SOFTWARE OPERATIONS MANUAL OF A
COMPUTERIZED BEEF GRADING INSTRUMENT

by

DON A. GILLILAND

B.S., Kansas State University, 1977

A MASTER'S REPORT

Submitted in partial fulfillment of the
requirements for the degree

MASTER OF SCIENCE

Department of Electrical Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas
1982

Approved by

[Signature]
Major Professor
THE FOLLOWING DOCUMENT(S) IS OF POOR LEGIBILITY IN THE ORIGINAL

THIS IS THE BEST COPY AVAILABLE
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>List of Figures</strong></td>
<td>iv</td>
</tr>
<tr>
<td><strong>Acknowledgement</strong></td>
<td>vi</td>
</tr>
<tr>
<td><strong>Section 1: General Information</strong></td>
<td></td>
</tr>
<tr>
<td>1.1 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Memory mapping</td>
<td>6</td>
</tr>
<tr>
<td>1.3 Program hierarchy</td>
<td>12</td>
</tr>
<tr>
<td><strong>Section 2: Meat Grading Programs</strong></td>
<td></td>
</tr>
<tr>
<td>2.1 Master loop (MASTER)</td>
<td>14</td>
</tr>
<tr>
<td>2.2 Next picture (NEXPIC)</td>
<td>19</td>
</tr>
<tr>
<td>2.3 Preprocessor (PREPRO)</td>
<td>22</td>
</tr>
<tr>
<td>2.4 Classifier (CLSIFR)</td>
<td>26</td>
</tr>
<tr>
<td>2.5 Boundary search (BOUSCH)</td>
<td>31</td>
</tr>
<tr>
<td>2.6 Spline (SPLINE)</td>
<td>43</td>
</tr>
<tr>
<td>2.7 Area acquisition (AREAAQ)</td>
<td>58</td>
</tr>
<tr>
<td>2.8 Maximum point search (MAXMAIN)</td>
<td>63</td>
</tr>
<tr>
<td>2.9 Fat thickness (FTTHCM)</td>
<td>71</td>
</tr>
<tr>
<td>2.10 Squeeze-display (KRMDSP)</td>
<td>82</td>
</tr>
<tr>
<td>2.11 Box distribution (BOXDIS)</td>
<td>86</td>
</tr>
<tr>
<td>2.12 Fat Pieces (FATPCS)</td>
<td>93</td>
</tr>
<tr>
<td>2.13 Calculate results (CLCRES)</td>
<td>106</td>
</tr>
<tr>
<td>2.14 Display results (DSPRES)</td>
<td>113</td>
</tr>
<tr>
<td>2.15 Save results (SAVRES)</td>
<td>117</td>
</tr>
<tr>
<td>2.16 Draw box (DRAWBOX)</td>
<td>123</td>
</tr>
</tbody>
</table>
### Section 3: Utility Programs

| 3.1 | Block display (BLKDPY) | 127 |
| 3.2 | Distribution (DISTRB) & Print (PTDIST) | 135 |
| 3.3 | Light meter (LMETER) | 141 |
| 3.4 | Communication to Nova (COMMOV) | 147 |
| 3.5 | Set date & location (SETDAT) | 150 |
| 3.6 | Light calibration (LFTCAL) | 153 |
| 3.7 | Line load (LINLOD) | 158 |
| 3.8 | Exclusive - or utility (XOR) | 160 |
| 3.9 | Classifier level change (CLLVCH) | 163 |
| 3.10 | Recipe (RECIPE) | 166 |

### Section 4: Disk Support Package Routines

| 4.1 | Disk support package (SUBS) | 170 |
| 4.2 | Initialize data disk (INTDAT) | 176 |
| 4.3 | Disk format (FORMAT) | 179 |
| 4.4 | Binary read from disk (DSKBIN) | 184 |
| 4.5 | Read/write disk picture (RDSDK) | 190 |

### References:

References: 195

### Appendices:

- **A.1** Test patterns 196
- **A.2** Start-up procedure 198
- **A.3** Procedure for making a new program disk 201
<table>
<thead>
<tr>
<th>Figure Numbers</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1-1</td>
<td>Meat surface cross section</td>
<td>5</td>
</tr>
<tr>
<td>1.2-1</td>
<td>Memory map</td>
<td>7</td>
</tr>
<tr>
<td>1.3-1</td>
<td>Program hierarchy</td>
<td>13</td>
</tr>
<tr>
<td>2.3-1</td>
<td>Picture memory vs. address from PREPRO</td>
<td>23</td>
</tr>
<tr>
<td>2.4-1</td>
<td>Picture memory</td>
<td>26</td>
</tr>
<tr>
<td>2.5-1</td>
<td>Boundary conditions</td>
<td>31</td>
</tr>
<tr>
<td>2.5-2</td>
<td>Isolated fat cell</td>
<td>32</td>
</tr>
<tr>
<td>2.5-3</td>
<td>Non-isolated fat cell</td>
<td>32</td>
</tr>
<tr>
<td>2.5-4</td>
<td>Direction box</td>
<td>32</td>
</tr>
<tr>
<td>2.5-5</td>
<td>Box overlayed onto memory cells</td>
<td>33</td>
</tr>
<tr>
<td>2.5-6</td>
<td>Center fat cell to boundary</td>
<td>33</td>
</tr>
<tr>
<td>2.5-7</td>
<td>New box location</td>
<td>34</td>
</tr>
<tr>
<td>2.5-8</td>
<td>Too small boundary</td>
<td>35</td>
</tr>
<tr>
<td>2.5-9</td>
<td>Too small boundary (Block Display)</td>
<td>35</td>
</tr>
<tr>
<td>2.5-10</td>
<td>Correct boundary</td>
<td>36</td>
</tr>
<tr>
<td>2.5-11</td>
<td>Correct boundary (Block Display)</td>
<td>36</td>
</tr>
<tr>
<td>2.6-1</td>
<td>Spline before CONNT</td>
<td>44</td>
</tr>
<tr>
<td>2.6-2</td>
<td>Spline after CONNT, REST and BOUSCH</td>
<td>44</td>
</tr>
<tr>
<td>2.6-3</td>
<td>Before spline</td>
<td>46</td>
</tr>
<tr>
<td>2.6-4</td>
<td>Spline iteration 1</td>
<td>46</td>
</tr>
<tr>
<td>2.6-5</td>
<td>Spline iteration 2</td>
<td>47</td>
</tr>
<tr>
<td>2.6-6</td>
<td>Spline iteration 3</td>
<td>47</td>
</tr>
<tr>
<td>2.6-7</td>
<td>Spline iteration 4</td>
<td>48</td>
</tr>
<tr>
<td>2.6-8</td>
<td>Spline iteration 5, no change</td>
<td>48</td>
</tr>
<tr>
<td>2.9-1</td>
<td>Right sloped loin eye</td>
<td>71</td>
</tr>
<tr>
<td>2.9-2</td>
<td>Left sloped loin eye</td>
<td>71</td>
</tr>
<tr>
<td>Section</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>--------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>2.9-3</td>
<td>Minimum fat thickness</td>
<td>72</td>
</tr>
<tr>
<td>2.9-4</td>
<td>Fat thickness points</td>
<td>73</td>
</tr>
<tr>
<td>2.9-5</td>
<td>Fat thickness points (Block Display)</td>
<td>74</td>
</tr>
<tr>
<td>2.9-6</td>
<td>Approximate fat thickness location</td>
<td>75</td>
</tr>
<tr>
<td>2.11-1</td>
<td>Boxes for box distribution</td>
<td>88</td>
</tr>
<tr>
<td>2.12-1</td>
<td>Fat pieces, no fat boundary drawn</td>
<td>95</td>
</tr>
<tr>
<td>2.12-2</td>
<td>Fat pieces fat boundary drawn</td>
<td>96</td>
</tr>
<tr>
<td>2.12-3</td>
<td>Fat pieces (Squeeze Display)</td>
<td>97</td>
</tr>
<tr>
<td>2.14-1</td>
<td>Empty DPYBUF output</td>
<td>114</td>
</tr>
<tr>
<td>2.14-2</td>
<td>Sample data in DPYBUF output</td>
<td>114</td>
</tr>
<tr>
<td>3.1-1</td>
<td>Block display addresses</td>
<td>128</td>
</tr>
<tr>
<td>3.2-1</td>
<td>Output print distribution</td>
<td>137</td>
</tr>
<tr>
<td>3.3-1</td>
<td>Light meter scales</td>
<td>143</td>
</tr>
<tr>
<td>A.1-1</td>
<td>Test pattern #2</td>
<td>197</td>
</tr>
<tr>
<td>A.1-2</td>
<td>Test pattern #3</td>
<td>197</td>
</tr>
</tbody>
</table>
Acknowledgement

The author wishes to express his deep appreciation to Professor Don Lenhert of the Electrical Engineering Department. The author also wishes to thank Tim Anderson and Kevin Williams for their hardware and software contributions, Dr. Dell Allen of the Animal Science Department, and Robert Young for his development of the XASM86 cross-assembler.
Introduction

1.1 The Kansas State University Meat Grader is basically an image processing system. The documentation for the system consists of three manuals titled: Operating and Hardware Service Manual Section 1-8, Software Operations Manual, and Operating and Hardware Service Manual Section 9. This manual, Software Operations, describes all software for the system as well as providing flow-charts and summary sheets for each program. Each summary sheet contains the subroutines, registers and variables used by the routine. Routine inputs and outputs are indicated. The version number is located at the bottom of the summary sheet and should match the corresponding assembly code version number located at the top of the code listing. For further details refer to the assembly listing for each program.

Section 1 describes some of the system software organization. The memory map and system program hierarchy are explained.

Section 2 contains the beef grading programs which include some display and computational modules. Although this section contains routines which are seemingly not beef grading e.g. Save results, the routines were included because they are the basis for a minimum software beef grading system.

Section 3 contains the utility programs to facilitate data analysis. The utility programs allow the system operator to 1) display picture data; 2) plot area as a function of light intensity; 3) adjust the average light intensity; 4) provide communication to a Nova 4 computer system; 5) enter date and location information for data storage; 6) successively store one line of picture data for diagnostics; 7) quickly check purged data and 8) allow classifier level changing.
Section 4 contains the floppy disk support package for off line storage of data and/or pictures.

Shown in Fig. 1.1-1 is a typical cross section of the meat surface showing the various muscle (lettered areas), bone and fat locations [4]. It may be helpful to refer to the figure when reading some sections of the manual.

A typical grading session is described below to give the operator some insight into the software procedures.

The video camera is held a fixed distance and perpendicular to the cut through the loin eye muscle and surrounding tissue and the button on the handgrip is pressed. The analog video image is digitized (analog to digital conversion) and entered into high or picture memory via command from a picture loading program called Next picture. After storing the image, the processing begins. The picture data is classed as lean meat, fat, or background based on its light reflectance. A boundary is drawn in picture memory totally or partially around the outside of the longissimus dorsi (loin eye) muscle and a determination is made as to whether the eye is left or right sloped. An attempt is made to eliminate those muscles or bounded areas that should not be included. The total lean and fat area is found and stored for later calculation. Depending on whether a right or left loin eye is indicated the fat thickness measurement is taken on the appropriate right or left side and stored. At the present time the fat thickness location depends upon operator positioning of the cross hair targets at the proper location within the loin eye.

The operator at this time may view the progress of the algorithm by calling in the Squeeze-display program. The program compresses the 240 row x 248 column data point picture memory for display on 24 row x 80 column data terminal.
A section of picture memory expected to be within the loin eye is sampled for marbling data, where the fat, lean and number of fat pieces is determined and stored.

All the intermediate calculations are stored until the next image is processed. Potentially the yield grade and quality grade could be calculated.

The system user is encouraged to become familiar with the 8086 microprocessor architecture as well as the 86/12 monitor and monitor commands. Knowledge of the registers and how they are used will be invaluable for any future modifications by the user [1].

A short listing is given below;

<table>
<thead>
<tr>
<th>Destination Index</th>
<th>DI - usually used as picture pointer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source Index</td>
<td>SI - general purpose</td>
</tr>
<tr>
<td></td>
<td>CX - usually used as counter, general purpose</td>
</tr>
<tr>
<td></td>
<td>BX - general purpose</td>
</tr>
<tr>
<td></td>
<td>DX - general purpose</td>
</tr>
<tr>
<td>Accumulator</td>
<td>AX - general purpose</td>
</tr>
<tr>
<td>Base pointer</td>
<td>BP - used as memory pointer, general purpose</td>
</tr>
<tr>
<td>Code Segment register</td>
<td>CS - usually set to zero</td>
</tr>
<tr>
<td>Data Segment register</td>
<td>DS - usually zero, 1000 or 2000</td>
</tr>
<tr>
<td>Extra Segment register</td>
<td>ES - usually zero or 40</td>
</tr>
<tr>
<td>Stack Segment register</td>
<td>SS - usually zero</td>
</tr>
</tbody>
</table>

Addresses to picture memory are given in row-column form, e.g., 7E6FH is row 7E, column 6F. Throughout the manual, if a number is followed by an H, it is a Hexadecimal Number (base 16).
Knowledge of the 86/12 monitor will be helpful in debugging. A short description follows:

<table>
<thead>
<tr>
<th>Keys</th>
<th>Name</th>
<th>Example</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>Display</td>
<td>D7000,73FF</td>
<td>Memory is displayed from 7000H to 73FFH</td>
</tr>
<tr>
<td>M</td>
<td>Move</td>
<td>M1000:400,7FFF,400</td>
<td>Memory is moved from 1000H on high memory locations 400H to 7FFFH to low memory starting at location 400H</td>
</tr>
<tr>
<td>S</td>
<td>Substitute</td>
<td>S1E00,00 FF</td>
<td>Substitute FFH for 00H in location 1E00H</td>
</tr>
<tr>
<td>G</td>
<td>GO</td>
<td>G1000,1100</td>
<td>Execute at location 1000H until 1100H break point</td>
</tr>
</tbody>
</table>

**NOTE:** If the breakpoint is never reached a CC will be stored at the breakpoint location. This must be removed for proper program execution.

