutine DUMP .................................................................................... 23

3.2.4 Subroutine REED .................................................................................... 23

3.2.5 Subroutine RITE ....................................................................................... 26

3.2.6 Subroutine EXIST .................................................................................... 26

3.2.7 Subroutine DEC ......................................................................................... 26

3.2.8 Subroutine CONVERT ............................................................................ 27

3.2.9 Subroutine HEX ......................................................................................... 27

3.2.10 Subroutine FIBM .................................................................................. 28

IV FIBM .................................................................................................................... 29

4.1 Introduction ..................................................................................................... 29
4.2 The Memory of the Simulated IBM 360 .......... 30
4.3 The Storage of the PSW and the General Registers .................................. 31
4.4 The Execution of the User's Program .......... 32
4.5 An Alternative Structure for FIBM .......... 34

V The Flowchart of LUIS .......................... 35
5.1 Notation ....................................... 35
5.2 Flowchart .................................... 36

VI Improving LUIS .................................. 104
6.1 Introduction .................................. 104
6.2 Adding More Memory .......................... 105
6.3 Adding More Instructions ..................... 106
6.4 Abbreviated Messages From LUIS ............. 107
6.5 Additional Simulator Commands ............... 108

References ....................................... 109
Appendix A ....................................... 110
Appendix B ....................................... 117
Vita ............................................... 120
List of Figures

1.1 The Program Status Word .................................. 4
1.2 The Simulated IBM 360 ....................................... 4
3.1 The "GOOD" Common Block .................................. 22
3.2 A Typical Output Printed by Subroutine DUMP;
   Entry Point PREDUMP ............................................. 22
3.3 The "STATUS" Common Block .................................. 24
3.4 A Typical Output Printed by Subroutine DUMP ............. 24
3.5 The "BYTES" Common Block .................................... 25
3.6 The "IN" Common Block ......................................... 25
## Contents of the Flowchart

<table>
<thead>
<tr>
<th>Main Program</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAIN1</td>
<td>36</td>
</tr>
<tr>
<td>MAIN2</td>
<td>36</td>
</tr>
<tr>
<td>REQUEST</td>
<td>37</td>
</tr>
<tr>
<td>DUM</td>
<td>38</td>
</tr>
<tr>
<td>INSERT</td>
<td>39</td>
</tr>
<tr>
<td>INPGM</td>
<td>40</td>
</tr>
<tr>
<td>ININ</td>
<td>40</td>
</tr>
<tr>
<td>EXECUTE</td>
<td>41</td>
</tr>
<tr>
<td>SQ</td>
<td>41</td>
</tr>
<tr>
<td>STATUS</td>
<td>41</td>
</tr>
<tr>
<td>STORE</td>
<td>42</td>
</tr>
<tr>
<td>REWINDP</td>
<td>43</td>
</tr>
<tr>
<td>REWINDS</td>
<td>43</td>
</tr>
<tr>
<td>Subroutine BOMB</td>
<td>44</td>
</tr>
<tr>
<td>Subroutine DUMP</td>
<td>44</td>
</tr>
<tr>
<td>Entry PREDUMP</td>
<td>44</td>
</tr>
<tr>
<td>Subroutine REED</td>
<td>45</td>
</tr>
<tr>
<td>Subroutine RITE</td>
<td>46</td>
</tr>
<tr>
<td>Subroutine EXIST</td>
<td>46</td>
</tr>
<tr>
<td>Subroutine DEC</td>
<td>46</td>
</tr>
<tr>
<td>Subroutine CONVERT</td>
<td>46</td>
</tr>
<tr>
<td>Subroutine HEX</td>
<td>46</td>
</tr>
<tr>
<td>Subroutine FIBM</td>
<td>47</td>
</tr>
<tr>
<td>CHANGE</td>
<td>48</td>
</tr>
<tr>
<td>WHYBOMB</td>
<td>48</td>
</tr>
<tr>
<td>IOERR</td>
<td>49</td>
</tr>
<tr>
<td>ER</td>
<td>49</td>
</tr>
<tr>
<td>INSTRA</td>
<td>49</td>
</tr>
<tr>
<td>RR</td>
<td>50</td>
</tr>
<tr>
<td>NON</td>
<td>50</td>
</tr>
<tr>
<td>L</td>
<td>50</td>
</tr>
<tr>
<td>LTR</td>
<td>51</td>
</tr>
<tr>
<td>LCR</td>
<td>51</td>
</tr>
<tr>
<td>LPR</td>
<td>51</td>
</tr>
<tr>
<td>LNR</td>
<td>52</td>
</tr>
<tr>
<td>A</td>
<td>52</td>
</tr>
<tr>
<td>CCC</td>
<td>53</td>
</tr>
</tbody>
</table>

- vii -
<table>
<thead>
<tr>
<th>Command</th>
<th>80</th>
<th>81</th>
<th>82</th>
<th>83</th>
<th>84</th>
<th>85</th>
<th>86</th>
<th>87</th>
<th>88</th>
<th>89</th>
<th>90</th>
<th>91</th>
<th>92</th>
<th>93</th>
<th>94</th>
<th>95</th>
<th>96</th>
<th>97</th>
<th>98</th>
<th>99</th>
<th>100</th>
<th>101</th>
<th>102</th>
<th>103</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRDL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BXH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BXLE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RANGE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FLOAT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HALFB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FULB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OFLOW</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KILL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REGERR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIVBAD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CVB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DOUB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATER</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KEYER</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MVC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MVN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MVZ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>XC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CVD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PACK</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UNPK</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MVO</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DECI</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SK</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WRRDD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIO</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OUT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Abstract

The Lehigh University IBM 360 Simulator (LUIS) is an interactive program which enables a user to execute an IBM 360 machine language program on a Control Data 6400 Computer System. LUIS simulates all but seven of the instructions in the IBM System/360 standard instruction set.

The simulator provides a simulated ten thousand byte memory whose limits are initially specified by the user. The user can load his IBM 360 machine language program into this memory either by entering the program from the remote terminal or by reading the program from a local file. He may then request that either all or a specific number of instructions in the program be executed. If the simulator encounters a situation which would normally interrupt a real IBM 360, it terminates execution of the user's program and issues an error message explaining the reason for interruption. The user can ask to examine the contents of any portion of the simulated IBM 360 memory and to examine the contents of the simulated IBM 360 general registers and the program status word. He can also modify his machine language program and store the modified program on a file for future use.
I Introduction

1.1 Background

This thesis describes an interactive IBM System/360 simulator written for a Control Data 6400 Computer System. The simulator was written to allow Lehigh University students to familiarize themselves with the IBM System/360 family of machines. These students would otherwise be unable to do so in the Lehigh University computing environment.

In particular, the Department of Electrical Engineering offers a senior level elective in systems programming, EE 315. The text currently being used in the course is Systems Programming by John J. Donovan. Donovan (like many other authors) uses the IBM System/360 for his examples. Lehigh University has only a Control Data 6400 Computer System. It would be beneficial to have a set of programs which would enable Lehigh students, those taking EE 315 and others, to write programs in IBM System/360 Basic Assembly Language and have them assembled and executed on a simulated IBM System/360. A first program would assemble the student's program on the Control Data 6400 Computer System and produce a machine language version of the program. A second program would then take this machine language program and execute it.
This thesis describes the second program, the Lehigh University IBM 360 Simulator (LUIS). LUIS is an interactive program which simulates all but seven of the instructions in the IBM System/360 standard instruction set \[1\] (Diagnose, Set System Mask, Load PSW, Halt I/O, Supervisor Call, Test Channel, and Test I/O). The program is available through the Lehigh University Computing Center.

The simulated IBM 360 has a ten thousand byte memory (which is stored in the NW array in the simulator) and sixteen general registers (which are stored in the "STATUS" common block). The simulated IBM 360 also has a program status word (PSW) which is stored in the "STATUS" common block. The PSW (Fig. 1.1) contains information needed to execute the user's program. Subroutine FIBM acts as the central processing unit for the simulated IBM 360. (Fig. 1.2)

Since all user input and output must be done through the Control Data 6400 Computer System and the simulator, the input/output instructions Test Channel, Test I/O, and Halt I/O are not used. Instead all input/output is done by using a modified version of the Start I/O instruction. Since input/output is not performed in the usual manner, the system mask, which is concerned with input/output interrupts, is not relevant and thus the Set System Mask instruction is omitted. The user initially specifies the PSW and can change it through the use of
condition code
instruction length code (ILC)

program mask—If the leftmost bit of the four bit hexadecimal character is one, program interruption will occur if there is a fixed-point overflow. The other three bits of the program mask are ignored by LUIS.

AMWP—If the leftmost bit of the four bit hexadecimal character is one, generated decimal sign and zone codes are in USASCII-8 (normally they are in EBCDIC). The other three bits of the hexadecimal character are ignored by LUIS.

Fig. 1.1 The Program Status Word
(Each block represents one four bit hexadecimal character.)

Fig. 1.2 The Simulated IBM 360

PSW
(stored in the 'STATUS' common block)

Central Processing Unit
Subroutine
FIBM

Memory
(NW array)

General Registers
(stored in the 'STATUS' common block)
the request NEWPSW (see LUIS User's Guide), so the Load PSW instruction is not needed. Obviously, since one is not working with a real IBM System/360, the Diagnose instruction (which is used for testing the system's hardware) has no meaning and is not included. Because all input/output is done through the simulator, and because none of the other privileged operations are included, there is no need for a Supervisor Call and this instruction has also been omitted.
1.2 Contents of the Thesis

The remaining portion of this thesis is divided into five chapters and two appendices. Chapter II is a user's guide to using the Lehigh University IBM 360 Simulator. Chapter III gives a short description of each of the ten routines which comprise the simulator. Chapter IV gives a more detailed description of the subroutine (FIBM) which actually simulates the IBM 360. Chapter V contains the flowchart of LUIS. The last chapter, Chapter VI, describes how additional capabilities can be added to LUIS. Appendix A contains material (reproduced from the manual, IBM System/360 Principles of Operation) which should be of value to those who are unfamiliar with the IBM 360 structure. Appendix B contains a typical output from LUIS.
II User's Guide to LUIS

2.1 Introduction

The Lehigh University IBM 360 Simulator (LUIS) takes a program written in IBM System/360 machine language and executes it on a Control Data 6400 Computer System. LUIS simulates all but seven of the instructions in the IBM System/360 standard instruction set. (The instructions Diagnose, Set System Mask, Load PSW, Halt I/O, Supervisor Call, Test Channel, and Test I/O are omitted.) This chapter describes the features and use of LUIS. Section 2 discusses a restriction on the addresses used with LUIS. Section 3 describes the commands of LUIS. The last section, Section 4, explains how to use the simulator.