<table>
<thead>
<tr>
<th>Keys</th>
<th>Name</th>
<th>Example</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>Next</td>
<td>N 1000</td>
<td>Execute the instruction at location 1000H. A typed comma single steps to the next instruction</td>
</tr>
<tr>
<td>O</td>
<td>Output</td>
<td>OCE, 82</td>
<td>Output an 82H to output port CEH</td>
</tr>
<tr>
<td>I</td>
<td>Input</td>
<td>IC2,</td>
<td>Input a value from port C2H.</td>
</tr>
</tbody>
</table>
TECHNICAL NAME
A. Obliquus abdominis externus
B. Diaphragma
C. Serratus dorsalis, posterior
D. Longissimus costarum
E. Longissimus lumborum
F. Multifidus dorsi
G. Spinalis dorsi

FIG. 1.1-1. Meat Surface Cross Section.

1. Rib
2. Thoracic vertebra
The Memory map gives information relating to the locations of all programs, data and variables in the computer system, Fig. 1.2-1. Remember, this is program memory, not picture memory which is located much higher in the system architecture. The addresses are listed on the left by hexadecimal numbers from top of program memory to bottom. The program name is next followed by the shortened mnemonic that corresponds to the program. This mnemonic is used for the XASM86 Assembler as a shortened program name for convenience purposes. Listed with the mnemonic is a symbol. To run a program from the Master loop program only that symbol need be typed in. Variable names are represented by the following fields; variable name: (description), program using the variable, next program using the variable. The isolated input-output (I/O) port is shown at the bottom of the memory map. The setting and clearing of these bits control the focus and flood lights as well as communication to the data terminal.
<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>NAME</th>
<th>SYMBOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>9000-97FF</td>
<td>FIRMWARE-BOOTSTRAP ON EPROM</td>
<td></td>
</tr>
<tr>
<td>7A00-7CFF</td>
<td>SPLINE REDUCTION</td>
<td>SPLINE</td>
</tr>
<tr>
<td>7500-7517</td>
<td>TRACK SET VAR FOR R/W/PIC ON DISK</td>
<td>X</td>
</tr>
<tr>
<td>7200-79FA</td>
<td>SPLINE ADDR STOR (OK OVERWRIT A/DV)</td>
<td></td>
</tr>
<tr>
<td>7100-71FF</td>
<td>DATA BUFFER FOR INDAT</td>
<td></td>
</tr>
<tr>
<td>7030-7050</td>
<td>RECIPE FOR MASTERLOOP</td>
<td>RECIPE</td>
</tr>
<tr>
<td>7000-</td>
<td>MASTER LOOP STRING</td>
<td>STRING</td>
</tr>
<tr>
<td>6200-627D</td>
<td>SET DATE AND LOCATION</td>
<td>SETDAT</td>
</tr>
<tr>
<td>6100-6183</td>
<td>INITIALIZE DATA DISK</td>
<td>INTDAT</td>
</tr>
<tr>
<td>6000-60E7</td>
<td>FORMAT DISK</td>
<td>FORMAT</td>
</tr>
<tr>
<td>5700-57FF</td>
<td>BUFFER FOR READ BIN</td>
<td></td>
</tr>
<tr>
<td>5300-53CA</td>
<td>BINARY READ VARIABLES</td>
<td></td>
</tr>
<tr>
<td>5000-524B</td>
<td>READ BIN DISK</td>
<td>DSKBIN</td>
</tr>
<tr>
<td>4D00-4D36</td>
<td>LINE LOAD UTILITY</td>
<td>LINLOAD</td>
</tr>
<tr>
<td>4800-4D00</td>
<td>LIGHT METER</td>
<td>LMETER</td>
</tr>
<tr>
<td>4600-4748</td>
<td>BOX DISTRIBUTION</td>
<td>BOXDIS</td>
</tr>
<tr>
<td>4400-45A3</td>
<td>FATBOUND SUBROUTINE</td>
<td>FATBDY</td>
</tr>
<tr>
<td>4200-42FB</td>
<td>FAT PIECES</td>
<td>FATPCS</td>
</tr>
<tr>
<td>4180-41F9</td>
<td>DRECTN SUBROUTINE</td>
<td>DRECTN</td>
</tr>
<tr>
<td>4000-415D</td>
<td>FAT THICKNESS</td>
<td>FTHCM</td>
</tr>
<tr>
<td>3D20-3D2F</td>
<td>TRANSFER CORR TABLE</td>
<td>TRSCOR</td>
</tr>
<tr>
<td>3000-3D1E</td>
<td>CLOCKWISE TABLE</td>
<td>CWTLB</td>
</tr>
<tr>
<td>3000-3C1E</td>
<td>COUNTERCLOCKWISE TABLE</td>
<td>CCWTBL</td>
</tr>
<tr>
<td>3800-382A</td>
<td>FIND MAX SUBROUTINE</td>
<td>FINDM</td>
</tr>
<tr>
<td>3A00-3A9E</td>
<td>NEXTPOINT SUBROUTINE</td>
<td>NEXTPT</td>
</tr>
<tr>
<td>3900-39BF</td>
<td>DISTANCE SUBROUTINE</td>
<td>DISTAN</td>
</tr>
<tr>
<td>3800-3842</td>
<td>MAX POINT SEARCH</td>
<td>MXMAIN</td>
</tr>
<tr>
<td>3600-3773</td>
<td>PRINT DISTRIBUTION</td>
<td>PTDIST</td>
</tr>
<tr>
<td>3580-35CD</td>
<td>DISTRIBUTION</td>
<td>DISTRB</td>
</tr>
<tr>
<td>3400-34E8</td>
<td>AREA AQUISITION</td>
<td>AREAAQ</td>
</tr>
<tr>
<td>3340-33CF</td>
<td>SQUEEZE DISPLAY</td>
<td>KRMDSP</td>
</tr>
<tr>
<td>3100-3302</td>
<td>BOUNDARY SEARCH</td>
<td>BOUSCH</td>
</tr>
<tr>
<td>3000-3050</td>
<td>CLASSIFIER</td>
<td>CLSIFR</td>
</tr>
<tr>
<td>2F00-2F17</td>
<td>MOVPOS SUBROUTINE, BOUSCH</td>
<td></td>
</tr>
<tr>
<td>2B00-2E8C</td>
<td>CALCULATE RESULTS</td>
<td>CLCRES</td>
</tr>
<tr>
<td>2A20-2A31</td>
<td>PAUSE</td>
<td>PAUSE</td>
</tr>
<tr>
<td>2A00-2A10</td>
<td>TEMPORARY STOP</td>
<td>TMPSTP</td>
</tr>
<tr>
<td>2700-275F</td>
<td>NEXT PICTURE</td>
<td>NEXPIC</td>
</tr>
<tr>
<td>2600-268C</td>
<td>DISPLAY RESULTS</td>
<td>DSPRES</td>
</tr>
<tr>
<td>2550-25D0</td>
<td>DRAW A BOX IN MEMORY</td>
<td>DRWBOX</td>
</tr>
<tr>
<td>2528-2545</td>
<td>MESSAGES FOR DISPIC</td>
<td></td>
</tr>
<tr>
<td>247B-248B</td>
<td>NON SEQ WRITE PICTURE</td>
<td>NSWRDSK</td>
</tr>
<tr>
<td>2434-245C</td>
<td>WRITE DISK PICTURE</td>
<td>WRSK</td>
</tr>
<tr>
<td>2400-2433</td>
<td>DISK READ PICTURE</td>
<td>RRSK</td>
</tr>
<tr>
<td>2200-2270</td>
<td>MESSAGES FOR SAVE RESULTS</td>
<td></td>
</tr>
</tbody>
</table>
SAVE RESULTS
LIGHT CALIBRATION
PREPROCESSOR 64K
MESSAGE FOR MASTER LOOP, PROGRAM?
OFFSET BACK MASTER
CNT1:METER
GET STRING ENTRY PT
MASTER LOOP
DISK CONTROL PROGRAMS

VBLES FOR DSK SUBS,HEX,DECTBL,NUM,TEMP
CLASSIFIER LEVEL CH MESSAGES
CLASSIFIER LEVEL CHANGE CLLVCH
BLOCK DISPLAY PICTURE BLKDPY
M:CLCRES
KFHH:CLCRES
WGTH:CLCRES
MAPCNT:CLCRES
LESCNT:CLCRES
GRTCNT:CLCRES
EQUCNT:CLCRES
SGBCNT:CLCRES
FATTH:CLCRES
NPCS:CLCRES
PFATB<PERCENT FAT IN BOX>:CLCRES
OF29  PMEAT:(PERCENT MEAT),CLCRES
OF27  PFAT:(PCTFAT),CLCRES
OF25  AMEAT:(AREA OF MEAT),CLCRES
OF23  AFAT:(AREA OF FAT),CLCRES
OF21  AREA:(AREA HU),CLCRES
OF1F  SIGL:(SIGNIFICANT BOX LEVEL),CLCRES
OF1D  BXAREA:(BOX AREA),CLCRES
OF1B  DELL:(DELTA LOWER),CLCRES
OF19  DELU:(DELTA UPPER),CLCRES
OF17  TEMP:(TEMPORARY STORAGE),CLCRES
OF16  ASCBUF:(ASCII BUFFER),DSPRES
;

OF12  CTRL:DUMMY:,SUBS
OF11  CNT3:
OF10  SGBT:,CLCRES
OF0F  CNT2:,SUBS
OF0D  CNT1:,SUBS
OF0C  DRV:,SUBS
OF0A  ADDR:,SUBS
OF08  BSEADDR:,SUBS
OF06  FIRST:,SUBS
OF05  LEN:,SUBS
OF04  TRK1:,SUBS
OF03  NUM:,SUBS
OF02  SEC:,SUBS
OF01  TRK:,SUBS
OF00  COMMAND:,SUBS
0D0A  PROFLG:
0D09  SAVB:,FTHICM
0D07  SAVG:,FTHICM
0D05  SUMDST:,FTHICM
0D03  BKGD POINT ROW ADDR
0D02  BKGD POINT COL ADDR
0D01  EDGE BNDY PT. ROW ADDR
0D00  EDGE BNDY PT. COL ADDR
0CC0  RESERVED FOR DISPLAY RESULTS
;

OC34  DPLYBUF:(DISPLAY STRING),DSPRES
;

OA56  PCKDEC:(PACKED DECIMAL STRING),DSPRES
     ,SARES,INTDAT,CLCRES
0A00
09FE  SAVCNT:SARES,INTDAT,SETDAT
09FC  LCON1:SARES,INTDAT,SETDAT
09FA  DAT2:SARES,INTDAT,SETDAT
09F8  DAT:SARES,INTDAT,SETDAT
0940  HEX VALUE STRING FOR CALCUL
;