*LUIS operates under INTERCOM. INTERCOM is a subsystem which allows the user to run an interactive program from a remote terminal on the Control Data 6400 Computer System.
2.2 Restricted Addresses

One important difference between the simulator and the IBM System/360 is that addresses used with LUIS are restricted to a maximum of seventeen significant bits. If the eighteenth bit is a one, the address will be interpreted as a negative number and the program will be terminated if and when that address is actually used for addressing. If more than eighteen significant bits are used, the address will be truncated when it is used for addressing. If the truncated address is within the memory area of the program, the program will continue to execute normally. However, if the resulting address is outside the memory area of the program, the simulator will issue an error message and terminate execution of the program. This restriction on the length of addresses is necessitated by the fact that the A and B registers in the CDC 6400 Computer System are only eighteen bits long. (One should note the fact that if an address field is not actually used for addressing storage such as in a "Load Address" instruction or shifting instructions, then this limitation does not apply.)
2.3 The Instruction Set

All of the simulated instructions (except Start I/O) function as described in the manual, *IBM System/360 Principles of Operation*¹. The floating-point feature instructions, the decimal feature instructions, the protection feature instructions, and the direct control feature instructions are not available. Conditions which would normally produce an interrupt in an IBM 360 cause the simulator to terminate execution of the program and print an error message indicating the reason for the interrupt.

The user's program can perform I/O by using a modified version of the Start I/O instruction.

```
  B1
 /   \
|    |
|    | I/O code
\    /
  B1   D1
```

The I/O code in the instruction indicates the type of I/O that is to be performed by the program. The code is 01 for writing and 00 for reading. (All other codes are invalid.) The address specified by B1 and D1 is a fullword somewhere in the user's (memory) storage area. This fullword contains the count of the number of words which the program will read or write and a base register and displacement which specify the starting address for I/O. The starting address must specify a fullword boundary.
When the I/O code specifies writing, the simulator responds:

WORDS PRINTED FROM MEMORY BY THE USER'S PROGRAM

The simulator then prints the words and their corresponding addresses. When the I/O code specifies reading, the simulator responds:

YOUR PROGRAM WANTS TO READ SOME WORDS.
IF YOU HAVE THE WORDS ON THE FILE PGM, TYPE PGM.
IF YOU ARE GOING TO ENTER THE WORDS, TYPE INPUT.

Words are read in the same manner as described for "INSERT". If an error occurs during reading, the simulator issues an error message and terminates execution of the program. If the reading of the words is completed in a satisfactory manner the simulator responds:

WORDS READ INTO MEMORY BY THE USER'S PROGRAM

The simulator then prints the words which it read and the locations where each was placed.

If no errors were detected while performing I/O, then the execution of the program continues after I/O is completed.
2.4 Use of the Simulator

The simulator is an interactive program. To use the simulator the user first attaches the simulator while in the INTERCOM command mode. If the user has his program on a permanent file, then he must also attach this file (with the local file name of PGM). After the user attaches the simulator, he types:

LUIS.

At this point the user is in the simulator program. The simulator responds:

LEHIGH UNIVERSITY IBM 360 SIMULATOR

ENTER THE SMALLEST ADDRESS IN YOUR PROGRAM AS AN EIGHT DIGIT HEXADECIMAL NUMBER.

The user enters the smallest address in his program. This address determines one of the boundaries of his storage area. The address can be between 00000000 and 0001D8A8 hexadecimal and must specify a fullword boundary. If the address is not acceptable to the simulator, it responds with an error message and repeats its request. If the address is acceptable the simulator responds:

ENTER THE MAXIMUM SIZE OF YOUR PROGRAM (IN BYTES) AS A FOUR DIGIT DECIMAL INTEGER.

The user enters the number of bytes in his storage area. This number, together with the smallest address, determines the boundaries of the user's program. The simulator will prevent the user from exceeding the boundaries of his storage area. If the number which the user enters is
not acceptable to the simulator, it responds with an error message, and repeats its request. If the number is acceptable, the simulator responds:

ENTER THE PSW AS A 16 DIGIT HEXADECIMAL NUMBER.

The user enters his program's PSW. If the PSW is not acceptable, an error message will be issued and the request will be repeated. If the PSW is acceptable the simulator responds:

REQUEST=

At this point the user is in the simulator request mode and can issue any of the following requests:

END
BYE
INSERT
NEWPSW
DUMP
STATUS
S
EXECUTE
REWINDP
STORE
REWINDS

These requests are explained below.

END

The request END enables the user to redefine his storage area and start a new program without leaving the simulator program. The simulator responds:

PROGRAM ENDED BY USER

The simulator then requests the information needed to define the storage area (the smallest address in the
program, the maximum size of the program, and the PSW). One should note that the simulator zeros the program's storage area when the storage area is defined by the user. Thus any bytes which were in the storage area from a previous program are wiped out.

**BYE**

The request **BYE** terminates the simulator program and returns the user to the INTERCOM command mode. INTERCOM will respond:

```
EXIT
COMMAND-
```

The request **BYE** may also be called after the simulator responds:

```
ENTER THE SMALLEST ADDRESS IN YOUR PROGRAM AS AN EIGHT DIGIT HEXADECIMAL NUMBER.
```

**INSERT**

The request **INSERT** tells the simulator that the user wants to insert some bytes into his storage area. The simulator responds:

```
ENTER THE STARTING ADDRESS (OF THE COLLECTION OF BYTES) AS AN EIGHT DIGIT HEXADECIMAL NUMBER.
```

The address must specify a fullword boundary. If the address which the user enters is not acceptable, the simulator issues an error message and repeats its request. If the address is acceptable the simulator responds:

```
ENTER THE NUMBER OF BYTES WHICH WILL BE INSERTED AS A FOUR DIGIT DECIMAL INTEGER.
```
If the number is not acceptable the simulator issues an error message and repeats its request. If the number is acceptable the simulator responds:

IF YOU HAVE THE BYTES ON THE FILE PGM, TYPE PGM.
IF YOU ARE GOING TO ENTER THE BYTES, TYPE INPUT.

If the user types "INPUT" the simulator responds:

ENTER THE BYTES. FOUR BYTES PER LINE.

and prints the location where the bytes will be inserted. When the user enters the bytes, the simulator responds by printing the location where the next group of bytes will be inserted. This process continues until the user finishes entering all of the bytes. After the simulator reads all of the bytes, it responds:

THE BYTES HAVE BEEN READ

and returns the user to the request mode.

If the user types "PGM" the simulator assumes that the bytes are on the local file PGM. If the files does not exist the simulator will respond:

PGM DOES NOT EXIST

and will return the user to the request mode. The simulator assumes that the file PGM has the bytes packed forty bytes per card image. The simulator reads as many cards as are necessary to satisfy the user's INSERT request. If a second INSERT request causes the simulator to read from PGM a second time, the simulator will begin reading at the next card image. If the simulator reads the EOF because PGM does not contain enough bytes, the
simulator issues an error message and zeros all locations mentioned in the INSERT. After the simulator reads all of the bytes, it responds:

THE BYTES HAVE BEEN READ

and returns the user to the request mode.

NEWPSW

The request NEWPSW enables the user to change his program's PSW. The simulator responds:

ENTER THE PSW AS A 16 DIGIT HEXADECIMAL NUMBER.

After the user enters an acceptable PSW, the simulator returns the user to the request mode.

DUMP

The request DUMP enables the user to dump all or a portion of his memory (storage area). The simulator responds:

TYPE ALL OR PARTIAL

If the user types PARTIAL, the simulator asks the user to supply the starting address of the dump and the length of the dump in bytes. The dump must start on a fullword boundary. The user is not permitted to dump outside his storage area.

Insertions and dumps can only be done with fullword units. Thus all requests are rounded up to the nearest number of fullwords. This rounding process may cause a request to exceed the user's memory area by a fraction of -15-
a word. However, this fact will be ignored by the simulator and will not cause any problem.

After the simulator performs the requested dump, it returns the user to the request mode.

STATUS

The request STATUS causes the simulator to print the PSW, the instruction address, the ILC, the condition code, and the contents of all of the general registers. The simulator then returns the user to the request mode.

S

The request S enables the user to request that a specified number of instructions be executed in his program (starting with the one specified by the instruction address portion of the PSW). The simulator responds:

ENTER THE NUMBER OF INSTRUCTIONS TO BE EXECUTED AS A FOUR DIGIT DECIMAL INTEGER.

After the user enters the number, the simulator executes the specified number of instructions and then returns the user to the request mode.

It is possible for the simulator to return to the request mode before it finishes executing all of the specified instructions. This happens when the simulator encounters a condition which would produce an interrupt in a real IBM 360. In this case the execution of the program stops, and the simulator prints an error message.
indicating the reason for program termination.

**EXECUTE**

The request EXECUTE enables the user to execute his entire program. EXECUTE tells the simulator to execute 9,999 instructions. Usually this will be much greater than the number of instructions in the user's program. Thus the user's program will end when a condition occurs which would produce an interrupt in a real IBM 360. However, if the user's program is either longer than 9,999 instructions or has an endless loop in it, the simulator will return to the request mode after it executes 9,999 instructions. The user can recognize when this happens because no interrupt message occurs before the simulator returns to the request mode.

**REWINDP**

The request REWINDP enables the user to rewind the file PGM. The user must do this if he has previously read from PGM, and then wants to start at the beginning of the file. The simulator rewinds the file and returns the user to the request mode. If the file does not exist the simulator will respond:

**PGM DOES NOT EXIST**

and will return the user to the request mode.
STORE

The request STORE enables the user to dump all or a portion of his memory (storage area) onto a file named STORE. In this manner the user can save the contents of his memory for future use. The simulator responds:

TYPE ALL OR PARTIAL

If the user types PARTIAL, the simulator asks the user to supply the starting address of the dump and the length of the dump in bytes. The request STORE has the same restrictions as the request DUMP (See DUMP). The bytes are written forty bytes per card image on the file STORE. If a second STORE request causes the simulator to write on file STORE a second time, the simulator will begin writing at the next card image. The simulator rewinds file STORE when the user initially executes the simulator program. File STORE is not destroyed when the user terminates the simulator program. Thus the file can be saved by the user and used as the file PGM at some later time.

After the simulator dumps the bytes onto file STORE, it responds:

THE BYTES HAVE BEEN DUMPED ONTO FILE STORE and returns the user to the request mode.
REWINDS

The request REWINDS enables the user to rewind the file STORE. The user must do this if he has previously written on STORE, and then wants to start writing at the beginning of the file. The simulator rewinds the file and returns the user to the request mode.
III The Routines of LUIS

3.1 Introduction

The Lehigh University IBM 360 Simulator is written in COMPASS\[^2,3\] (assembly language) and FORTRAN. The program consists of a main routine and nine subroutines. The COMPASS subroutines assume that the FORTRAN routines are compiled by using the RUN compiler. Approximately 23.6 CP seconds are needed to compile and assemble the program. \(31242\)\(^\text{octal}\) words of central memory are required by the program and the various system routines which it calls. Naturally, additional central memory is needed for the loader and the loader tables. Section 2 gives a general description of each of the routines of LUIS.
3.2 Description of the Routines

3.2.1 The Main Routine

The main routine is written in FORTRAN. All of the communications between the user and the simulator (except some error messages and program I/O) are handled by this routine. The information needed to define the boundaries of the user's memory area is initially requested by the main routine and all user requests are processed through it. Checks are made to insure that no insertions or dumps are performed outside of the user's memory area. If any invalid requests or otherwise erroneous input are received, an appropriate error message is printed, and the request for the input is repeated.
(See flowchart pp. 36 - 43)

3.2.2 Subroutine BOMB

Subroutine BOMB is written in FORTRAN. This subroutine prints error messages. The calling routine passes one parameter to BOMB. This parameter determines which of twenty error messages is to be printed. BOMB also sets a flag if either of two particular error messages are printed. This flag is passed to the main routine through the "GOOD" common block (Fig. 3.1).
(See flowchart p.44)
IBAD

Fig. 3.1 The "GOOD" common block

<table>
<thead>
<tr>
<th>PROGRAM STATUS WORD</th>
<th>00000000798000048</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSTRUCTION ADDRESS</td>
<td>00048</td>
</tr>
<tr>
<td>CONDITION CODE</td>
<td>1</td>
</tr>
<tr>
<td>ILG</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GENERAL REGISTERS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 00000000</td>
</tr>
<tr>
<td>1 0000000000</td>
</tr>
<tr>
<td>2 0000000010</td>
</tr>
<tr>
<td>3 0000000010</td>
</tr>
<tr>
<td>4 0000000000</td>
</tr>
<tr>
<td>5 0000000000</td>
</tr>
<tr>
<td>6 0000000000</td>
</tr>
<tr>
<td>7 0000000000</td>
</tr>
<tr>
<td>8 0000000000</td>
</tr>
<tr>
<td>9 0000000000</td>
</tr>
<tr>
<td>A 0000000000</td>
</tr>
<tr>
<td>B 0000000000</td>
</tr>
<tr>
<td>C 0000000000</td>
</tr>
<tr>
<td>D 0000000000</td>
</tr>
<tr>
<td>E 0000000000</td>
</tr>
<tr>
<td>F 0000000000</td>
</tr>
</tbody>
</table>

Fig. 3.2 A typical output requested by the user through the use of the request; STATUS. (Output printed by subroutine DUMP; entry point PREDUMP.)
3.2.3 Subroutine DUMP

Subroutine DUMP is written in FORTRAN. The subroutine is actually composed of two separate parts. The first part (DUMP) prints the program status word, the instruction address, the condition code, the ILC, and the contents of each of the sixteen general registers (Fig. 3.2). This information is passed to DUMP through the "STATUS" common block (Fig. 3.3). The second part of the subroutine (entry point PREDUMP) prints the byte addresses and the contents at each address (in hexadecimal) (Fig. 3.4). The subroutine must know the byte address of the first word, the corresponding index in the NW array (where the words are actually stored), and the number of bytes which are to be dumped. This information is passed to DUMP through the "BYTES" common block (Fig. 3.5).

(See flowchart p.44)

3.2.4 Subroutine REED

Subroutine REED is written in FORTRAN. REED allows the user's program to read words into memory. Subroutine FIBM passes three parameters (the first byte address, the corresponding index in the NW array (where the user's program is stored), and the number of bytes which are to be read) to REED. The subroutine stores the parameters it receives in the "BYTES" common block and reads the words from the source indicated by the user. If the reading
Fig. 3.3 The "STATUS" common block

Fig. 3.4 A typical output requested by the user through the use of the request; DUMP. (Output printed by subroutine DUMP.)
Fig. 3.5 The "BYTES" common block

Fig. 3.6 The "IN" common block
process is not completed in a satisfactory manner, a parameter (the number of bytes which are to be read) is set to zero before returning to subroutine FIBM. (See flowchart p.45)

3.2.5 Subroutine RITE

Subroutine RITE is written in FORTRAN. This subroutine allows words to be printed from memory by the user's program. The subroutine is passed three parameters (the first byte address, the corresponding index in the NW array, and the number of bytes which are to be printed) by subroutine FIBM. Subroutine RITE stores the parameters it receives in the "BYTES" common block and then calls the routines needed to print the words in hexadecimal. (See flowchart p.46)

3.2.6 Subroutine EXIST

Subroutine EXIST is written COMPASS. The subroutine builds the necessary table and then calls the system macro\textsuperscript{[4]}, FILESTAT, to determine whether the file PGM exists. If PGM does not exist, the second word in the "IN" common block will be set to zero (Fig. 3.6). (See flowchart p.46)

3.2.7 Subroutine DEC

Subroutine DEC is written in COMPASS. This subroutine takes a word which contains four digits in
display code (right justified and zero filled) and converts the word to an integer. The word is passed to DEC as a parameter by the calling routine. (See flowchart p.46)

3.2.8 Subroutine CONVERT

Subroutine CONVERT is written in COMPASS. Two parameters (the number of characters to be converted and the address of the first word which is to be converted) are passed to subroutine CONVERT by the calling routine. CONVERT assumes that each word contains eight hexadecimal characters in display code (right justified and zero filled). The subroutine replaces the eight characters in each word with their thirty-two bit binary equivalent (right justified and zero filled). (See flowchart p.46)

3.2.9 Subroutine HEX

Subroutine HEX is written in COMPASS. The calling routine passes two parameters (the number of words to be converted and the address of the first word which is to be converted). Hex assumes that each word contains a thirty-two bit binary number (right justified and zero filled). Subroutine HEX replaces each number with its equivalent eight hexadecimal characters. These characters are stored right justified (and zero filled) in display code. (See flowchart p.46)
3.2.10 Subroutine FIBM

Subroutine FIBM is written in COMPASS. This subroutine is the routine which actually simulates the IBM 360. Three parameters are passed to FIBM by the main routine. The parameters passed are the starting address of the user's program, the number of bytes in the user's program, and the starting address of the NW array. The "STATUS" common block contains additional information which is used by FIBM. One of the words in the "STATUS" common block specifies how many instructions are to be executed by FIBM. Subroutine FIBM will continue to execute instructions until the specified number have been executed or until a condition occurs which would produce an interrupt in a real IBM 360.
(See flowchart pp.47 - 103)
4.1 Introduction

Subroutine FIBM is the heart of the Lehigh University IBM 360 Simulator. FIBM is the routine which actually simulates the IBM 360. Section 2 describes the memory of the simulated IBM 360. Section 3 describes how the PSW and the general registers are stored. Section 4 describes how the user's program is executed by subroutine FIBM. Section 5 discusses an alternative structure for FIBM.
4.2 The Memory of the Simulated IBM 360

The memory of the simulated IBM 360 is a 2500 word array called NW. Each word in the array holds one IBM 360 fullword. The thirty-two bit IBM 360 fullword is stored right justified in the sixty bit CDC word. The leftmost twenty-eight bits of each word are zero. To locate a byte in memory, FIBM first removes the rightmost two bits of the byte address. Then it right shifts the address two places and adds the contents of register B7 to it. Register B7 contains a number which when added to the shifted byte address gives that address' actual location in the CDC 6400. While subroutine FIBM is executing the contents of register B7 remains fixed at that number. In this manner, FIBM is able to locate the word which contains the desired byte. Finally FIBM uses the rightmost two bits which were originally removed from the byte address to determine which of the four bytes in the word is desired. Register B6 contains the smallest byte address which the simulator is allowed to read or write. Register B2 contains one plus the largest byte address which the simulator is allowed to read or write. The contents of these two registers remains fixed while FIBM is executing. FIBM checks to make sure that every byte address falls within the range set by the contents of these two registers.
4.3 The Storage of the PSW and the General Registers

The PSW is stored in the "STATUS" common block. Subroutine FIBM extracts the instruction address, the condition code, the ILC, and the program mask from the PSW and stores them at separate locations in the "STATUS" common block when it is called by the main routine. The locations containing the condition code and the program mask may be examined and their contents may be altered during the execution of the user's program. The contents of the location containing the ILC is altered every time an instruction is fetched by FIBM. While subroutine FIBM is executing, the contents of register XI contains the updated instruction address. Whenever FIBM terminates execution of the user's program, it stores the contents of register XI in the location reserved for the instruction address in the "STATUS" common block. Then it puts the current value of the condition code, the ILC, the program mask, and the instruction address in the location reserved for the second half of the PSW in the "STATUS" common block.