0900  FREAD:,CLCRES
08D8  MFSQU:(MEAT FAT IN SQUARE),CLCRES
08D6 MEATSQ: (MEAT IN SQUARE), CLCRES, AREAAD
08D4 FATSR: (FAT IN THE SQUARE), CLCRES, AREAAD
08D2 SMLFLG: CLCRES, DRWBOX, CLSIFR, AREAAD
08D0 LCRS1, DRWBOX, BOUSCH, FTHCM
08C8 OLDOUN: (BOUNDARY WRITE OVER LOC), BOUSCH
08C6 LURLV2: (LOWER LEVEL 2), CLSIFR
08CA LURLV1: (LOWER LEVEL 1), CLSIFR
08C8 URRLV2: (UPPER LEVEL 2), CLSIFR
08C8 URRLV1: (UPPER LEVEL 1), CLSIFR
08C6 ECTRL: (LGCAL, PREPRO, FTHCM)
08C4 LMSPK: LGCAL, PREPRO, FTHCM
08C2 LMFLG: LGCAL, PREPRO, FTHCM
08C0 LMRCAL: LGCAL, PREPRO, FTHCM
0810-08A0 COMMUNICATION TO NOVA COMNOV W
0718 FATCNT: (FATCOUNT), CLSIFR
0716 MEATC: (MEAT COUNT), CLSIFR
0714 BKGDCNT: CLSIFR, AREAAD, BOXDIS
0712 CFLG:
0710 MEACNTP: (MEAT COUNT POS SLOPE), FTHCM
070E FATCNT: (FAT COUNT NEG SLOPE), FTHCM
070C SLEN: LENGTH OF SPLINE), SPLINE
070A STMP: (START POINT SPLINE), SPLINE
0708 SFPLG: (SPLINE FLAG), SPLINE
0706 SPLTD: (SPLINE TAIL DIRECTION), SPLINE
0704 SPLT: (SPLINE TAIL ADDRESS), SPLINE
0702 SPLHD: (SPLINE HEAD DIRECTION), SPLINE
0700 SPLH: (SPLINE HEAD ADDRESS), SPLINE
06B1;
0672 BXDISV: (BOX DISTRIBUTION), CLCRES, BOXDIS
0672 BXDISV: BOX DISTRIBUTION, CLCRES, BOXDIS
0670 BXSIZE: ROW, CLCRES, BOXDIS
064E KPH: (PERCENT KIDNEY), CLCRES
064C WEIGHT: CLCRES
064A FMPFLG: LME
0649;
0640 ERROR COUNT:
0639 OLDVAT: (OLD VALUE), CLCRES
0637 EQNAT: CLCRES
0633 NOPCS: (NUMBER OF PIECES), FATPCS, CLCRES
062E EQFT: CLCRES
062A FTTENS: CLCRES
0628 FATEA: CLCRES
0626 EQKPH: CLCRES
0622 EQWEI: CLCRES
0620 FTHRUG: (RUN_FLAG), FTHCM, CLSIFR, KRDSP
061C EFRAG: (ERROR LOC FOR FATBDY), FATPCS
061A MTF: (CHANGED MEAT TO FAT), BOUSCH
0619 FAT THICKNESS: FTHCM
0618 FAT THICKNESS: NUMM1, FTHCM
0617 FAT THICKNESS: MSD, FTHCM
0616 BKGDPNT ROW ADDR: FTHCM
0615 BKGDPNT COL ADDR: FTHCM
0614  EDGE BNDY PT, ROW ADDR:, FTHICM
0613  EDGE BNDY PT, COL ADDR:, FTHICM
0612  MAX DIS STORAGE:, MXMAIN
0611  MAX DIS STORAGE:, MXMAIN
0610  MAX DIS STORAGE MSD:, MXMAIN
060F  MVBLE ROW MAX DIS PT J A:, MXMAIN, AREAQQ
060E  MVBLE COL MAX DIS PT J B:, MXMAIN, AREAQQ
060D  STNRY ROW MAX DIS PT J B:, MXMAIN, AREAQQ
060C  STNRY COL MAX DIS PT J B:, MXMAIN, AREAQQ
060B  MCHIGH: (MEAT COUNT HIGH), CLCRES, AREAQQ
060A  MCLOW: (MEAT COUNT LOW), CLCRES, AREAQQ
0609  FCHIGH: (FAT COUNT HIGH), CLCRES, AREAQQ
0608  FCLOW: (FAT COUNT LOW), CLCRES, AREAQQ
0607  BROW: (BOT ROW ADR), BOXDIS, FATPCS, AREAQQ
0606  BCOL: (BOT COL ADR), BOXDIS, FATPCS, AREAQQ
0605  TROW: (TOP ROW ADR), BOXDIS, FATPCS, AREAQQ
0604  TCOL: (TOP COL ADR), BOXDIS, FATPCS, AREAQQ
0603  RROW: (RGT ROW ADR), BOXDIS, FATPCS, AREAQQ
0602  RCOL: (RGT COL ADR), BOXDIS, FATPCS, AREAQQ
0601  LROW: (LFT ROW ADR), BOXDIS, FATPCS, AREAQQ
0600  LCOL: (LFT COL ADR), BOXDIS, FATPCS, AREAQQ
05FF  RESERVED BLOCK FOR LSD’S:, DISTRB
;
;
0500  RESERVED BLOCK FOR MSD’S:, DISTRB
! THIS AREA WILL BE WRITTEN OVER IF
! THE DISTRIBUTION PROGRAM IS RUN
0400  ALSO USED BY BOUNDARY SEARCH.
011A  ADR: , NEXPIC

PORT ASSIGNMENTS

CB
OUTPUT PORT
BIT 0 - PICTURE
BIT 1 - FLOOD
BIT 2 - FOCUS

CA
INPUT PORT
BIT 0

CE
STATUS CONTROL REGISTER USUAL SET 02

DE
COMM PORT STATUS-CONSOLE

DB
COMM PORT DATA-CONSOLE

END
Program Hierarchy

1.3 The Beef grading, utility and disk support programs are semi-independent. Certain programs can be run at any time (Set date & location) while others require a particular order to have meaningful results.

The program hierarchy chart is shown in Fig. 1.3-1. The chart gives an indication of the proper order for program execution. For each block the preceding block must at least be run for outputs to have meaningful data.

For example, the Squeeze-display program must be preceded by the running of the Classifier program. Squeeze display can also be run after Fat thickness or any program following Classifier. Note that Squeeze-display cannot be run directly following Next picture.
FIG. 1.3-1. Program Hierarchy
Master Loop (MASTER)

2.1 Master loop is designed to input a string of symbols or letters from the keyboard corresponding to the programs to be executed and then executes each one in order. When the string of programs has completed execution Master loop continuously repeats the string except with the Q (Temporary Stop) or L (Pause) command.

Master loop starts by calling subroutine INIT to initialize the string pointer to -1 (FFFF Hex). Next the routine GETSTR inputs the string of symbols. Each symbol is converted to a number (A=1, B=2, etc.) and stored in the string called STR. The program continues inputting symbols until a [CR] is pressed. Then a zero is stored to terminate STR, and the string pointer is zeroed. If an error is made on the keyboard, a rubout should be typed. The RUBOUT routine causes the pointer to decrement thus ignoring the error.

The first value in STR is loaded and multiplied by two. Now the number corresponds to a location in the jump table TBL. The values in TBL are the starting addresses of each program. An indirect call is performed to call that program. When control returns to Master loop the string pointer is incremented and this process is repeated. A program called BACK is executed when the string pointer points to a value of zero in STR. BACK resets the string pointer so that the next program executed is the first value in STR. An example of Master loop response is shown below.

G 2434- 38 1E00

Programs: PVABXDGHCISROK [CR]
Master Loop (MASTER)

SUBROUTINES: GETSTR, MESSAGE, CHARIN, RUBOUT, CHAROT, INIT

REGISTERS: BP - input string pointer
BX - calling program address
SI - message pointer
DI - string pointer
AX - general purpose, counter, input, output

VARIABLES: 1F82H-1FB6H TBL
1FB8H-1FBFH MESS1
1F80H CNTL
7000H STR

INPUTS: ASCII characters from data terminal

OUTPUT: Indirect calls to stored programming

LOCATION: 1E00H

SIZE: 0B1H

VERSION: 2/12/81
Next Picture Routine (NEXPIC)

2.2 NEXPIC turns on the focus lights then waits for the picture taking button to be pushed on either the computer enclosure or the camera handle. Once the button has been depressed NEXPIC turns off the focus lights, turns on the flash and after the light has reached full intensity NEXPIC then activates the digitizer to take a picture. Upon the completion of the picture storage by the direct memory access (DMA) board this program turns off all lights then returns.

If while NEXPIC is waiting for the picture button a [CNTL C] is pressed on the keyboard the program returns control back to the monitor routine. This gives an escape point from the string of programs presently executing.

The warmup time for the flood light was chosen to be 30 frames (about one second). This allows the lamp to reach a constant color temperature prior to storing the picture.

NOTE-

<table>
<thead>
<tr>
<th>PORT A IS OUTPUT</th>
<th>PORT B IS INPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>bit 0</td>
<td>bit 0</td>
</tr>
<tr>
<td>G1 and G2</td>
<td>picture taking button</td>
</tr>
<tr>
<td>bit 1</td>
<td>bit 1</td>
</tr>
<tr>
<td>focus lamp control</td>
<td>VSYNC</td>
</tr>
<tr>
<td>bit 2</td>
<td>bit 2</td>
</tr>
<tr>
<td>flash lamp control</td>
<td>n/c</td>
</tr>
<tr>
<td>bit 3</td>
<td>bit 3</td>
</tr>
<tr>
<td>n/c</td>
<td>n/c</td>
</tr>
<tr>
<td>bit 4</td>
<td>bit 4</td>
</tr>
<tr>
<td>n/c</td>
<td>n/c</td>
</tr>
<tr>
<td>bit 5</td>
<td>bit 5</td>
</tr>
<tr>
<td>n/c</td>
<td>n/c</td>
</tr>
<tr>
<td>bit 6</td>
<td>bit 6</td>
</tr>
<tr>
<td>n/c</td>
<td>n/c</td>
</tr>
<tr>
<td>bit 7</td>
<td>bit 7</td>
</tr>
<tr>
<td>n/c</td>
<td>n/c</td>
</tr>
</tbody>
</table>

where n/c means no connection
Next Picture (NEXPIC)

SUBROUTINES: none

REGISTERS: AX - Input, Output
          CX - delay counter, counter

VARIABLES: A11AH ADR

INPUTS: Control signals from digitizer and ports

OUTPUTS: Signals to ports for flood, focus and DMA transfer
to take picture.

LOCATION: 2700H

SIZE: 062H

COMMENTS: Must be run from Master Loop

VERSION: 3 8 22.36
Preprocessor (PREPRO)

2.3 The purpose of the Preprocessor is to help remove the non-uniformity of the flood light illumination.

Each location in picture memory has its own encoded 8 bit multiplier. The multipliers are stored in upper memory (20000H) henceforth called multiplier memory which is above the picture memory (10000H). In general the program takes in each multiplier, decodes it, then multiplies the corresponding location in picture memory and adjusts that location towards the average intensity found from LMETER.

The encoding of multipliers enables multiplications by any number between zero to 1.99 decimal. Bits 0-6 of the multiplier represent the value of the multiplier. Bit 7 is the code bit. A 1 represents a subtraction while a 0 represents addition. For example if value 20H was a multiplier at address 20143H and 13H was a number stored in address 10143H, then after decoding and multiplication the top byte of (13H x 20H) = 0260H or 02H would be added to the value leaving an 15H at address 10143H. The addition occurs because of the encoding of a 0 in the most significant bit and the multiplier actually represents the fractional difference to the average. Refer to Section 3.6 for a further discussion of multiplier encoding.

Fig. 2.3-1 shows a graph of picture memory value vs. address. After running the Preprocessor the values are closer to a sample average as determined by the Light calibration program and LMETER. Also marked improvement can be observed in an intensity distribution of a uniform surface. Obviously the correct multipliers must already reside in multiplier memory before attempting to execute the Preprocessor routine.
Intentionally or unintentionally the preprocessor can bias the entire picture memory. Care must be taken to load in the proper multipliers.

FIG. 2.3-1. Picture Memory Vs. Address For PREPRO
Preprocessor (PREPRO)

SUBROUTINES: None

RESISTERS: DI - Picture pointer
DS - segment set to 1000
ES - segment set to 2000
CX - general purpose
AX - multipliers, general purpose
BL - add or subtract flag
DL - general purpose

VARIABLES: 08C4 LMSKP

INPUTS: Values from picture memory and multiplier memory.

OUTPUTS: Values to picture memory linearized with respect to light intensity.

LOCATION: 2000H

SIZE: 36H

NOTE: Preprocessor should be executed before Classifier and the proper multipliers should reside in multiplier memory.
This program could be transferred to the Classifier routine but is left in the present configuration to aid analysis.

VERSION: 8 10 13.52
SUBROUTINE PREPROCESSOR

FLOWCHART

START

INITIALIZE REGISTERS

SET POINTER TO TOP LEFT EDGE OF MEMORY

PRODUCT

SET MULTIPLIER VALUE

DI

ROTATE VALUE LEFT TO DECOD

SUBTRACT?

YES

SET FLAG TO 0

NO

MULTIPLY TP MULTIPLIER

SCALE

ADD MULTIPLIER TO PICTURE VALUE

YES

SET FOR ADD?

NO

SUBTRACT PICTURE VALUE

YES

SUBTRACT PICTURE VALUE FROM PICTURE VALUE

ES

DONE WITH ALL VALUES?

YES

SET LIMITER FOR SLIPS

NO

MOVE POINTER TO RIGHT

RETURN TO MASTER LOOP
Classifier (CLSIFR)

2.4 The Classifier routine separates the picture data stored in picture memory as fat (02), meat (00) or background (03). The useful picture information resides between rows 00 and EF and columns 00 to EC as shown in Fig. 2.4-1. All surrounding areas are ignored. In addition boundary values are stored on the perimeter of the useful stored image by the calling of DRWBOX Subroutine.