Each of the sixteen IBM 360 general registers is stored in the "STATUS" common block. The contents of each register occupies the rightmost thirty-two bits of a sixty bit location.
4.4 The Execution of the User's Program

When FIBM is called by the main routine, the subroutine extracts the instruction address, the condition code, the ILC, and the program mask from the PSW and stores them at separate locations in the "STATUS" common block. FIBM checks the protection key portion of the PSW to make sure that it is zero and sets the contents of registers X1, B2, B6, and B7. Next the subroutine subtracts one from the contents of a location (called STEP) in the "STATUS" common block which specifies the number of instructions which are to be executed. If the resulting contents is not equal to zero, then the subroutine fetches the halfword specified by the contents of register X1 (instruction address). FIBM examines the leftmost two bits to determine whether the halfword is an RR instruction, part of an RX instruction, part of an RS or SI instruction, or part of an SS instruction. Then FIBM branches according to the type of instruction to one of four sections in the subroutine. In these sections the remaining portion of the instruction is fetched if necessary, the contents of register X1 is updated, and all of the information necessary for the execution of the instruction is extracted from the instruction. Any needed byte addresses are generated from the address components contained in the instruction. Finally the op code is examined and a branch is made to the section...
of FIBM which actually performs the operation specified by the instruction. Once the operation has been performed, FIBM branches back to the section of the subroutine where one is subtracted from the contents of the location STEP in the "STATUS" common block. The process repeats itself until the contents of that location is zero, or unless a condition occurs which would produce an interrupt in a real IBM 360.

When the contents of location STEP is zero, FIBM terminates execution of the user's program, updates the PSW, and returns to the main routine. When a condition occurs which would produce an interrupt in a real IBM 360, FIBM terminates execution of the user's program, updates the PSW, return jumps to subroutine BOMB (to print the reason for interruption), and returns to the main routine.
4.5 An Alternative Structure for FIBM

One should note that it would be possible to determine the op code of the instruction when the halfword is initially fetched. The subroutine could then branch directly to the section which actually performed the operation specified by the instruction. Each section would fetch the remaining portion of the instruction if necessary, update the contents of register X1, extract all of the needed information from the instruction, and generate any needed byte addresses from the address components contained in the instruction. Undoubtedly this change in the subroutine would tend to decrease somewhat the time required to execute each instruction. However, the change would greatly increase the length of the subroutine and would not produce a significant difference in the time required to execute the user's program.
### 5.1 Notation

The following symbolic notations are used in the flowchart:

<table>
<thead>
<tr>
<th>Symbolic Notation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>name ← y</code></td>
<td>set 'name' equal to ( y )</td>
</tr>
<tr>
<td><code>C(Xi) ← b</code></td>
<td>set the contents of register ( Xi ) equal to ( b )</td>
</tr>
<tr>
<td><code>L(d) ← C(Xi)</code></td>
<td>put the contents of register ( Xi ) into location ( d )</td>
</tr>
<tr>
<td><code>+</code></td>
<td>arithmetic sum</td>
</tr>
<tr>
<td><code>*</code></td>
<td>arithmetic product</td>
</tr>
<tr>
<td><code>-</code></td>
<td>arithmetic difference</td>
</tr>
<tr>
<td><code>/</code></td>
<td>arithmetic division</td>
</tr>
<tr>
<td><code>.NOT.</code></td>
<td>boolean complement</td>
</tr>
<tr>
<td><code>.AND.</code></td>
<td>boolean AND</td>
</tr>
<tr>
<td><code>.OR.</code></td>
<td>boolean OR</td>
</tr>
<tr>
<td><code>.XOR.</code></td>
<td>boolean EXCLUSIVE OR</td>
</tr>
</tbody>
</table>
REWINDP

CALL EXIST

REWIND FILE "PEM"

NEOF ← 0

N'THERE() EQUAL 0

PRINT A MESSAGE THAT FILE DOES NOT EXIST

REQUEST

REWINDS

REWIND FILE "STORE"

REQUEST
SUBROUTINE RITE (IP1, IP2, IP3)

LOC ← IP2

INDEX ← IP2

BYTE ← IP1

CALL FILESTAT (INDEX, INDEX)

PRINT A HEADING

CALL PREDUMP

RETURN TO THE ROUTINE WHICH CALLED RITE

SUBROUTINE CONVERT

CONVERT THE SPECIFIC WORD FROM BINARY IN DISPLAY CODE TO BINARY. IF AN INVALID CHARACTER IS ENCOUNTERED DURING CONVERSION, SET INFAC EQUAL TO 1 AND RETURN JUMP TO SUBROUTINE BOMB

RETURN TO THE ROUTINE WHICH CALLED CONVERT

SUBROUTINE EXIST

CALL FILESTAT (INDEX, INDEX)

RETURN TO THE ROUTINE WHICH CALLED EXIST

SUBROUTINE DEC

CONVERT THE SPECIFIC WORD FROM DISPLAY CODE (BN FORMAT) TO BINARY. IF AN INVALID CHARACTER IS ENCOUNTERED DURING CONVERSION, SET INFAC EQUAL TO 20 AND RETURN JUMP TO SUBROUTINE BOMB

RETURN TO THE ROUTINE WHICH CALLED DEC

SUBROUTINE HEX

CONVERT THE SPECIFIED WORD FROM BINARY TO HEXADECIMAL IN DISPLAY CODE

RETURN TO THE ROUTINE WHICH CALLED HEX
SUBROUTINE

C(XI) ← INSTRUCTION
ADDRESS PORTION OF
THE PSW

L (EMASK) ← PROGRAM
MASK PORTION OF
THE PSW

L (CCODE) ← CONDITION
CODE PORTION OF
THE PSW

L (ILC) ← ILAC PORTION OF
THE PSW

PROTECTION
KEY PORTION
OF THE PSW
ZERO?

NO

KEYER

YES

L (ADDR) ← C(XI)

C(B6) ← FIRST BYTE
ADDRESS

C(B2) ← LAST BYTE
ADDRESS +1

C(B7) ← ACTUAL
STARTING
ADDRESS OF
THE BYTES

C(0) < C(B6)

YES

RANGE

NO

CHANGE

-47-
10ERR

NFLAG $\leftarrow 2$

$C(x6) \leftarrow 6$

WHYBOMB

ER

NFLAG $\leftarrow 3$

$C(x6) \leftarrow 6$

WHYBOMB

INSTRA

$C(\text{L(~LC)}) \leftarrow 0$

RANGE

-49-
\[ I \]

\[ L(c_{d+1} \oplus r_{e+1}) \]

\[ C(x_e) \rightarrow 0 \]

\[ C(x_e) \rightarrow 2 \]

\[ \text{REMOVE CARRY BIT} \]

\[ \text{SUM EQUAL ZERO?} \]

\[ \text{NO} \]

\[ \text{YES} \]

\[ \text{CHANGE} \]

\[ S \]

\[ C(e) \rightarrow 0 \]

\[ \text{OVERFLOW?} \]

\[ \text{YES} \]

\[ \text{NO} \]

\[ \text{CCC} \]

\[ -54- \]
$C(\times 6) \leftarrow 0$

$L(\text{CODE}) \leftarrow C(\times 6)$

$C(\times 6) \leftarrow 2$

$C_L(\text{CODE}) \leftarrow C(\times 6)$

$C(\times 6) \leftarrow 0$

$L(\text{CODE}) \leftarrow C(\times 6)$

$C(\times 6) \leftarrow 1$
BALR

\l(c(x)+REG)\n
\text{THE CURRENT VALUE OF THE SECOND HALF OF THE PSW}\n
\text{YES}\n\text{CHANGE}\n
C(x)\text{ EQUAL } 0 ?

\text{NO}\n
C(x) \leftarrow C(x2)

C(x) < C(8E) ?

\text{NO}\n\text{CHANGE}

\text{YES}\n\text{INSTRA}

SPM

\l(c(x)+\text{REG})\text{ IS RIGHT SHIFTED 24 PLACES}\n
\text{THE RIGHTMOST FOUR BITS OF } \l(c(x)+\text{REG})\n
\l(c(x)+\text{REG})\text{ IS RIGHT SHIFTED 4 PLACES}\n
\text{THE RIGHTMOST TWO BITS OF } \l(c(x)+\text{REG})\n
\text{CHANGE}
C(Y3) = 0
YES
C(X2) IS
RIGHT SHIFTED
16 BITS

NO
C(X2) ← THE
RIGHTMOST
16 BITS OF
C(X2) ARE
CONVERTED
TO A 32-BIT
FULL-WORD
C

ST

L(X2) ← C(X2 + P8)
CHANGE

STH

C(X7) ← RIGHTMOST
16 BITS OF
C(X7 + P8)

C(Y3) = 0
YES
FIRST
HALFWORD
OF THE
WORD
SPECIFIED
BY L(C43)

NO
SECOND
HALFWORD
OF THE
WORD
SPECIFIED
BY L(C43)
C(X7)
CHANGE

-63-
BAL

\( L(E(K+1) \to REG) \)

THE CURRENT VALUE OF THE SECOND HALF OF THE PSW

\( C(x) \leftarrow C(b_1) \)

\( C(b_1) < C(b_3) \) ?