<table>
<thead>
<tr>
<th>USEFUL PICTURE INFORMATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000 0001 0002 0003 ... 00EC ... 00FF</td>
</tr>
<tr>
<td>0100 0101 0102 0103 ... 01EC ... 01FF</td>
</tr>
<tr>
<td>0200 0201 0202 0203 ... 02EC ... 02FF</td>
</tr>
<tr>
<td>0300 0301 0302 0303 ... 03EC ... 03FF</td>
</tr>
<tr>
<td>0400 0401 0402 0403 ... 04EC ... 04FF</td>
</tr>
<tr>
<td>0500 0501 0502 0503 ... 05EC ... 05FF</td>
</tr>
</tbody>
</table>
| ... ... ... ... ... ...
| ... ... ... ... ... ...
| ... ... ... ... ... ...
| ... ... ... ... ... ...
| EF00 EF01 EF02 EF03 ... EFEC ... EFFF |
| ... ... ... ... ... ...
| ... ... ... ... ... ...
| ... ... ... ... ... ...
| ... ... ... ... ... ...
| FF00 FF01 FF02 FF03 ... FFEC ... FFFF |

Fig. 2.4-1. Picture Memory

Fig. 2.4-2 is a distribution showing the light intensity on the x axis vs. area on the y axis for all data points in picture memory. Initially assume points 1, 2 are together and points 3, 4 are together. The classifier separates meat from background (1,2) then meat from fat (3,4). The values 03, 00, 02 are assigned to background, meat, and fat respectively.
If points 1, 2 and 3, 4 are separated there remains regions B and D that are unknowns. For these unknown regions the hysteresis algorithm can be used to determine their classifications based on the previously classified point. For example if the pointer had been classifying background and hit a slightly higher intensity pixel (region B) it will be classed as background. Only if it exceeds point 2 will it be classed as meat or fat. This effect is commonly known as hysteresis and may be implemented merely by separating in value points 1, 2 and 3, 4.
Classifier (CLSIFR)

SUBROUTINES: DRWBOX

REGISTERS USED: DI - used as pointer to upper memory picture array
CX - temp storage
   - store the # of columns in a row
   - last address storage
EX - # of rows in the matrix

VARIABLES: 620H FTHRUN
            8C8H UPRLV1
            8C9H UPRLV2
            8CAH LWRLV1
            08CBH LWRLV2
            8CCH BIGFLG
            8D2H SMLFLG

INPUTS: Hex intensity values from picture.

OUTPUTS: Values 00,02,03 back to picture memory.
Previous values purged.

LOCATION: 3000H

SIZE: 7CH

COMMENTS: Classifier must precede any beef grading analysis programs.

VERSION: 9 5 14.18
CLASSIFIER LAMNT
PLACEDNT

START

CLEAN PUD
THICKNESS RUN
FLAG

INITIALIZE
REGISTERS

LCOP1
D2 = CURRENT
LOCATION — NU
CONTAIN LEVEL — NU
DATAT — D3
YES

PUT LEVEL IN
CURRENT
LOCATION DENCK
LEV 1

NO

WHATIS
D2 = CURRENT
LOCATION DENCK
LEV 2

YES

PUT LEVEL IN
CURRENT
LOCATION DENCK
LEV 3

NO

LCOP2
POINT TO WEST
LOCATION IN
PICTURE ARRAY

D0 = PICTURE
NU = ARRAY BECK
NU = CLASSIFIED

YES

CONTINUE
RETURN

29
Boundary Search (BOUSCH)

2.5 This program finds the boundary between meat and fat. In Boundary search the background is also treated as fat so a boundary can be drawn between meat and background also.

The search starts with a pointer at the center of the loin eye (location 7F76H). Unusual conditions must be accounted for in starting at the center. If the pointer lands in the middle of a large piece of fat or background the pointer must be moved until it is in the meat area. The pointer is positioned according to the cross hairs from the focus lights. (It is assumed that the pointer will land inside the loin eye.)

Some of the conditions that the program will encounter are illustrated in Fig. 2.5-1. Pointer movement is from right to left in this case with the exclamation point indicating where the boundary will be drawn.

<table>
<thead>
<tr>
<th></th>
<th>B</th>
<th>F *</th>
<th>M</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>OK</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No meat error</td>
<td>B</td>
<td>F</td>
<td></td>
<td>F</td>
</tr>
<tr>
<td>OK</td>
<td></td>
<td>F</td>
<td>B *</td>
<td>M</td>
</tr>
<tr>
<td>No meat error</td>
<td>B</td>
<td>B</td>
<td>F</td>
<td>B</td>
</tr>
<tr>
<td>OK</td>
<td>B</td>
<td>B</td>
<td>F *</td>
<td>M</td>
</tr>
<tr>
<td>OK</td>
<td>B</td>
<td>B</td>
<td>B *</td>
<td>M</td>
</tr>
<tr>
<td>OK</td>
<td>B</td>
<td>B *</td>
<td>M</td>
<td>B</td>
</tr>
<tr>
<td>OK</td>
<td>B *</td>
<td>M</td>
<td>F</td>
<td>B</td>
</tr>
<tr>
<td>OK</td>
<td>F *</td>
<td>M</td>
<td>B</td>
<td>F</td>
</tr>
<tr>
<td>OK</td>
<td>B</td>
<td>F *</td>
<td>M</td>
<td>B</td>
</tr>
</tbody>
</table>

FIG. 2.5-1. Boundary Conditions
When the pointer finds a fat cell it checks to see if it is an isolated cell. If it is isolated Fig. 2.5-2 the pointer will start searching for another fat cell again. If another fat cell is found and non-isolated, a boundary search is attempted, Fig. 2.5-3.

```
M M M
M F M
M M M
```

**FIG. 2.5-2.** Isolated Fat Cell (no attempt to draw boundary).

```
F M M
F M M
M M M
```

**FIG. 2.5-3.** Non-isolated Fat Cell (boundary search attempted).

When a boundary search is attempted the pointer must follow a line that will lead it to the inner most edge between meat and fat. The first boundary point is saved.

The following describes the Freeman chain code algorithm [2]. The boxes in Fig. 2.5-4 represent the directions the pointer may take in searching for the next fat cell. The pointer moves in a counterclockwise direction e.g., 0-1-2-3 urtil it finds a fat cell.

```
3 2 1
4  C 0
5  6 7
```

**FIG. 2.5-4.** Direction Boxes
When a fat cell is found the direction boxes are overlayed onto the fat cell with the fat cell in the center of the boxes, Fig. 2.5-5. The center fat cell is changed to a boundary point represented by an "X" Fig. 2.5-6. The boundary address is stored in lower memory.

\[
\begin{array}{cccc}
F & F & M & M \\
3 & 2 & 1 & M \\
4 & C & 0 & M \\
5 & 6 & 7 & M \\
\end{array}
\]

FIG. 2.5-5. Box Overlayed Onto Memory Cells.

\[
\begin{array}{cccc}
F & F & M & M \\
3 & 2 & 1 & M \\
4 & C & X & M \\
5 & 6 & 7 & M \\
\end{array}
\]

FIG. 2.5-6. Center Fat Cell to Boundary Point.

At this point a new direction is sought. The new direction is chosen by the relationship.

\[
N^+ = \begin{cases} 
\text{Mod(N-1)} & \text{N even} \\
\text{Mod(N-2)} & \text{N odd}
\end{cases}
\]

where \(N^+\) is the new direction

\(N\) is the old direction

Mod 8 arithmetic is used.

Example 1. If the fat point was found in the direction 4,

even) the next direction starts at direction 3.

Example 2. Old direction = 5

New start direction = 3
If a fat cell is not found in the new direction the pointer moves in a counterclockwise direction until it finds a fat cell. The direction boxes are centered about the new cell, Fig. 2.5-7, and the process is repeated until the pointer reaches the original starting point.

The boundary is now checked to see if it is large enough to be called a valid area. If the bounded area found is too small (Fig. 2.5-8 and Fig. 2.5-9) then all previously found boundary points are changed back to fat. The locations of the boundary points to be changed have been stored in lower memory.

The pointer is moved vertically, about an inch, and the search for a large enough boundary continues. Fig. 2.5-10 and Fig. 2.5-11 shows the correct boundary. The initial pointer direction is determined by the LCRS flag which is set by the DRECTN Subroutine. If the LCRS flag is a 1 the pointer moves to the left in subroutine MOVPOS. If the LCRS is zero, the pointer moves to the right or is incremented in MOVPOS subroutine.

It is interesting that after classification, many boundaries exist. The Boundary search program along with Spline must choose the proper valid boundary.
FIG. 2.5-8. Too Small Boundary.

FIG. 2.5-9. Too Small Boundary (Block Display).
FIG. 2.5-10. Correct Boundary.

FIG. 2.5-11. Correct Boundary (Block Display).
Boundary Search (BOUSCH)

SUBROUTINE: THER, OVRWRT, DRECTN, MOVPOS

REGISTER:

DI - picture pointer
BP - pointer to lower memory storage of Boundary Location
BH - fat or Bkgd indicator, # of Boundary points
DX - starting point
CX - counter

VARIABLES: 400-600H MBLK 8CCH BIGFLG
061AH MTFT 8CDH OLDROU
06D2H FLAGOV 8DOH LCRS

INPUTS: Bytes 00,02,03 nominal or 00,01,02,03 from picture memory.

OUTPUTS: Boundary (01) along Meat/Background or Meat/Fat edge.

LOCATION: 3100H

SIZE: 1DBH

COMMENT: Must be preceded by Classifier.

VERSION: 9 3 13.52
Spline (SPLINE)

2.6 The Spline routine eliminates the boundary, if any, around the Spinalis dorsi, longissimus costarum and serratus dorsalis muscles. See Fig. 1.1-1. These muscles should not be included within the longissimus dorsi (Loin eye) boundary. This is an acute problem because the separation may not be large enough to determine the muscles from the running of Boundary search routine.

Spline also reduces the "breakout" effect that may occur in Boundary search. Breakout occurs when the boundary follows the edge of the carcass towards the edge of picture memory.

The routine first steps to the boundary and overlays a line of specified length on top of the existing boundary. This line is called a spline. The first spline length drawn is extremely long compared to the circumference of the longissimus costarum (LC) muscle. Its purpose is to help eliminate the breakout problems associated with incomplete fat around the loin eye proper.

The entire spline is moved one position along the boundary.

The distance between the beginning (H) and ending (T) points is computed. If the distance is within an allowable minimum distance the H and T addresses are stored in memory for later processing.

After the points are stored the spline is moved one position along the boundary and a check on the H-T distance is made again.

The process continues until all allowable minimum distance points are loaded into memory. Subroutine PRECON is called which calls CONNT Subroutine to connect point pairs whose addresses are stored in memory. The points are connected by storing an (01) boundary along the path between 2 points. Fig. 2.6-1, illustrates the original boundary. Fig. 2.6-2 shows
FIG. 2.6-1. Spline Before CONNT.

FIG. 2.6-2. Spline After CONNT, REST and BOUSCH.
the altered boundary after CONNT, REST and BOUSCH subroutines. REST is a subroutine that converts all boundary points to fat cells. BOUSCH is the Freeman chain code algorithm described in Section 2.5.

After the CONNT subroutine has connected all the points previously stored in memory, the entire process is repeated using shorter spline lengths. Ideally the spline length should be reduced by one for each complete path around the boundary. The new spline length is chosen as new length = old length - n. As n gets larger the number of iterations about that loin eye is reduced thus improving execution speed. Also as n increases the smoothness of the boundary decreases.

In Fig. 2.6-3 the original boundary is shown. Note the inclusion of the L.C. muscle and musculature in the lower right section. The first run through Spline removes some of the excess boundary from the bottom, Fig. 2.6-4. The next run further reduces boundary in this area, Fig. 2.6-5. Next the top L.C. muscle is removed, Fig. 2.6-6. More boundary is removed to the final configuration, Fig. 2.6-7. The final Figure 2.6-8 is included to show that after the spline program is called again no further reduction of the boundary takes place.
FIG. 2.6-3. Before Spline.

FIG. 2.6-4. Spline Iteration.
FIG. 2.6-5. Spline Iteration 2.

FIG. 2.6-6. Spline Iteration 3.
FIG. 2.6-7. Spline Iteration 4.

FIG. 2.6-8. Spline Iteration 5, No Change.
Spline (SPLINE)

SUBROUTINE: REST, BOUSCH, SPLN, SMOVE, PRECON

NEXTPT, SQUARE, CONNT, POCK

REGISTERS: CX - counter, spline length

DI - picture pointer

BX - direction pointer, general

DX - difference, slope flag, general

BP - memory pointer to 7200

VARIABLES: 060EH SPLHM 0709H SPFLG

060CH SPLTM 070AH STTPT

060CH SPLTB 070CH SLENGTH

060EH SPLHB

0700H SPLH

0702H SPLHD

0704H SPLT

0706H SPLTD

INPUTS: 00, 01, 02, 03 from picture array

OUTPUTS: 00, 01, 02, 03 to picture array (altered shape)

LOCATION: 7A00H

SIZE: 1EDH

COMMENTS: Boundary search must precede Spline

VERSION: 8 2 11.13
Area Acquisition (AREAAQ)

2.7 Area acquisition totals the meat and fat within the boundary. The picture pointer DI is initialized to the bottom left point of useful picture information in upper memory. The pointer is moved across the row searching for a boundary point. If the end of the row is reached before a boundary point is found, the pointer is moved up one row, placed to the left edge and the process continues until a boundary point is found. When the first boundary point is found its picture memory address is saved as the bottom point. The LRFIND subroutine is then called to record whether the point found is the left or right most boundary point. Assuming the pointer is not at the end of the row a check is made on BIGFLG. If BIGFLG is zero, and a meat cell is found to the right of the boundary point the fat or meat counting begins. The fat and meat are counted as the pointer moves left to right across the row. If BIGFLG is set, counting begins regardless of the type of cell following the boundary. When the pointer reaches the next boundary the counting ceases, the pointer is moved up one row and the process is repeated.

An error condition exists if the pointer finds the end of the row before finding a boundary point. This error usually occurs when only one boundary point is found in the row, i.e., top or bottom point. In this case the previously counted cells must be subtracted from their respective counters.

The picture pointer moves up a row and continues until a boundary is no longer found in the row. The top most boundary point address is saved and the fat cell, meat cell and background cell count is saved in memory.
The output of Area acquisition is the bottom, top, left and right most boundary points of the enclosed boundary along with the total meat cells, total fat cells and total background cells.

Note that the background cells are recorded separately in memory, but since they were located within the bounded area they should be considered very dark meat.
Area Acquisition (AREAAQ)

SUBROUTINE: LRFIND

REGISTERS: BP - variable pointer
           DI - picture pointer
           AX - meat counter/adder compare
           BX - fat counter
           CX - loop counter
           DH - first boundary point flag
           DL - row boundary point flag
           SI - current boundary point

VARIABLES: 0600H LCOL 0606H BCOL 0714H BKGDNT
           0601H LROW 0607H BROW 08CCH BICFGL
           0602H RCOL 0608H FCLOW 08D2H SMLFLG
           0603H RROW 0609H FCHIGH 08D4H FATSQ
           0604H TCOL 060AH MCLOW 08D6H MEATSQ
           0605H TROW 060BH MCHIGH

INPUTS: 00,01,02,03 from picture array.

OUTPUTS: Meat and fat cell count as well as top, bottom, left and right
         edge points. Alternate storage locations depending on flag
         settings.

LOCATION: 3400H

SIZE: 11BH

COMMENTS: Must be preceded by Boundary search.

VERSION: 9 5 12.34
Max Point Search (MXMAIN)

2.8 Max point search program finds the longest line or axis that can be drawn within the loin eye. The top and bottom most boundary point addresses are loaded in from their storage locations as found from AREAQQ. The picture pointer is set to the address of the top most boundary point. This point is called movable. The other point is called stationary. The movable point is moved to either side of the original point searching for the maximum distance to the stationary point. When a certain number of points each side of the original point are checked then the program makes the movable point stationary and changes the stationary point to movable, then repeats the process. The two points that have the maximum distance are stored in memory to be used later in the fat thickness measurement routine.

Within the Max point search package is included the subroutines Nextpoint, Find max and Distance.

The Nextpoint subroutine finds an adjacent point along the boundary. The Freeman chain code algorithm is used [2]. The description is the same as Boundary search with the exception that the Nextpoint routine can change direction. A clockwise or counterclockwise table is used for this function. The table is chosen by the top byte of a two byte register BX called BH. The specific direction within the table is referenced by the register BL, the lower byte of the BX register. Every time BL is loaded within the subroutine, a new direction is chosen. When a boundary point is found, the program returns to the calling program with the picture pointer (DI) at the next boundary point.
The Findmax subroutine is essentially the counter that records the number of points each side of the original point. The changing of BH via correction table is to reverse directions of the next point.

The distance subroutine is used by Max point to find the distance between two points. The maximum or minimum distance can be found depending on flag settings. Note that the distance subroutine has various checks to recognize the calling program, i.e. SPFLAG, SI register.

The stationary row and column is loaded into the DX general purpose register. The movable point is already pointed to by the DI register.

The equation to determine distance between the two points is

\[ d^2 = (mrow - srow)^2 + (mcol - scol)^2 \]

The program computes for the distance squared not the distance. It is not necessary to take the square root at this time.

Since the aspect ratio (width to height) of the individual pixels is not 1:1, a correction must be made to linearize the squared distance.
Max Point Search (MXMAIN)

SUBROUTINES: FINDM, NEXTPT, DISTAN

REGISTERS: CX - counter, general purpose, 2 MSD, LSD

SI - offset data pointer, move data pointer,
temporary storage of movable, stationary.

DI - move data pointer

BP - data pointer

DI - picture pointer

DX - mask and test for picture pointer, temp DI storage

BL - direction indicator for table

BH - which table pointer

DL - most significant digit for distance

VARIABLES: 704H SPLT

708H SPFLG

3C00H-31FH CCW table

3D00H-3DIFH CW table

3D20H-302F TABLE transfer correction

INPUTS: Values from area acquisition, TROW, BROW, TCOL, BCOL.

Values 00,01,02,03 from picture memory.

OUTPUTS: Maximum axis points on the boundary.

LOCATION: 3800H

SIZE: 52EH
COMMENTS: Area acquisition must precede Max point.

VERSION: 9 5 12.34
Fat Thickness (FTHICM)

2.9 The Fat thickness program finds the fat thickness at a point between a boundary and the external background. The boundary point that is chosen is based on the placement of the cross hairs during picture loading. If the loin is right sloped then the fat thickness will be found off the right cross hair. If loin is a left sloped, the fat thickness will be measured from the left edge of the cross hair. The placement of the cross hair is important in locating the proper fat thickness measurement location.

Before Fat thickness is called, DRECTN should already have been run by a call from Boundary search. The DRECTN Subroutine determines whether the stored picture image is right or left slope oriented. A short description follows here because of the subroutine's location in the assembly listing.

The MEACNTP and MEACNTN are cleared initially and the picture pointer is set to 7679H (center of the screen where the cross hairs cross).

FIG. 2.9-1. Right Sloped Loin Eye. FIG. 2.9-2. Left Sloped Loin Eye.
Two slopes at 45° to the cross are followed down until a boundary is reached. The meat is counted in each line and totalled in MEACNTP and MEACNTN.

If the MEACNTN was less than MEACNTP then the LCRS flag is cleared, Fig. 2.9-1. Otherwise the LCRS flag is set, Fig. 2.9-2.

The pointer is shifted left or right depending upon the slope. The routine then returns to the calling program.

The proper center location is found after CORRECTQ has been called. A check is made to see if the LCRS flag is set or cleared. If set, the picture pointer (DI) moves to the left searching for a boundary point. When a boundary is found the program checks to see if it is the first boundary point found or not. If the pointer moves a fixed distance left or right without finding a boundary the program terminates.

After the boundary is found a search begins for the background point. When both a boundary point and background point are found each address is stored for processing. A minimum distance must be found between the background point and a boundary point to eliminate angle as an influencing factor, Fig. 2.9-3.

![Diagram](image)

FIG. 2.9-3. Minimum Fat Thickness
FIG. 2.9-4. Fat Thickness Points.
FIG. 2.9-5. Fat Thickness Points (Block Display).
Before the fat thickness measurement is summed the minimum fat thickness must be found by moving the boundary point up and down the edge and calculating the distance at each location. When the minimum is found, it is summed in a program called TABUL for tabulation. The above procedure is repeated eleven times, Fig. 2.9-4. Each time the procedure is rerun the initial start point is moved downward by CORRECTQ. Each fat thickness measurement is summed in SUMDST, Fig. 2.9-5.

When the iteration is complete the average over 11 distances are found and loaded into NUMM.

The program returns to Master loop. If an error occurs, e.g. not finding a boundary point, the average over the existing sum is taken and loaded into NUMM. The approximate fat thickness location for a left slope loin is shown in Fig. 2.9-6.

FIG. 2.9-6. Approximate Fat Thickness Location.
Fat Thickness (FTHICM)

SUBROUTINES: CORRECTQ, DISTAN, NEXTPT, TABUL, CHECK

REGISTERS: 
BH - table pointer
BL - element pointer
DI - picture pointer, boundary address
SI - address pointer
AL - boundary found flag
DX - background address, row-column check
AX - summations
CS - counter

VARIABLES: 
620H - FTHRUN
708H - SPFLG
618H - NUMM
710H - MEACNTP
7DEH - MEACNTN
8C6H - STARTD
8DOH - LCRS
D05H - SUMDST
D07H - SAVG
D09H - SAVB

INPUTS: Values from picture array. 00,01,02,03

OUTPUTS: Squared average fat thickness value to NUMM. The output is an average squared of eleven measurements.

LOCATION: 4000H

SIZE: 1F9

COMMENTS: Fat thickness must be preceded by Boundary search.

VERSION: 9 2 10.32
Squeeze-Display (KRMDSP)

2.10 The Squeeze display program allows the operator to see a compressed view of the picture memory.

A determination can quickly be made as to whether a correct boundary has been found in addition to the fat thickness measurement location.

Note that the view as seen on a data terminal is distorted, mainly because the data terminal is usually an 80 column by 24 row display.

The cells in a 3 x 10 block of picture array are summed. The sum is compared to a constant level value and a character is loaded into the printer buffer according to this level. For example, if the 3 x 10 area is mostly fat cells (02) then the character for fat is loaded into the printer buffer. If the 3 x 10 group of cells was mostly meat then the meat character is loaded into the print buffer. If any one cell in the 3 x 10 group is an edge character (04) or boundary character (01) the ASCII character representing an edge or boundary is loaded into the print buffer directly without regard to the sum. An edge character has priority over a boundary character for printing.

After the single character for the 3 x 10 block is printed the character for the next 3 x 10 block to the right is printed and so on until all characters for all blocks in useful picture memory are printed. At this point the operator should see a complete picture on the data terminal.

Note that Squeeze-display may be run after Fat pieces or Fat thickness to get some indication of the fat piece distribution and/or fat thickness measurement location. The display is a summation-comparison process and
gives no useful data other than a rough estimate of the accuracy or inaccuracy of the boundary and fat thickness measurement locations. Examples of the output from Squeeze-display may be seen in Figs. 2.9-4 and 2.6-5.
Squeeze-Display (KRMDSP)

SUBROUTINES: CHRO

REGISTERS: DI - picture pointer
          AX - character register
          SI - sum register
          CX - character rows
          BX - character columns
          DX - mask and compare rows and columns

VARIABLES: 620H FTHRUN

INPUTS Values from picture memory 00,01,02,03,04

OUTPUTS: ASCII characters to data terminal representing a
          3 x 10 block in picture memory.

LOCATION: 3300H

SIZE: 8FH

COMMENTS: Classifier must precede Squeeze-display.

VERSION: 9 3 16.47
Box Distribution (BOXDIS)

2.11 The Box distribution module breaks the boundary enclosed area into sixteen separate boxes and stores the cell type distribution for each box in memory.

The Area acquisition module must be run first in order to obtain the left, right, top and bottom edges of the bounded area. These four edges form a large box around the area.

In the initialization section of the module the edge points from Area acquisition are used to find the box sizes. After determination, the box sizes are stored in DCOL and DROW.

Box distribution works very much like Area acquisition in that the pointer (DI) begins at the lower most edge of the large box surrounding the bounded area. The pointer moves from left to right while looking for a boundary point. If found the program attempts to determine whether the pointer is inside or outside the area. It does this by checking to the right of the boundary point if BIGFLG is a zero. If the position is fat or background then the pointer must be outside the loin eye area. If the cell pointed to is meat then the counting or distribution data gathering begins. If BIGFLG is set to a one then no check is made to the right of the boundary point.

Subroutine Whichbox finds what box the pointer DI is in. It then gets the memory pointer (BP) to point to the proper meat or fat storage location for the box. In Fig. 2.11-1, numbers 672 and 673 are the meat count locations, 1 is the box number and 674 and 675 are the fat count locations. The figure shows the meat and fat counter locations for all 16 boxes.
Each time the pointer is incremented, a check is made to see if the pointer is at the right edge of the large box surrounding the area. If the end of the row or edge of the large box is found before finding the right boundary point an error condition exists. Upon error the Fixup routine subtracts off the cell counts that were accumulated in error. The usual running of the Fixup routine occurs when there is a single boundary point in the bottom or top row. The cells to the right of this singular point are not in the bounded region and thus are subtracted from the box distribution tally locations.

With no error condition the end of the row is found. The pointer is moved up 1 row (decremented in memory) and the process of finding the distribution is repeated.