YES

INSTRA

NO

CHANGE

LA

\( L(E(K+1) \to REG) \)

24 BIT ADDRESS SPECIFIED BY C(x3)

CHANGE

IC

RIGHTMOST EIGHT BITS (ONE BYTE) SPECIFIED BY C(x3) AND C(x2)

CHANGE
STC

EIGHT BIT LOCATION SPECIFIED BY C(B3) AND C(x3)

CHANGE

EX

FIRST HALFWORD OF INSTR. LOCATED AT ADDRESS SPECIFIED BY C(B3) AND C(x3)

C(x3)

EQUA 0

OP CODE

EQUAL 44 HEX

NO

NUMBER OF BYTES IN INSTRUCTION

C(B5) ← FIRST TWO BITS OF OP CODE

KILL

NO

YES

YES

-66-
RSSI

$C(x_1) + 3 \geq C(B2)$

NO

$C(x_1) \leftarrow C(x_1) + 4$

IMMEDIATE OPERAND

$C(x_3) \leftarrow$ IMMEDIATE OPERAND

DOES THE OP CODE START WITH 81H?

NO

OP CODE EQUAL 98HEX?

NO

OP CODE < 86HEX?

NO

OP CODE < 91HEX?

YES

SIM

NON

OP CODE > 91HEX?

YES

C(x3) \leftarrow THE NUMBER OF THE FIRST REGISTER

C(x3) \leftarrow THE NUMBER OF THE THIRD REGISTER

C(x2) \leftarrow ADDRESS OF SECOND OPERAND

BRANCH ACCORDING TO OP CODE

WRDD BXH BXLE SRL SLL SRA SLA SRDL SDL SIGA SIGB SIGC SIGD SIGE SIGF SIGG SIGH SIGI SIGJ SIGK SIGL SIGM SIGN SIGO SIGP SIGQ SIGR SIGS SIGT SIGU SIGV SIGW SIGX SIGY SIGZ SIGAA SIGAB SIGAC SIGAD SIGAE SIGAF SIGAG SIGAH SIGAI SIGAJ SIGAK SIGAL SIGAM SIGAN SIGAO SIGAP SIGAQ SIGAR SIGAS SIGAT SIGAU SIGAV SIGAW SIGAX SIGAY SIGAZ SIGBA SIGBB SIGBC SIGBD SIGBE SIGBF SIGBG SIGBH SIGBI SIGBJ SIGBK SIGBL SIGBM SIGBN SIGBO SIGBP SIGBQ SIGBR SIGBS SIGBT SIGBU SIGBV SIGBW SIGBX SIGBY SIGBZ SIGCA SIGCB SIGCC SIGCD SIGCE SIGCF SIGCG SIGCH SIGCI SIGCJ SIGCK SIGCL SIGCM SIGCN SIGCO SIGCP SIGCQ SIGCR SIGCS SIGCT SIGCU SIGCV SIGCW SIGCX SIGCY SIGCZ SIGDA SIGDB SIGDC SIGDD SIGDE SIGDF SIGDG SIGDH SIGDI SIGDJ SIGDK SIGDL SIGDM SIGDN SIGDO SIGDP SIGDQ SIGDR SIGDS SIGDT SIGDU SIGDV SIGDW SIGDX SIGDY SIGDZ SIGEA SIGEB SIGEC SIGED SIGEE SIGEF SIGEG SIGEH SIGEI SIGEJ SIGEK SIGEL SIGEM SIGEN SIGEO SIGEP SIGER SIGES SIGET SIGEU SIGEV SIGEW SIGEX SIGEY SIGEZ SIGFA SIGFB SIGFC SIGFD SIGFE SIGFF SIGFG SIGFH SIGFI SIGFJ SIGFK SIGFL SIGFM SIGFN SIGFO SIGFP SIGFQ SIGFR SIGFS SIGFT SIGFU SIGFW SIGFX SIGFY SIGFZ SIGGA SIGGB SIGGC SIGGD SIGGE SIGGF SIGGG SIGGH SIGGI SIGGJ SIGGK SIGGL SIGGM SIGGN SIGGO SIGGP SIGGQ SIGGR SIGGS SIGGT SIGGU SIGGW SIGGX SIGGY SIGGZ SIGHA SIGHB SIGHC SIGHD SIGHE SIGHF SIGHG SIGHI SIGHJ SIGHK SIGHL SIGHM SIGHN SIGHO SIGHP SIGHQ SIGHS SIGHT SIGHU SIGHW SIGHX SIGHY SIGHz SIGIA SIGIB SIGIC SIGID SIGIE SIGIF SIGIG SIGIH SIGIJ SIGIK SIGIL SIGIM SIGIN SIGIO SIGIP SIGIQ SIGIR SIGIS SIGIT SIGIU SIGIW SIGIX SIGIY SIGIZ SIGJA SIGJB SIGJC SIGJD SIGJE SIGJF SIGJJ SIGJK SIGJL SIGJM SIGJN SIGJO SIGJP SIGJQ SIGJR SIGJS SIGJT SIGJU SIGJW SIGJX SIGJY SIGJZ SIGKA SIGKB SIGKC SIGKD SIGKE SIGKF SIGKG SIGKH SIGKI SIGKJ SIGKK SIGKL SIGKM SIGKN SIGKO SIGKP SIGKQ SIGKR SIGKS SIGKT SIGKV SIGKW SIGKX SIGKY SIGKZ SIGLA SIGLB SIGLC SIGLD SIGLE SIGLF SIGLG SIGLI SIGLJ SIGLK SIGLL SIGLM SIGLN SIGLO SIGLP SIGLQ SIGLR SIGLS SIGLT SIGLU SIGLV SIGLX SIGLY SIGLZ SIGMA SIGMB SIGMC SIGMD SIGME SIGMF SIGMG SIGMH SIGMI SIGMJ SIGMK SIGML SIGMM SIGMN SIGMO SIGMP SIGMQ SIGMR SIGMS SIGMT SIGMU SIGMV SIGMW SIGMX SIGMY SIGMZ SIGNA SIGNB SIGNC SIGND SIGNE SIGNF SIGNG SIGNH SIGNI SIGNJ SIGNK SIGNL SIGNM SIGNO SIGNP SIGNQ SIGNR SIGNS SIGNT SIGNU SIGNV SIGNW SIGNX SIGNY SIGNZ SIGOA SIGOB SIGOC SIGOD SIGOE SIGOF SIGOG SIGOH SIGOI SIGOJ SIGOK SIGOL SIGOM SIGON SIGOO SIGOP SIGOQ SIGOR SIGOS SIGOT SIGOU SIGOV SIGOW SIGOX SIGOY SIGOZ SIGPA SIGPB SIGPC SIGPD SIGPE SIGPF SIGPG SIGPH SIGPI SIGPJ SIGPK SIGPL SIGPM SIGPN SIGPO SIGPQ SIGPR SIGPS SIGPT SIGPU SIGPV SIGPW SIGPX SIGPY SIGPZ SIGQA SIGQB SIGQC SIGQD SIGQE SIGQF SIGQG SIGQH SIGQI SIGQJ SIGKK SIGQK SIGQL SIGQM SIGQO SIGQQ SIGQR SIGQS SIGQT SIGQU SIGQV SIGQW SIGQX SIGQY SIGQZ SIGRA SIGRB SIGRC SIGRD SIGRE SIGRF SIGRG SIGRH SIGRI SIGRJ SIGRK SIGRL SIGRM SIGRN SIGRO SIGRP SIGRQ SIGRR SIGRS SIGRT SIGRU SIGRV SIGRW SIGRX SIGRY SIGRZ SIGSA SIGSB SIGSC SIGSD SIGSE SIGSF SIGSG SIGSH SIGSI SIGSJ SIGSK SIGSL SIGSM SIGSN SIGSO SIGSP SIGSQ SIGSR SIGSS SIGST SIGSU SIGSW SIGSX SIGSY SIGSZ SIGTA SIGTB SIGTC SIGTD SIGTE SIGTF SIGTG SIGTH SIGTI SIGTJ SIGTK SIGTL SIGTM SIGTN SIGTO SIGTP SIGTQ SIGTR SIGTS SIGTU SIGTW SIGTX SIGTY SIGTZ SIGUA SIGUB SIGUC SIGUD SIGUE SIGUF SIGUG SIGUH SIGUI SIGUJ SIGUK SIGUL SIGUM SIGUN SIGUO SIGUP SIGUQ SIGUR SIGUS SIGUT SIGUU SIGUW SIGUX SIGUY SIGUZ SIGVA SIGVB SIGVC SIGVD SIGVE SIGVF SIGVG SIGVH SIGVI SIGVJ SIGVK SIGVL SIGVM SIGVN SIGVO SIGVP SIGVQ SIGVR SIGVS SIGVT SIGVU SIGVW SIGVX SIGVY SIGVZ SIGWA SIGWB SIGWC SIGWD SIGWE SIGWF SIGWG SIGWH SIGWI SIGWJ SIGWK SIGWL SIGWM SIGWN SIGWO SIGWP SIGWQ SIGWR SIGWS SIGWT SIGWU SIGWV SIGWX SIGWY SIGWZ SIGXA SIGXB SIGXC SIGXD SIGXE SIGXF SIGXG SIGXH SIGXI SIGXJ SIGXK SIGXL SIGXM SIGXN SIGXO SIGXP SIGXQ SIGXR SIGXS SIGXT SIGXU SIGXW SIGXX SIGXY SIGXZ SIGYA SIGYB SIGYC SIGYD SIGYE SIGYF SIGYG SIGYH SIGYI SIGYJ SIGYK SIGYL SIGYM SIGYN SIGYO SIGYP SIGYQ SIGYR SIGYS SIGYT SIGYU SIGYW SIGYX SIGYY SIGYZ SIGZA SIGZB SIGZC SIGZD SIGZE SIGZF SIGZG SIGZH SIGZI SIGZJ SIGZK SIGZL SIGZM SIGZN SIGZO SIGZP SIGZQ SIGZR SIGZS SIGZT SIGZU SIGZV SIGZW SIGZX SIGZY SIGZZ

-69-
```
SLDA

C(x3) AN EVEN NUMBER?

NO REGEN

YES

C(x2) \rightarrow \text{RIGHTMOST SIX BITS OF } C(x3)

C(b3) \rightarrow 0

C(x7) \rightarrow 1

EVEN ODD REGISTER PAIR SPECIFIED BY C(x3)

THE 63 INTEGER BITS OF THE EVEN ODD REGISTER PAIR SPECIFIED BY C(x3) ARE LEFT SHIFTED C(x2) PLACES

CCC

C(x7) EQUAL 0?

NO

YES

C(b3) \rightarrow 1

C(x7) \rightarrow 1

C(x7) \rightarrow 0

SRDA

C(x3) AN EVEN NUMBER?

NO REGEN

YES

C(x2) \rightarrow \text{RIGHTMOST SIX BITS OF } C(x3)

C(b3) \rightarrow 0

CCC

C(x7) EQUAL 0?

NO

YES

C(x7) \rightarrow 1

C(x7) \rightarrow 0

NO

```
SRDL
CK(x2) AN EVEN NUMBER
YES
CK(x2) = RIGHTMOST SIX BITS OF CK(x2)

EVEN AND ODD REGISTER PAIR SPECIFIED BY CK(x3)
THE CK(x2) BIT OF THE EVEN ODD REGISTER PAIR SPECIFIED BY CK(x3) ARE RIGHT SHIFTED CK(x2) PLACES

CHANGE

BXH
CK(x3) ← CK(x3)+ REG

IS CK(x3) AN EVEN NUMBER?
YES
CK(x3) ← CK(x3)+ REG

L(CAS) ← L(CAS) + CK(x3)+ REG

IS CK(CAS) > CK(x3)?
YES
CK(x3) ← CK(x3)+ REG

CHANGE

IS CK(x2) < CK(B6)?
YES
CK(x1) ← CK(x2)
1. **KILL**
   - NFLAG ← 10
   - C(x, 0) ← 3
   - WHYBOMB

2. **RECEIVE**
   - NFLAG ← 11
   - C(x, 0) ← 6
   - WHYBOMB

3. **DIVBADM**
   - NFLAG ← 12
   - C(x, 0) ← 9
   - WHYBOMB

-84-
LOCATE THE NEXT FIRST OPERAND BYTE AND LOCATE THE NEXT SECOND OPERAND BYTE

C(B4) < C(B2) ?

C(B4) = THE LARGER OF THE TWO BYTE ADDRESSES
C(B4) = THE LARGER OF THE TWO BYTE ADDRESSES

C(B4) ≥ C(B2) ?

M V Z

SMALLEST OF THE TWO BYTE ADDRESSES

C(B6)

C(B4) ← C(B4) + 1
C(B4) ← C(B4) - 1

NO

MVZ

C(X0) ← MASK (11110000 binary) PROPERLY SHIFTED TO LINE UP WITH FIRST OPERAND BYTE
C(X7) ← THE FIRST HALF OF THE SECOND OPERAND BYTE PROPERLY SHIFTED TO LINE UP WITH THE FIRST OPERAND BYTE

C(X3) EQUAL 0 ?

C(X3) ← C(X3) - 1
C(X3) ← C(X3) + 1

NO

CHANGE

RANGE

C(B4) ≥ C(B2) ?

YES

NO

RANGE

-90-
CLC

L(0) ← 0

THE SMALLEST OF THE TWO BYTE ADDRESSES
C(x) <

YES

RANGE

C(x) ← THE LARGER OF THE TWO BYTE ADDRESSES

C(x) ≥ C(x)

YES

C(x) ← C(x) + 1

NO

LOCATE THE NEXT FIRST OPERAND BYTE AND LOCATE THE NEXT SECOND OPERAND BYTE

CHANGE

YES

C(x) ← C(x) - 1

NO

C(x) ← 2

NO

C(x) ← 1

YES

THE FIRST OPERAND BYTE GREATER THAN THE SECOND OPERAND BYTE

NO

C(x) ← C(x)

YES

L(0) ← C(x)

CHANGE
The diagram illustrates a process involving numbers and operations. It starts with a decision point asking if a double-word address is specified. If yes, it proceeds to a range check; if no, it goes to doubt. The range check involves comparing conditions and applying logical operations to determine the outcome. The flowchart includes operations such as changing the sign of a number, inserting digits, and determining if certain values are equal or have specific conditions.
DEC1

NFLAG ← 16

C(XC) ← 1

WRRDD

NFLAG ← 18

C(XC) ← 1

SIM

NFLAG ← 19

C(XC) ← 1

SK

NFLAG ← 17

C(XC) ← 1

WRRDD

WRRDD

WRRDD

WRRDD

-101-
OUT

LPSW+1
THE CURRENT VALUE OF THE SECOND HALF OF THE PSW

L(IADDR) ← C(X)

RETURN TO THE ROUTINE WHICH CALLED FIRM
VI    Improving LUIS

6.1 Introduction

While the Lehigh University IBM 360 Simulator is quite useful in its present form, there are a number of changes which could be made to improve it. Section 2 describes how more memory could be added to the simulated IBM 360. Section 3 describes how the decimal and floating-point feature instructions could be added to the simulator. Section 4 suggests a modification to LUIS which would enable the user to select whether the simulator sent normal or abbreviated messages to the user. Section 5 describes how additional commands could be added to the simulator.
6.2 Adding More Memory

The maximum storage capacity of an IBM 360 is 16,777,216 decimal byte addresses. Unfortunately the CDC 6400 can only provide the user with a maximum of 40,960 decimal sixty bit storage locations. Therefore it is necessary to limit the size of the memory of the simulated IBM 360 to a small fraction of the maximum capacity. Since byte addresses are often placed into eighteen bit B registers by subroutine FIBM, byte addresses must be restricted to a maximum length of seventeen significant bits. This fact restricts the largest possible byte address to $131,071_{\text{decimal}}$.

LUIS allows the user to specify the portion of storage which will be used. The user may specify any starting address between 000000 and 01D8A8 hexadecimal (121,000 decimal). The user's portion of storage may contain a maximum of ten thousand bytes. While the size of the simulated memory is currently limited to ten thousand bytes, this limit could be increased. By increasing the dimension of the NW array and by modifying the DEC subroutine so that an integer larger than 9,999 could be read, the maximum size of the simulated memory could be increased. Naturally, even if this was done, it would still be necessary to limit the largest possible byte address to a number less than $131,072_{\text{decimal}}$. 

-105-
6.3 Adding More Instructions

The decimal feature instructions are not available in LUIS. However, one could easily add these instructions to LUIS. Currently a request for any of these instructions causes subroutine FIBM to branch to a section of the subroutine called DECI. DECI causes an error message to be printed and terminates execution of the user's program. One could add additional sections to FIBM which would perform the operations specified by the decimal feature instructions. Subroutine FIBM could then be changed so that a request for a decimal feature instruction would cause FIBM to branch to the appropriate section where the operation would be performed.

Currently a request for any of the floating-point feature instructions causes FIBM to branch to a section of the subroutine called FLOAT. FLOAT causes an error message to be printed and terminates execution of the user's program. One could add additional sections to FIBM which would perform the operations specified by the floating-point feature instructions. Sections RR and RX of subroutine FIBM would have to be modified to include branches to the sections of FIBM which would perform the floating-point operations. Naturally it would be necessary to add floating-point registers to LUIS. These registers could be storage locations in the "STATUS" common block.
6.4 Abbreviated Messages From LUIS

Whenever the user must supply some information to the simulator, the simulator specifies the exact nature and format of the information required. Usually the message is in the form of a one or two line sentence. These messages are especially useful to the person who has not used the simulator previously. Unfortunately these messages can limit the speed with which an experienced user can use the simulator. This fact is especially true when using a teletype.

For the experienced user, a two or three word phrase would provide sufficient information so that the user would know what type of information the simulator required. The simulator could be modified to provide two additional requests. One request would cause the simulator to go into the abbreviated mode where all messages sent to the user would be in an abbreviated form. Another request would place the simulator in the normal mode. In the normal mode all messages would be printed by the simulator in the manner currently being used.
6.5 Additional Simulator Commands

At the present time there are eleven different commands which the user may issue when the simulator is in the request mode. The main routine uses a series of IF statements to branch to the section of the routine which handles the particular request. Additional commands could be added to the simulator. Additional IF statements would be added to the present series which would check for the new commands. These IF statements would then branch to sections of the main routine which would handle the new commands.
References


Appendix A

The following material is reproduced from the manual, IBM System/360 Principles of Operation. This material should be of value to those who are unfamiliar with the IBM 360 structure.

Instruction Format

The length of an instruction format can be one, two, or three halfwords. It is related to the number of storage addresses necessary for the operation. An instruction consisting of only one halfword causes no reference to main storage. A two-halfword instruction provides one storage-address specification; a three-halfword instruction provides two storage-address specifications. All instructions must be located in storage on integral boundaries for halfwords. Figure 13 shows five basic instruction formats.

The five basic instruction formats are denoted by the format codes nn, nx, ns, si, and ss. The format codes express, in general terms, the operation to be performed. nn denotes a register-to-register operation; nx, a register-and-indexed-storage operation; ns, a register-and-storage operation; si, a storage and immediate-operand operation; and ss, a storage-to-storage operation. An immediate operand is one contained within the instruction.

For purposes of describing the execution of instructions, operands are designated as first and second operands and, in the case of branch-on-index instructions, third operands. These names refer to the manner in which the operands participate. The operand to which a field in an instruction format applies is generally denoted by the number following the code name of the field, for example, R1, B1, L4, D9.

In each format, the first instruction halfword consists of two parts. The first byte contains the operation code (op code). The length and format of an instruction are specified by the first two bits of the operation code.
The second byte is used either as two 4-bit fields or as a single eight-bit field. This byte can contain the following information:

- Four-bit operand register specification (R1, R2, or R3)
- Four-bit index register specification (X1)
- Four-bit mask (M1)
- Four-bit operand length specification (L1 or L2)
- Eight-bit operand length specification (L)
- Eight-bit byte of immediate data (I)

In some instructions a four-bit field or the whole second byte of the first halfword is ignored.