When a boundary point is not found in the row being scanned the module is done and returns to Master loop.
<table>
<thead>
<tr>
<th>672</th>
<th>Meat Count</th>
<th>676</th>
<th>67A</th>
<th>67E</th>
</tr>
</thead>
<tbody>
<tr>
<td>673</td>
<td>1 - Box #</td>
<td>674</td>
<td>Fat Count</td>
<td>677</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
<td>678</td>
<td>67C</td>
</tr>
<tr>
<td>680</td>
<td>681</td>
<td>682</td>
<td>686</td>
<td>68A</td>
</tr>
<tr>
<td>683</td>
<td>687</td>
<td>684</td>
<td>688</td>
<td>68C</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>685</td>
<td>689</td>
<td>68D</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>692</td>
<td>696</td>
<td>69A</td>
</tr>
<tr>
<td>693</td>
<td>697</td>
<td>694</td>
<td>698</td>
<td>69C</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>695</td>
<td>699</td>
<td>69D</td>
</tr>
<tr>
<td>11</td>
<td></td>
<td>6A2</td>
<td>6A6</td>
<td>6AA</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td>6A3</td>
<td>6A7</td>
<td>6AB</td>
</tr>
<tr>
<td>13</td>
<td>14</td>
<td>6A4</td>
<td>6A8</td>
<td>6AC</td>
</tr>
<tr>
<td>15</td>
<td></td>
<td>6A5</td>
<td>6A9</td>
<td>6AD</td>
</tr>
</tbody>
</table>

FIG. 2.11-1. Boxes For Box Distribution
Box Distribution (BOXDIS)

SUBROUTINES:  WHTBOX

REGISTERS:
CX  -  counter, general purpose
BP  -  memory pointer to box-meat, fat counts
DI  -  picture pointer
DX  -  masking for DI

VARIABLES:
600H  LCOL
602H  RCOL
605H  TROW
607H  BROW
670H  DCOL
671H  DROW
672-6B1  meat and fat counts for individual boxes

INPUTS:
Top, bottom, left and right edge points of the loin eye area (computed from Area acquisition)

OUTPUTS:
Stored meat and fat counts for each of the sixteen boxes and box size.

LOCATION:  4600H

SIZE:  148H

COMMENTS:
Area acquisition must precede Box distribution

VERSION:  8 14 14.59
Fat Pieces (FATPCS) and Fat Boundary Search (FATBDY)

2.12 FATPCS finds all the fat pieces within the loin eye and stores the number in memory.

The picture pointer (DI) begins outside the loin eye area (OLEA). The bottom row, top row, left column, right column information found from area acquisition is loaded into registers. The top row left column address is loaded into the DI register. The pointer DI is moved to the right until it hits a boundary point or the right column. If a boundary point is found the pointer is incremented to the right and checked to see if it has reached the right column. If the cell is exclusively meat then 1) a flag is set indicating inside the loin eye, 2) the picture pointer (DI) is incremented. The pointer continues row by row until finding a fat cell. Subroutine Fat boundary is called and a boundary is drawn around the fat piece and the number of fat pieces is increased by 1. Compare Fig. 2.12-1 and 2.12-2. Note that the boundary is drawn by storing a (04) in the picture memory location. This is to distinguish a fat boundary from a loin eye boundary (01). After the fat boundary is drawn the pointer is back to the starting point on the fat piece. The pointer is moved one position to the right. At this point it is likely that the pointer has landed inside the fat piece. The pointer is incremented until it finds a meat cell, then the search for all fat pieces continues. If a single fat cell is encountered, the cell is changed to a fat boundary and the fat counter incremented.

Fig. 2.12-1 shows a magnified view of picture memory. The fat cells are indicated by colons, meat by a period, and loin eye boundary by an asterisk. Fig. 2.12-2 shows the same area of picture memory after the fat pieces routine has executed. The fat boundaries are indicated by a
zero. In all, eleven fat pieces are shown. Figure 2.12-3 gives a rough estimate of where the fat pieces reside in the loin eye. Fig. 2.12-3 is the output of Squeeze-display. A zero may represent one or for worst case twenty-two fat boundary cells.

When the picture pointer is at the bottom row right column the procedure is done.
FIG. 2.12-1. Fat Pieces, No Fat Boundary Drawn.
FIG. 2.12-2. Fat Pieces, Fat Boundaries Drawn.
FIG. 2.12-3. Fat Pieces (Squeeze-display).
Fat Pieces (FATPCS) and Fat Boundary Search (FATBDY)

SUBROUTINES:    INCR, FATB

REGISTERS:     AX - fat pieces counter
               BX - last row, column
               CX - end address for current row
               DH - new row indicator
               DL - inside/outside loin eye flag
               BP - error pointer table
               DF - pointer to picture memory
               DS - pointer to data
               SI - first fat boundary in FATB search

VARIABLES:    600H  LCOL
               602H  RCOL
               605H  TROW
               607H  BROW
               633H  NOPCS

INPUTS:        Values from picture array 00,01,02,03

OUTPUTS:       Altered picture array. Fat boundaries around fat
               pieces. Number of fat pieces to memory NOPCS.

LOCATION:      4200H

SIZE:          3A3H

COMMENTS:     Area acquisition must precede Fat pieces.

VERSION:       9 13 16.59 98
Calculate Results (CLCRES)

2.13 CLCRES performs all the calculations necessary for the beef grading programs. Although the program has the capability to enter values from the keyboard e.g. weight, %KPH, the present version does not query the keyboard for information.

Two programs are called to give the proper results, CALCUL and PCNTOT. The CALCUL subroutine first adds the fat count and meat count together to form a total fat meat area. In each case a subroutine, AREAML, is called to convert to hundredths of an inch from the pixel element count.

The next section in CALCUL converts the meat, fat areas into %fat and %meat of the total area. The subroutine PERCNT does the actual calculation after the proper registers are loaded. The %fat in the square is also found.

The fat thickness calculations are performed by taking the squared average fat thickness and square rooting by an interval halving technique until the proper tolerance is found. The fat thickness is converted to inches.

Next, from the results of BOXDIS, CLCRES uses the subroutine PCNTDT to calculate the percent of the total loin eye cells in each box to the total cells in each box. PCNTDT then calculates the percent of fat cells in the box to the number of loin eye cells in the box. All percents are found for each of 16 boxes that contain the loin eye. Each calculation performed is checked for a zero divide and the fractional part is rounded not truncated.
The map for DSPRES is also created in CLCRES. The symbols are as follows:

[ ] - non significant box less than 1/3 of its contents is loin eye.

[--] - significant box, percent of loin eye fat in box is less than overall percent of loin eye fat.

[==] - significant box, percent of loin eye fat in box is the same as the overall percent of loin eye fat.

[++] - significant box, percent of loin eye fat in box is greater than the overall percent of loin eye fat.

After all the percentages are calculated, the numbers are converted to their equivalent decimal numbers and stored in the PCKDEC string for later printout or display.

NOTE - BOXDIS must be executed prior to executing CLCRES for this area to have any meaning.
Calculate Results (CLCRES)

SUBROUTINES: STX, INFINT, ASCII, STR1, DECHEX, CALCUL, PCNTDT,
MESSAGE, DECLOD, INSMART, DECMIL, CHAROT

REGISTERS: SI - header string pointer
CX - counter
AH - general purpose
DI - string transfer, map pointer
DX - general purpose
BX - general purpose

VARIABLES: 608H - FAT
60AH - MEAT
618H - NUM1
622H - EQUEI
626H - EQKPH
628H - FATMEA
62AH - FTTENS
62EA - EQFT
633H - NOPCS
637H - EQNA
639H - OLDVAT
900H - FREAD
A00H - PCKDEC, WGT
A02H - KPH
F13H-F16H-ASCBUF
F17H - TEMP
F19H - DELU
F1BH - DELL
F1DH - BXAREA
F1FH - SIGL
F21H - AREA
F23H - AFAT
F25H - AMEAT
F27H - PFAT
F29H - PMEAT
F2BH - PFATBX
F2DH - NPCS
F2FH - FATH
F31H - SGBCT
F33H - EQUCNT
F35H - GRTCNT
F37H - LESCNT
F39H - MAPCNT
F3AH - WGT
F3CH - KPHH
F3EH - M
2DEFH - 2DFEH HEX
2DFFH - 2EO8H DECTBL
2E09H - 2E11H NUM

INPUTS: Values from Box distribution, Area acquisition and Fat thickness that were stored in memory.

OUTPUTS: Computed beef grading results stored in memory.
LOCATION: 2B00H

SIZE: 311H

COMMENTS: Must be preceded by inputting routines; Box
distribution, Area acquisition, Fat thickness.

VERSION: 9 7 16.22
Display Results (DSPRES)

2.14 Display results takes the decimal values found in the PCKDEC string, converts the numbers to their ASCII equivalents, inserts the ASCII into a string called DPYBUF, then outputs the DPYBUF to the data terminal screen.

DPYBUF is a display string which contains the English identification (in ASCII equivalent code) for all the numbers in PCKDEC as well as the ASCII equivalents of PCKDEC. DPYBUF is created in the loading of the program and is updated with the new values of PCKDEC each time Display results is executed. The DPYBUF as created is seen in Fig. 2.14-1 and Fig. 2.14-2 includes a sample set of data. Notice that the X location in the first figure are updated to numbers in the second figure.

NOTE - This program assumes that the string PCKDEC has been properly created and contains the valid numbers for the selected carcass (i.e., all the measurement programs and calculation programs were run prior to DSPRES).
### FIG. 2.14-1. Empty DPYBUF Output.

<table>
<thead>
<tr>
<th>PERCENT</th>
<th>AREA</th>
<th>TOTAL FAT</th>
<th>TOTAL MEAT</th>
<th>TOT FAT</th>
<th>TOT MEAT</th>
</tr>
</thead>
<tbody>
<tr>
<td>XX.X</td>
<td>XX.X</td>
<td>XX.X</td>
<td>XX.X</td>
<td>XX.X</td>
<td>XX.X</td>
</tr>
</tbody>
</table>

### FIG. 2.14-2. Sample Data in DPYBUF Output.

<table>
<thead>
<tr>
<th>PERCENT</th>
<th>AREA</th>
<th>TOTAL FAT</th>
<th>TOTAL MEAT</th>
<th>TOT FAT</th>
<th>TOT MEAT</th>
</tr>
</thead>
<tbody>
<tr>
<td>07.5</td>
<td>06.1</td>
<td>00.0</td>
<td>00.0</td>
<td>02.0</td>
<td>MAP</td>
</tr>
<tr>
<td>00.7</td>
<td>12.0</td>
<td>00.0</td>
<td>12.9</td>
<td>78.5</td>
<td>51.0</td>
</tr>
</tbody>
</table>

---

114
Display Results (DSPRES)

SUBROUTINES: ASCII, STOR, MESSAGE, CHAROT

REGISTERS:
- BP - increment table position
- SI - message pointer
- DI - display pointer
- BX - length table pointer
- CS - counter
- AX - mask, data input, output
- DX - general

VARIABLES:
- A00H - A55H PCKDEC
- A56H - C34B DPYBUF
- C3CH - CCOH INCTBL
- CClH - CCCH LENTBL
- CCDH - STP

INPUTS:
- Tables and values stored in memory.

OUTPUTS:
- Proper ASCII characters to data terminal.

LOCATION:
- 2600H

SIZE:
- 8CH

COMMENTS:
- Must be preceded by Calculate results program.

VERSION:
- 9 5 19.44
Save Results (SAVRES)

2.15 Save results stores the date, location, area, and all the data presented in the displayed results (i.e. total fat, total meat) on a disk previously initialized as a data disk.

Save results reads track one sector one of the disk using a universal disk read/write routine DSKIO. The data is stored in a buffer called BUFF location at 7100H. This data is a directory of the date, location, carcass count, next track and sector addresses, and the number of data fields stored on the disk. Save results compares the present date and location to the one of the directory to see whether to continue with the carcass count (same date and location) or start counting at one (either different date, different location, or both). Using DSKIO, the present data in PCKDEC is stored on the disk at the track and sector specified by the directory. The sector address is incremented and compared to the maximum sectors on a track. If it exceeds the maximum then the track is set to one and a message is output to the terminal which reads; "THIS DISK IS FULL PLEASE REPLACE IT WITH A NEW FORMATTED DISK AND PRESS RETURN WHEN READY". The directory is then updated and stored on the disk using DSKIO whether a new disk was installed or not.

Save results uses the Disk support package and the routine DSKIO for support.

NOTE: This routine assumes that an initialized data disk is in drive 0 and all the data to be stored has been already calculated and the present date and location are stored in memory.
SUBROUTINES: DSKIO, MESSAGE, CHARIN, OUTHEX, DMASET, VFYDRVVRDY,
PARMREG, RESULT, ERROR

REGISTERS: DX - general purpose, track, sector number
DI - string pointers,
SI - string pointers, message pointer
AL - track number, sector number

VARIABLES: 2200 FLG
2202-225F MESS1 Full disk message
2260-2267 MESS2 Disk error message
0F00 COMMAND
0F01 TRK
0F02 SEC
0F03 NUM
0F04 TRK1
0F05 LEN
0F06 FIRST
0F08 BSEADDR
0FOA ADDR
0FOC DRV
0F0D CNT1
0F0F CNT2
0F11 CNT3
0F12 CRTL
0F13-0F16 ASCBUF
09F8 DAT
09FA     DAT2
09FC     LOC1
09FE     SAVCNT
0A00     PCKDEC
7100     BUFF

INPUTS:  Values from computed beef grading results. Input
         also from Date & location.

OUTPUTS: Stored beef grading results on data disk.

LOCATION: 2100H

SIZE: 167H

COMMENTS: Should be run after Calculate results.

VERSION: 3 19 14.11
2.16 The Draw box routine essentially forms a rectangular boundary of (01) cells in memory. The location and size of the box depends upon whether a right or left slope area is detected.