The second and third halfwords always have the same format:

- Four-bit base register designator (B1 or B2), followed by a 12-bit displacement (D1 or D2).
Address Generation

For addressing purposes, operands can be grouped in three classes: explicitly addressed operands in main storage, immediate operands placed as part of the instruction stream in main storage, and operands located in the general or floating-point registers.

To permit the ready relocation of program segments and to provide for the flexible specifications of input, output, and working areas, all instructions referring to main storage have been given the capacity of employing a full address.

The address used to refer to main storage is generated from the following three binary numbers:

- **Base Address (B)** is a 24-bit number contained in a general register specified by the program in the B field of the instruction. The B field is included in every address specification. The base address can be used as a means of static relocation of programs and data. In array-type calculations, it can specify the location of an array and, in record-type processing, it can identify the record. The base address provides for addressing the entire main storage. The base address may also be used for indexing purposes.

- **Index (X)** is a 24-bit number contained in a general register specified by the program in the X field of the instruction. It is included only in the address specified by the nx instruction format. The nx format instructions permit double indexing; i.e., the index can be used to provide the address of an element within an array.

- **Displacement (D)** is a 12-bit number contained in the instruction format. It is included in every address computation. The displacement provides for relative addressing up to 4095 bytes beyond the element or base address. In array-type calculations the displacement can be used to specify one of many items associated with an element. In the processing of records, the displacement can be used to identify items within a record.

In forming the address, the base address and index are treated as unsigned 24-bit positive binary integers. The displacement is similarly treated as a 12-bit positive binary integer. The three are added as 24-bit binary numbers, ignoring overflow. Since every address includes a base, the sum is always 24 bits long. The address bits are numbered 8-31 corresponding to the numbering of the base address and index bits in the general register.

-112-
The program may have zeros in the base address, index, or displacement fields. A zero is used to indicate the absence of the corresponding address component. A base or index of zero implies that a zero quantity is to be used in forming the address, regardless of the contents of general register 0. A displacement of zero has no special significance. Initialization, modification, and testing of base addresses and indexes can be carried out by fixed-point instructions, or by BRANCH AND LINK, BRANCH ON COUNT, or BRANCH-ON-INDEX instructions.

As an aid in describing the logic of the instruction format, examples of two instructions and their related instruction formats follow.

**RR Format**

<table>
<thead>
<tr>
<th>Add</th>
<th>7</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>7</td>
<td>13</td>
</tr>
</tbody>
</table>

Execution of the `ADD` instruction adds the contents of general register 9 to the contents of general register 7 and the sum of the addition is placed in general register 7.

**RX Format**

<table>
<thead>
<tr>
<th>Store</th>
<th>3</th>
<th>10</th>
<th>14</th>
<th>300</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>7</td>
<td>13</td>
<td>19</td>
<td>31</td>
</tr>
</tbody>
</table>

Execution of the `STORE` instruction stores the contents of general register 3 at a main-storage location addressed by the sum of 300 and the low-order 24 bits of general registers 14 and 10.
Sequential Instruction Execution

Normally, the operation of the CPU is controlled by instructions taken in sequence. An instruction is fetched from a location specified by the instruction address in the current rsw. The instruction address is then increased by the number of bytes in the instruction fetched to address the next instruction in sequence. The instruction is then executed and the same steps are repeated using the new value of the instruction address.

Conceptually, all halfwords of an instruction are fetched from storage after the preceding operation is completed and before execution of the current operation, even though physical storage word size and overlap of instruction execution with storage access may cause actual instruction fetching to be different. Thus, it is possible to modify an instruction in storage by the immediately preceding instruction.

A change from sequential operation may be caused by branching, status switching, interruptions, or manual intervention.

Branching

The normal sequential execution of instructions is changed when reference is made to a subroutine, when a two-way choice is encountered, or when a segment of coding, such as a loop, is to be repeated. All these tasks can be accomplished with branching instructions. Provision is made for subroutine linkage, permitting not only the introduction of a new instruction address but also the preservation of the return address and associated information.

Decision-making is generally and symmetrically provided by the BRANCH ON CONDITION instruction. This instruction inspects a two-bit condition code that reflects the result of a majority of the arithmetic, logical, and I/O operations. Each of these operations can set the code in any one of four states, and the conditional branch can specify any selection of these four states as the criterion for branching. For example, the condition code reflects such conditions as nonzero, first operand high, equal, overflow, channel busy, zero, etc. Once set, the condition code remains unchanged until modified by an instruction that reflects a different condition code.

The two bits of the condition code provide for four possible condition code settings: 0, 1, 2, and 3. The specific meaning of any setting is significant only to the operation setting the condition code.
List of Instructions by Set and Feature

Standard Instruction Set

<table>
<thead>
<tr>
<th>NAME</th>
<th>MNEMONIC</th>
<th>TYPE</th>
<th>CODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add</td>
<td>AR</td>
<td>RR C</td>
<td>1A</td>
</tr>
<tr>
<td>Add</td>
<td>A</td>
<td>RX C</td>
<td>5A</td>
</tr>
<tr>
<td>Add Halfword</td>
<td>AH</td>
<td>RX C</td>
<td>4A</td>
</tr>
<tr>
<td>Add Logical</td>
<td>ALR</td>
<td>RR C</td>
<td>1E</td>
</tr>
<tr>
<td>Add Logical</td>
<td>AL</td>
<td>RX C</td>
<td>5E</td>
</tr>
<tr>
<td>AND</td>
<td>NR</td>
<td>RR C</td>
<td>14</td>
</tr>
<tr>
<td>AND</td>
<td>N</td>
<td>RX C</td>
<td>54</td>
</tr>
<tr>
<td>AND</td>
<td>NI</td>
<td>SI C</td>
<td>94</td>
</tr>
<tr>
<td>AND</td>
<td>NC</td>
<td>SS C</td>
<td>D4</td>
</tr>
<tr>
<td>Branch and Link</td>
<td>BALR</td>
<td>RR</td>
<td>05</td>
</tr>
<tr>
<td>Branch and Link</td>
<td>BAL</td>
<td>RX</td>
<td>45</td>
</tr>
<tr>
<td>Branch on Condition</td>
<td>BCR</td>
<td>RR</td>
<td>07</td>
</tr>
<tr>
<td>Branch on Condition</td>
<td>BC</td>
<td>RX</td>
<td>47</td>
</tr>
<tr>
<td>Branch on Count</td>
<td>BCTR</td>
<td>RR</td>
<td>06</td>
</tr>
<tr>
<td>Branch on Count</td>
<td>BCT</td>
<td>RX</td>
<td>46</td>
</tr>
<tr>
<td>Branch on Index</td>
<td>BXH</td>
<td>RS</td>
<td>86</td>
</tr>
<tr>
<td>Branch on Index</td>
<td>BXLE</td>
<td>RS</td>
<td>87</td>
</tr>
<tr>
<td>Compare</td>
<td>CR</td>
<td>RR C</td>
<td>19</td>
</tr>
<tr>
<td>Compare</td>
<td>C</td>
<td>RX C</td>
<td>59</td>
</tr>
<tr>
<td>Compare Halfword</td>
<td>CH</td>
<td>RX C</td>
<td>49</td>
</tr>
<tr>
<td>Compare Logical</td>
<td>CLR</td>
<td>RR C</td>
<td>15</td>
</tr>
<tr>
<td>Compare Logical</td>
<td>CL</td>
<td>RX C</td>
<td>55</td>
</tr>
<tr>
<td>Compare Logical</td>
<td>CLC</td>
<td>SS C</td>
<td>D5</td>
</tr>
<tr>
<td>Compare Logical</td>
<td>CLI</td>
<td>SI C</td>
<td>95</td>
</tr>
<tr>
<td>Convert to Binary</td>
<td>CVB</td>
<td>RX</td>
<td>4F</td>
</tr>
<tr>
<td>Convert to Decimal</td>
<td>CVD</td>
<td>RX</td>
<td>4E</td>
</tr>
<tr>
<td>Diagnose</td>
<td>SI</td>
<td></td>
<td>83</td>
</tr>
<tr>
<td>Divide</td>
<td>DR</td>
<td>RR</td>
<td>1D</td>
</tr>
<tr>
<td>Divide</td>
<td>D</td>
<td>RX</td>
<td>5D</td>
</tr>
<tr>
<td>Exclusive OR</td>
<td>XR</td>
<td>RR C</td>
<td>17</td>
</tr>
<tr>
<td>Exclusive OR</td>
<td>X</td>
<td>RX C</td>
<td>57</td>
</tr>
<tr>
<td>Exclusive OR</td>
<td>XI</td>
<td>SI C</td>
<td>97</td>
</tr>
<tr>
<td>Exclusive OR</td>
<td>XC</td>
<td>SS C</td>
<td>D7</td>
</tr>
<tr>
<td>Execute</td>
<td>EX</td>
<td>RX</td>
<td>44</td>
</tr>
<tr>
<td>Halt I/O</td>
<td>HIO</td>
<td>SI C</td>
<td>9E</td>
</tr>
<tr>
<td>Insert Character</td>
<td>IC</td>
<td>RX</td>
<td>43</td>
</tr>
<tr>
<td>Load</td>
<td>LR</td>
<td>RR</td>
<td>18</td>
</tr>
<tr>
<td>Load</td>
<td>L</td>
<td>RX</td>
<td>58</td>
</tr>
<tr>
<td>Load Address</td>
<td>LA</td>
<td>RX</td>
<td>41</td>
</tr>
<tr>
<td>Load and Test</td>
<td>LTR</td>
<td>RR C</td>
<td>12</td>
</tr>
<tr>
<td>Load Complement</td>
<td>LCR</td>
<td>RR C</td>
<td>13</td>
</tr>
<tr>
<td>Load Halfword</td>
<td>LH</td>
<td>RX</td>
<td>48</td>
</tr>
<tr>
<td>Load Multiple</td>
<td>LM</td>
<td>RS</td>
<td>86</td>
</tr>
<tr>
<td>Load Negative</td>
<td>LNR</td>
<td>RR C</td>
<td>11</td>
</tr>
<tr>
<td>Load Positive</td>
<td>LPR</td>
<td>RR C</td>
<td>10</td>
</tr>
<tr>
<td>Load FSW</td>
<td>LFSW</td>
<td>SI L</td>
<td>82</td>
</tr>
</tbody>
</table>

-115-
<table>
<thead>
<tr>
<th>NAME</th>
<th>MNEMONIC</th>
<th>TYPE</th>
<th>CODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Move</td>
<td>MVI</td>
<td>SI</td>
<td>92</td>
</tr>
<tr>
<td>Move</td>
<td>MVC</td>
<td>SS</td>
<td>D2</td>
</tr>
<tr>
<td>Move Numerics</td>
<td>MNV</td>
<td>SS</td>
<td>D1</td>
</tr>
<tr>
<td>Move with Offset</td>
<td>MVO</td>
<td>SS</td>
<td>F1</td>
</tr>
<tr>
<td>Move Zones</td>
<td>MVZ</td>
<td>SS</td>
<td>D3</td>
</tr>
<tr>
<td>Multiply</td>
<td>MR</td>
<td>RR</td>
<td>1C</td>
</tr>
<tr>
<td>Multiply</td>
<td>M</td>
<td>RX</td>
<td>5C</td>
</tr>
<tr>
<td>Multiply Halfword</td>
<td>MH</td>
<td>RX</td>
<td>4C</td>
</tr>
<tr>
<td>OR</td>
<td>OR</td>
<td>RR C</td>
<td>16</td>
</tr>
<tr>
<td>OR</td>
<td>O</td>
<td>RX C'</td>
<td>56</td>
</tr>
<tr>
<td>OR</td>
<td>OI</td>
<td>SI C</td>
<td>96</td>
</tr>
<tr>
<td>OR</td>
<td>OC</td>
<td>SS C</td>
<td>D8</td>
</tr>
<tr>
<td>Pack</td>
<td>PACK</td>
<td>SS</td>
<td>F2</td>
</tr>
<tr>
<td>Set Program Mask</td>
<td>SPM</td>
<td>RR L</td>
<td>04</td>
</tr>
<tr>
<td>Set System Mask</td>
<td>SSM</td>
<td>SI</td>
<td>80</td>
</tr>
<tr>
<td>Shift Left Double</td>
<td>SLDA</td>
<td>RS C</td>
<td>8F</td>
</tr>
<tr>
<td>Shift Left Single</td>
<td>SLA</td>
<td>RS C</td>
<td>8B</td>
</tr>
<tr>
<td>Shift Left Double</td>
<td>SLDL</td>
<td>RS</td>
<td>8D</td>
</tr>
<tr>
<td>Logical</td>
<td>SLL</td>
<td>RS</td>
<td>89</td>
</tr>
<tr>
<td>Shift Right Double</td>
<td>SRDA</td>
<td>RS C</td>
<td>8E</td>
</tr>
<tr>
<td>Shift Right Single</td>
<td>SRA</td>
<td>RS C</td>
<td>8A</td>
</tr>
<tr>
<td>Shift Right Double</td>
<td>SRDL</td>
<td>RS C</td>
<td>8C</td>
</tr>
<tr>
<td>Logical</td>
<td>SRL</td>
<td>RS</td>
<td>88</td>
</tr>
<tr>
<td>Start I/O</td>
<td>SIO</td>
<td>SI C</td>
<td>9C</td>
</tr>
<tr>
<td>Store</td>
<td>ST</td>
<td>RX</td>
<td>50</td>
</tr>
<tr>
<td>Store Character</td>
<td>STC</td>
<td>RX</td>
<td>42</td>
</tr>
<tr>
<td>Store Halfword</td>
<td>STH</td>
<td>RX</td>
<td>40</td>
</tr>
<tr>
<td>Store Multiple</td>
<td>STM</td>
<td>RS</td>
<td>90</td>
</tr>
<tr>
<td>Subtract</td>
<td>SR</td>
<td>RR C</td>
<td>1B</td>
</tr>
<tr>
<td>Subtract</td>
<td>S</td>
<td>RX C</td>
<td>5B</td>
</tr>
<tr>
<td>Subtract Halfword</td>
<td>SH</td>
<td>RX C</td>
<td>4B</td>
</tr>
<tr>
<td>Subtract Logical</td>
<td>SLR</td>
<td>RR C</td>
<td>1F</td>
</tr>
<tr>
<td>Subtract Logical</td>
<td>SL</td>
<td>RX C</td>
<td>5F</td>
</tr>
<tr>
<td>Supervisor Call</td>
<td>SVC</td>
<td>RR</td>
<td>0A</td>
</tr>
<tr>
<td>Test and Set</td>
<td>TS</td>
<td>SI C</td>
<td>93</td>
</tr>
<tr>
<td>Test Channel</td>
<td>TCH</td>
<td>SI C</td>
<td>9F</td>
</tr>
<tr>
<td>Test I/O</td>
<td>TIO</td>
<td>SI C</td>
<td>9D</td>
</tr>
<tr>
<td>Test Under Mask</td>
<td>TM</td>
<td>SI C</td>
<td>91</td>
</tr>
<tr>
<td>Translate</td>
<td>TR</td>
<td>SS DC</td>
<td></td>
</tr>
<tr>
<td>Translate and Test</td>
<td>TRT</td>
<td>SS C</td>
<td>DD</td>
</tr>
<tr>
<td>Unpack</td>
<td>UNPK</td>
<td>SS</td>
<td>F3</td>
</tr>
</tbody>
</table>

Note: A "C" in the TYPE column means that the condition code is set. An "L" in the TYPE column means that a new condition code is loaded.
LEHIGH UNIVERSITY IBM 360 SIMULATOR

ENTER THE SMALLEST ADDRESS IN YOUR PROGRAM AS AN EIGHT DIGIT HEXADECIMAL NUMBER.
00000000

ENTER THE MAXIMUM SIZE OF YOUR PROGRAM (IN BYTES) AS A FOUR DIGIT DECIMAL INTEGER.
0080

ENTER THE PSW AS A 16 DIGIT HEXADECIMAL NUMBER.
0000000000000000

REQUEST=INSERT

ENTER THE STARTING ADDRESS (OF THE COLLECTION OF BYTES) AS AN EIGHT DIGIT HEXADECIMAL NUMBER.
00000000

ENTER THE NUMBER OF BYTES WHICH WILL BE INSERTED AS A FOUR DIGIT DECIMAL INTEGER.
0072

IF YOU HAVE THE BYTES ON THE FILE PGM, TYPE PGM.
IF YOU ARE GOING TO ENTER THE BYTES, TYPE INPUT.
PGM

THE BYTES HAVE BEEN READ

REQUEST=DUMP

TYPE ALL OR PARTIAL ALl

00000 05709C00 00004 702E5810 00008 70365910 0000C 703A47A0
00010 70145810 00014 703A5910 00018 703E47A0 0001C 70205810
00020 703E5010 00024 70429C01 00028 703247F0 0002C 70000000
00030 00037036 00034 00017042 00038 00000023 0003C 00000034
00040 00000012 00044 00000034 00048 00000000 0004C 00000000

REQUEST=EXECUTE

YOUR PROGRAM WANTS TO READ SOME WORDS

IF YOU HAVE THE WORDS ON THE FILE PGM, TYPE PGM.
IF YOU ARE GOING TO ENTER THE WORDS, TYPE INPUT.

INPUT

ENTER THE WORDS. (ONE WORD PER LINE)

00038 -0000004C
0003C -00000039
00040 -0000000D

WORDS READ INTO MEMORY BY THE USER'S PROGRAM

00038 0000004C
WORDS PRINTED FROM MEMORY BY THE USER'S PROGRAM

00044 0000004C
YOUR PROGRAM WANTS TO READ SOME WORDS
IF YOU HAVE THE WORDS ON THE FILE PGM, TYPE PGM.
IF YOU ARE GOING TO ENTER THE WORDS, TYPE INPUT.
INPUT
ENTER THE WORDS. (ONE WORD PER LINE)

00038 -00000004
0003C -0000000A
00040 -000000FF

WORDS READ INTO MEMORY BY THE USER'S PROGRAM

00038 00000004
0003C 0000000A
00040 000000FF

WORDS PRINTED FROM MEMORY BY THE USER'S PROGRAM

00044 000000FF
YOUR PROGRAM WANTS TO READ SOME WORDS
IF YOU HAVE THE WORDS ON THE FILE PGM, TYPE PGM.
IF YOU ARE GOING TO ENTER THE WORDS, TYPE INPUT.
INPUT
ENTER THE WORDS. (ONE WORD PER LINE)

00038 -00000000
0003C -00000000
00040 -000ZZZZW

WORDS READ INTO MEMORY BY THE USER'S PROGRAM

00038 00000000
0003C 00000000
00040 000ZZZZW

-118-
***** AN INVALID CHARACTER APPEARED IN A HEXADECIMAL NUMBER. THE BYTES WERE REMOVED FROM MEMORY

REQUEST=STATUS

PROGRAM STATUS WORD

0000000090000006

INSTRUCTION ADDRESS 0006
CONDITION CODE 1
ILC 2

GENERAL REGISTERS

0 00000000 1 000000FF 2 00000000 3 00000000
4 00000000 5 00000000 6 00000000 7 40000000
8 00000000 9 00000000 A 00000000 B 00000000
C 00000000 D 00000000 E 00000000 F 00000000

REQUEST=STORE
TYPE ALL OR PARTIAL
ALL
THE BYTES HAVE BEEN DUMPED ONTO FILE STORE
REQUEST=BYE
EXIT

-119-
Vita

Leonard I. Horey, son of Helen and Henry Horey, was born on January 19, 1951 in Newark, New Jersey. He did his undergraduate work at Lehigh University and in 1973 received the Degree of Bachelor of Science in Electrical Engineering with highest honors.

The author was a teaching assistant in the Department of Electrical Engineering throughout his graduate program.