A call is made to DRECTN subroutine to determine right or left slope. The LCRS flag is set if a left slope was found. Before the box is drawn a call is made to RESTORE subroutine. The subroutine removes any previously drawn boundaries (01) by changing them to (02) cells.

The appropriate starting top left edge of the box is loaded into registers.

The top edge of the box is drawn, then down to the bottom right edge, followed by the left and bottom edge. When complete, flags are set and the program returns to Master loop.
Draw Box (DRWBOX)

SUBROUTINES: DRECTN, RESTORE, DRWBOX

REGISTERS: DI - picture pointer
            BX - start address for right box
            DX - height, width
            AL - boundary
            CX - number of columns, counter

VARIABLES: 08CCH - BIGFLG
            08DOH - LCRS
            08D2 - SMLFLG

INPUTS: Values from picture memory.

OUTPUTS: Boundary points in a square pattern in picture memory.

LOCATION: 2550 Start
           2580 DRWBOX subroutine

SIZE: 80H

COMMENTS: The start location refers to the normal entry point to
draw a box within the loin eye proper. Any box drawn
without regard to the loin eye slope should be implemented
with a direct call to DRWBOX subroutine.

VERSION: 9 12 1.58.21
SUBROUTINE DRAGX

FLOWCHART

START

LOAD IN NUMBER
OF COLUMNS

LOAD IN NUMBER
OF ROWS

DRAW THE TOP
EDGE AND STORE
BOUNDARIES

DRAW THE RIGHT
EDGE AND STORE
BOUNDARIES

SET THE POINTER
BACK TO TOP
LEFT EDGE

DRAW LEFT EDGE
AND STORE
BOUNDARY POINTS

SET BIG FLAG

SET SMALL FLAG

RETURN
Block Picture Display (BLKDPY)

3.1 BLKDPY displays a block of upper memory 80x23 bytes on the terminal screen as well as the address of the upper left hand corner (the start of the block). BLKDPY inputs a character from the keyboard and compares it to its set of commands. If the character is not one of the commands the program ignores the character. The valid commands are:

R (Relocate) - changes the starting address of the block (frame) to be displayed to that specified. Then displays that block. EXAMPLE: R<new address>[CR].

H (left arrow) - changes the starting address of the block (frame) to be 80 columns to the left of the present position (one to the left). Then displays that block.

K (up arrow) - changes the starting address of the block (frame) to be 23 rows up from the present position (one frame up). Then displays that block.

L (right arrow) - changes the starting address of the block (frame) to be 80 columns to the right of the present position (one frame to the right). Then displays that block.

J (down arrow) - changes the starting address of the block (frame) to be 23 rows down from the present position (one frame down). Then displays that block.

E (escape) - exit the program and return to the calling program.

After a command is typed the program offsets to the proper value then calls subroutine Display frame. Examples of output from block display can be seen in Figs. 2.9-5 and 2.6-1, 2. Although any location within memory can be the
<table>
<thead>
<tr>
<th>Column</th>
<th>0050</th>
<th>00A0</th>
<th>00F0</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Row</th>
<th>17</th>
<th>2E</th>
<th>45</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>5C</td>
<td>73</td>
<td>8A</td>
</tr>
<tr>
<td></td>
<td>A1</td>
<td>B8</td>
<td>CF</td>
</tr>
<tr>
<td></td>
<td>E6</td>
<td>FD</td>
<td>*</td>
</tr>
</tbody>
</table>

* = wraparound

FIG. 3.1-1. Block Display Addresses
top left location in the displayed block, Fig. 3.1-1 shows the block locations if only the arrow commands are used. Remember the data terminal output is distorted because a rectangular block in memory is shown as a square-like display.
Block Picture Display (BLKDPY)

SUBROUTINES: MESSAGE, CHARIN, INPINT, DISP, OUTWRD, CHAROT, OUTHEX

registers: SI - message pointer, picture location
            AL - input, output data
            DX - picture location
            BX - row counter
            CX - column counter

Variables: 110D LW
            110E-1111H TABLE
            1112-1121H HEX
            1122-1177H MESS1
            1178-117FH MESS2
            1180-1187H MESS3

Inputs: Values from picture memory

Outputs: ASCII characters to data terminal representing a
         block of picture memory.

Location: 1000H

Size: 180H

Comments: Block display is run after the data is classified.

Version: 0
Distribution (DISTRB) and Print Distribution (PTDIST)

3.2 The purpose of Distribution and Print distribution is to display for the operator a plot of light intensity (abscissa) vs. area (ordinate).

If the operator wishes to view the plot on the data terminal the Baud rate must be slow enough to view the information. The Baud rate (rate at which the terminal allows the information to be received) is set by first removing the plate on the data terminal located to the left of the keyboard. With a ball point pen or other object set the rate of 300 or 1200 Baud by pressing in the appropriate switch. The previous setting (usually, 9600) must be disabled by resetting the 9600 switch. The operator should then reset the system. The Distribution program loads in the starting and ending picture row and column. A section of memory is cleared to make ready for fresh data. The intensity values in upper memory can range from 00H to FFH. If the value pointed to by the picture pointer (DI) was 3DH, location 3DH in lower memory would be incremented by 1. If all of upper memory was filled with the value 3DH then another counter would be needed since each address in lower memory can be increased to FFH maximum. Therefore, two addresses for each value are needed.

If the first address fills to FF and is incremented, the second address is incremented and the first cleared.

The picture pointer is set to the next position and a check is made for last row, last column. The process repeats until all memory values are tabulated.
After the distribution is stored into lower memory it must be printed or displayed on a data terminal to be of any use. Print distribution program takes these values and sends them to the data terminal along with the appropriate control characters, Fig. 3.2-1.

If the distribution is such that there is a gap of no area between two values, two stars are printed in place of intensity numbers. This procedure saves paper. After each intensity value is printed, the number or area is printed followed by a series of stars. The number of stars is equal to the area divided by a constant. The constant can be changed to scale the ordinate of the graph.

When all the intensity and area numbers are printed the program returns to Master loop.

Note that Print distribution can be run without destroying the stored values from Distribution.
FIG. 3.2-1. Output Print Distribution.
Distribution (DISTRB) and Print Distribution (PTDIST)

SUBROUTINES: CHAR

REGISTERS: AX - starting row, column
EX - ending row, column
PI - picture pointer
CS - number of locations counter

VARIABLES: 36D3H-36E3H TABL - ASCII numbers for printing

INPUTS: Values 00H-FFH from picture array

OUTPUTS: Values to 400H-600H in lower memory and ASCII values to data terminal.

LOCATION: 3580H

PTDIST
STARTING ADDRESS: 3600H

SIZE: 173H

COMMENTS: Should be run after a picture is loaded to upper memory.

VERSION: 9 4 20.26
Light Meter (LMETER)

3.3 The Light meter routine finds the average light intensity from a block of picture memory (256 points) and displays the result on the data terminal.

This utility program is used to set the light intensity seen by the charge injection device (CID) imaging circuit in the camera. To run the program the proper symbol must be entered into the Master loop monitor command string. After typing a carriage return the program is in a loop waiting for the operator to take a picture. After taking a picture the average intensity is displayed. The operator should adjust the f-stop on the lens until the average is at the proper level.

The program immediately calls the Next picture program to operate the focusing and flood lamps and takes care of loading the image to picture memory.

The data terminal screen is cleared to make ready for output information. The row counters are cleared and a check is made to see if a key was pressed at the data terminal. This is the escape from program execution. If a key was not pressed the sum for averaging is cleared. The pointer DI is set to location 6766H in picture memory. The value pointed to by DI is added to the sum. DI is moved two positions to the right and the value pointed to is again added to the sum. The process continues across the row until 32 points are summed. The pointer is moved down three rows and the process repeats until 32 points in 8 rows are summed. The sum is divided by 256 for the average intensity.

The next section of the program displays one of the four scales shown in Fig. 3.3-1. Scale 2 shows an average level of 86.
To determine which scale is displayed the average intensity is compared to 75. If less than 75 scale one is displayed. If greater than 75 but less than 151 scale two is displayed, etc. The exclamation mark is displayed to show the level. The entire program repeats until a key on the data terminal is pressed.
FIG. 3.3-1. Light Meter Scales.
Light Meter (LMETER)

SUBROUTINES: CHRO, NEXPIC

REGISTERS: SI - row character pointer
AX - input, output, general
BX - scale counter (pointer)
DX - sum value
DI - picture pointer

VARIABLES: D64AH STRFLG
48FDH-49E6H MSCO1 Scale 1
49E7H-4AD0H MSCO2 Scale 2
4AD1H-4EBAH MSCO3 Scale 3
4BBBH-4C1FH MSCO4 Scale 4

INPUTS: Now classified values 00-FFH from picture memory

OUTPUTS: ASCII characters to data terminal representing
relative light intensity.

LOCATION: 4800H

SIZE: 41FH

COMMENTS: This program loads the picture automatically.
Previous picture data is destroyed.

VERSION: 8 10 14.15
Communication to Nova (CONNOV)

3.4 The purpose of the module is to transfer picture memory to the Nova/Grinnell image processing system. The image processing system has, at the present time, matrix handling as well as disk and magnetic tape storage capabilities.

The proper hardware connections should be made between the Nova and the data terminal on the beef grading instrument. The RS232 terminal cable from the Nova needs to be connected to the extension output of the ADM data terminal. After data is loaded into picture memory the program is started by typing the proper symbol into the Master loop. A carriage return is typed to start execution. The program should start before the Nova comes on line to request data. After the program receives the proper message from the Nova receiver, an on-line message is sent to the Nova. The routine sends one row of picture data then waits for receiver ready. The process is repeated until all rows are sent.
Communication to NOVA (COMNOV)

SUBROUTINES: CHARIN, CHAROT, MESSOUT

REGISTERS: AL - input, output data
           SI - pointer for message routines
           DH - top row
           DL - left column
           DI - picture pointer

VARIABLES: 874-884H MESS1 - on line code to NOVA
385-888H MESS2 - received data code
389H DUMMY

INPUTS: Data from picture array. Usually not classified.

OUTPUTS: Input data to output ports to NOVA

Ports: STATUS ODEH
       DATA O18H

LOCATION: 810H

SIZE: 8FH

COMMENTS:

VERSION: 8 15 9.54
Set Date and Location (SETDAT)

3.5 SETDAT is a stand alone program starting execution at 6200H. SETDAT inputs from the keyboard the month, day, year, and a three letter identifier for the location. The string is stored in memory starting at 9F8H. The data is stored as a number and the location is stored in ASCII. For a date of 5/26/81, for example, the string as seen in memory would be as follows: (note all values are in hexadecimal notation) 26H, 05H, 4DH, 81H, 4EH, 41H. The location was MAN for Manhattan. The data is stored as words; 05, 26, 81, M, A, N.
Set Date and Location (SETDAT)

SUBROUTINES: MESSAGE, INPINT, CHARIN

REGISTERS: SI - message pointer
AX - input, output

VARIABLES: 9F8H - DAT
9FAH - DAT 2
9FCH - LOC
624DH-625DH MESS1 month ? message
625EH-626CH MESS2 day ? message
626DH-6275H MESS3 year ? message
6276H-627DH MESS4 location ? message

INPUTS: Values of month, day, year and location from data terminal.

OUTPUTS: Storage into memory

LOCATION: 6200H

SIZE: 7DH

COMMENTS: Set date and location must proceed Initialize data disk routine.

VERSION: 3 19 14.09
Light Calibration (LGTCAL)

3.6 The purpose of Light calibration is to ultimately load in multipliers to the multiplier memory. The routine will store values to linearize the video image with respect to light from the flood lamps.

The initial value to be determined is the peak or average value of an area of the stored image. To obtain an average value the light meter flag (LMFLAG) is set to 00H. To get a peak value, LMFLAG is set to 01.

After calling LMETER (description 3.3 in this manual) the location LMRVAL contains the average or peak value.

The multiplier routine (DDDD) is called to find the multiplier for a value in picture memory pointed to be DI.

\[
\text{(pointer DI)(multiplier) = average value}
\]

A multiplier for each value of picture memory is found and stored in the multiplier memory.

The multipliers are encoded to indicate whether the DI value was larger or smaller than the average value.

For example if the average was 47H and the DI value was 45 the multiplier would be

\[
\frac{1.0H}{45} \text{ with a remainder of } 2. \text{ The remainder needs to be divided again by } 45H \text{ to obtain } 07H.
\]

\[
\frac{0.07H}{45} \rightarrow 02.00H
\]

Since 45H is smaller than 47H it is encoded as such by clearing the carry bit and shifting to the right. By shifting, one bit of resolution is lost.
00000111
00000011

The multiplier is stored as 03H.

During the Preprocessor routine the multiplier is loaded back and shifted left. The carry was cleared so the multiplicand will be added to image memory.

\[
\begin{array}{c}
45H \\
\times \ 06H \\
01.9EH
\end{array}
\]

The value of 01H is added to 45H to obtain 46H or 47H $\pm$ 1 bit. If the value in DI was greater than 47H, then it will be encoded as negative (carry set) and a subtraction will take place in Preprocessor. When the multipliers for every position are determined and stored in multiplier memory the program returns to Master loop.
Light Calibration (LGTCAL)

SUBROUTINES: LMETER, DDDD

REGISTERS: CX - general, temp. storage LMRVAL
            DI - picture pointer
            ES - extra segment to 2000H
            DS - data segment to 1000H
            AL - average or peak value, general
            BL - plus or minus flag
            DL - general

VARIABLES:  88C0-LMRVAL
            C8C2-LMFLAG
            08C4-NUMZERO

INPUTS: Average or peak value from LMETER program.
        Unclassified values from picture memory.

OUTPUTS: Multipliers to multiplier memory located at 20000H

LOCATION: 2060H

SIZE: 65H

COMMENT: Light calibration is usually run at the beginning of a beef
         grading session. If the multipliers are to be retained they
         may be stored on diskette and reloaded.

VERSION: 8 10 13.52
Line Load (LINLOD)

3.7 The Line load utility's purpose is to load the same line from successive pictures into multiplier memory to test camera repeatability.

The SI register points to the desired line out of the picture. The output and input ports are initialized. The picture is taken with no control signals to the focus or flood lights and loaded into picture memory 10000H to 1FFFFH. The selected line is pulled out and stored into the first line of multiplier memory 20000H. The multiplier memory pointer (DI) is moved to the next line and the process repeats storing the selected line into 20001H. When all multiplier memory is loaded the program returns to Master loop.
Line Load (LINLOD)

SUBROUTINES: None

REGISTERS: CX - general purpose, counter
DI - destination index
SI - source index, line loaded
AL - port control, data

VARIABLES: None

INPUTS: Line of data from each stored picture image.

OUTPUTS: Line storage to multiplier memory.

LOCATION: 4DOOH

SIZE: 36H

COMMENT: This utility is used for camera evaluation.

VERSION: 9 8 15.55
Exclusive-or (XOR)

3.8 The XOR utility is a simple data check scheme. The area checked is between the 10000H and 20000H physical address space.

The program sets the pointer to the address 10000H and loads in the first value. The pointer is incremented and the value at 10001H is exclusive Or'ed with the previous value. The process continues until location 1FFFFDH is reached. The old value of the exclusive oring process is stored at 1FFFFEH and 1FFFFFH. If a change has occurred between the old memory block and the new memory block the new XOR value will not match the old value. In this case the new value is stored into the old location and a message is output to the terminal indicating a change of data has occurred.

The normal procedure for running XOR is to load the EPROM program down into memory, read in a picture or program disk, then run the XOR program to see if the program disk programs were unchanged. After the XOR program is run the program disk contents between 10400H and 17FFFH can be loaded into work addressable memory space if desired.
Exclusive-or (XOR)

SUBROUTINES: MESSAGE

REGISTERS: CX - general purpose
DI - picture pointer
DX - Exclusive-or register

VARIABLES: 153DH-154DH MESS4

INPUTS: Values from picture memory, classified or unclassified.

OUTPUTS: Stored Exclusive-or code into picture memory with a message to the data terminal.

LOCATION: 1500H

SIZE: 4DH

COMMENTS: XOR resides in the start-up EPROM. It can be used on program as well as on image.

VERSION: 9 16 18.00
Classifier Level Change (CLLVCH)

3.9 The Classifier level change utility retrieves and stores classifier levels from the data terminal to a specific memory location.

Initially the message pointers are set and a message is displayed on the data terminal asking for the lower classifier level. The old classifier level is displayed and the program is now prompting for input. If a valid number is entered it is stored into a memory location. If an improper number is entered the program skips to the next section and nothing is stored in memory. The same procedure occurs for the upper level except that upon completion the program returns to the Master loop program.
Classifier Level Change (CLLVCHR)

SUBROUTINES:  CHAR, INPINT, MESSAGE

REGISTERS:  BX - hex table pointer
            SI - message pointer
            CL - counter for shifts
            AL - general purpose transfer

VARIABLES:  8C8H LWRLV1
            8C9H LWRLV2
            8CAH UPRLV1
            8CPH UPRLV2
            126C-1295 MESS1 - lower classifier message
            1296-12BF MESS2 - upper classifier message
            12CO-12CF hex table

INPUTS:  Values from program memory and/or the keyboard.

OUTPUTS:  Values stored back into memory for classifier levels.

LOCATION:  1200E

SIZE:  CFH

COMMENTS:  Classifier level change is the program by which quick level changes may be made.

VERSION:  9 5 14.18
Recipe (RECIPE)

3.10 The Recipe routine moves a total of 32 characters to the Master loop program.

The characters or program list (e.g. PABCHCQ) must already reside in program memory.

To run the standard recipe only a (\) backslash need be typed followed by a carriage return.

PROCEDURE FOR LOADING A NEW RECIPE

1. Type G1E00 [CR]

System => 2. PROGRAMS?

3. PABCGHCQ or new string characters.

4. Do not type [CR]

5. Reset the system.

6. Type U,U

System => 7. 86/12 monitor message.

8. Type G7045 (stores the new string).

9. Type G1E00 (back to Master loop).
Recipe (RECIPE)

SUBROUTINES: None

REGISTERS: CX - counter
            SI - location of recipe string, master string (SENTRY)
            DI - location of master string, recipe string (SENTRY)

VARIABLES: 1F80H CNT1

INPUTS: Values from recipe string or master string.

OUTPUTS: Values to master string or recipe string.

LOCATION: 7030H RECIPE
            7045H SENTRY

SIZE: 20H

COMMENTS: Recipe makes easy the running of strings of programs by
typing in only one symbol in Master loop.

VERSION: 9 14 21.52
Disk Support Package (SUBS)

4.1 SUBS are the subroutines necessary for control of the floppy disk control board, and general use utilities. Twelve routines are in this package. They are: MESSAGE, CHARIN, CHAROT, DMASET, VFYDRVRDY, COMMANDBUZY PERMREG, RESULT, OUTHEX, DECHEX, INPINT, and DECMIL.

MESSAGE
- This routine outputs an ASCII string of characters starting at the address passed in the SI register and ending at an ASCII null character (00). The output goes to the terminal.

CHARIN
- This routine inputs an ASCII character from the terminal keyboard, masks off the most significant bit (not used in ASCII characters) and uses CHAROT to echo the character to the terminal display screen.

CHAROT
- This routine outputs to the display screen an ASCII character passed to it.

DMASET
- DMASET sets the floppy disk controller's;
  1) base address register,
  2) starting memory address register,
  3) the DMA enable bit and
  4) the DMA control register,
by outputting to its proper port addresses.

VFYDRVRDY
- This routine is to verify the selected drive is ready by outputting a read drive status command of the drive specified in DRV. A drive goes "not ready" on power up or when the drive door is opened. To clear the "drive not ready" line the read drive status command must be output twice.
COMMANDBUSY - This is also a wait routine. It waits for the command busy line of the FDC to go inactive. (Ready for another command byte.)

PARMREG - This is also a wait routine. It waits for the parameter register full line to go inactive. (Ready for another parameter byte.)

RESULT - This routine waits for the command to complete its execution and retrieves the result byte.

OUTHEX - This routine takes the byte passed to it and converts the byte to the ASCII representation of that byte and uses CHAROT to output the ASCII to the terminal screen.

DECHEX - This routine converts the number passed to it to a hexadecimal equivalent of that number. NOTE: this routine assumes the number passed is a valid BCD number. The result is passed in the AX register.

INPINT - This routine inputs a string of numbers until a [CR] is pressed. Then takes the last four and converts it from ASCII to a hexadecimal number. The number is returned in the AX register.

DECMIL - This routine takes the hexadecimal number passed to it and converts it to its equivalent decimal value. The result is returned in AX register.
4.2 INTDAT constructs the directory for a data storage disk, then writes that directory on the disk in drive #0. The directory consists of the month (1-12), day (1-31), year (0-99), location (any three ASCII characters), and the carcass count of zero. The directory is written on sector one of track one on drive #0.

INTDAT is a stand alone program starting execution at 6100H.

NOTE: The set date routine must have been run prior to this program.
Initialized Data Disk (INTDAT)

SUBROUTINES: DSKIO, MESSAGE, OUTHEX, DMASET, VFYDRVFDY,
              COMMANDBUSY, PARMREG, RESULT, ERROR

REGISTERS: CX - counter
           DI - buffer pointer
           SI - data pointer, message pointer
           AX - input, output
           DX - port address

VARIABLES: 9F8H - DAT
            9FAH - DAT 2
            9FCH - LOC 1
            9FEH - SAVCNT
            A00H - PCKDEC
            FOOH - COMMAND
            F01H - TRK
            F08H - BSEADDR
            F0CH - DRV
            F12H - CTRL
            617BH - MESS1 disk error message

INPUTS: Input values from Set date and location

OUTPUTS: Constructed directory on disk in drive Ø

LOCATION: 6100H

SIZE: 83H

VERSION: 3 19 10.29
Disk Format (FORMAT)

4.3 FORMAT meets the requirements of the floppy disk controller (FDC) for formatting a floppy disk to IBM standard format (128 bytes per sector, 26 sectors per track, and 76 usable tracks per disk) for single density floppy disks. To meet the requirements of the FDC, FORMAT creates a table of numbers the FDC manual calls a format table. The table consists of (in order): track addresses, head address, sector address, and sector length for each of the 26 sectors of a track. The table is for only one track. The numerically consecutive sectors (1, 2, 3, etc.) are spaced or interleaved three apart. This means the adjacent sectors on a track are numbered; 1, 10, 19, 2, 11, 20, 3, 12, 21, 4, 13, 22, 5, 14, 23, 6, 15, 24, 7, 16, 25, 8, 17, 26, 9, 18. After this table is written for track zero just the track addresses and head addresses are updated for the remaining 76 tracks. The error bit is checked after each track is formatted and if an error is detected the routine outputs the error code to the terminal and terminates execution. The buffer for the format table is at 7100H in the system memory. Disk format is a stand alone program set to format a write enabled disk in drive 1.
Disk Format (FORMAT)

SUBROUTINES: VFYDRVRDY, TABLE, DMASET, COMMANDBUSY, PARMREG, RESULT,
ERROR, MESSAGE, OUTHEX, CRLF

REGISTERS: AX - general purpose input, output
DX - port address
BX - track pointer
SI - base pointer, pointer to format table/message pointer
CX - sector count

VARIABLES: 0F00H - COMMAND
0F01H - TRK
0F08H - BSEADDR
0F0AH - ADDR
0F0CH - DRV
60DFH - MESS1 disk error number message

INPUTS: None

OUTPUTS: Formatted data disk

LOCATION: 6000H

SIZE: E7H

COMMENT: Format disk must be run on a blank data disk before trying
to store disk data. Any previous disk data will be purged.

VERSION: 3 18 20.52
Binary Read From Disk (DSKBIN)

4.4 DSKBIN inputs from the keyboard a block address and number of blocks then reads the blocks from the disk in drive #1. The data blocks are on the disk in a PDP/11 format.

A PDP/11 block structure is as follows:

<table>
<thead>
<tr>
<th>BLOCK ADDRESS</th>
<th>TRACK ADDRESS</th>
<th>SECTORS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FIRST</td>
<td>SECOND</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>11</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>17</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>25</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>16</td>
</tr>
<tr>
<td>6</td>
<td>1/2</td>
<td>24</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>11</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>19</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>18</td>
</tr>
<tr>
<td>12</td>
<td>2</td>
<td>26</td>
</tr>
<tr>
<td>13</td>
<td>3</td>
<td>13</td>
</tr>
<tr>
<td>14</td>
<td>3</td>
<td>21</td>
</tr>
<tr>
<td>15</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>16</td>
<td>3</td>
<td>11</td>
</tr>
<tr>
<td>17</td>
<td>3</td>
<td>20</td>
</tr>
<tr>
<td>18</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>19</td>
<td>3/4</td>
<td>10</td>
</tr>
<tr>
<td>20</td>
<td>4</td>
<td>23</td>
</tr>
</tbody>
</table>
This continues in the same manner for all the 492 blocks.

To translate the block numbers inputted to the proper track and sector addresses the equations look like these:

\[
\text{TRK} = \text{INTEGER } (I/26)+1 \\
\text{SEC} = [\text{SECI}(I+1) \text{MOD} 26] + 6(\text{TRK} - 1) \text{MOD} 26
\]

with \( I \) being \( \text{BLKx}4 \) for the first sector of the block, \( I=\text{BLKx}4+1 \) for the second sector and so on for the third and fourth sectors. The string SECI in the above equation is

<table>
<thead>
<tr>
<th>location</th>
<th>SECI</th>
<th>location</th>
<th>SECI</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>26</td>
<td>13</td>
<td>25</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>14</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>15</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>16</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>17</td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td>9</td>
<td>18</td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td>11</td>
<td>19</td>
<td>12</td>
</tr>
<tr>
<td>7</td>
<td>13</td>
<td>20</td>
<td>14</td>
</tr>
<tr>
<td>8</td>
<td>15</td>
<td>21</td>
<td>16</td>
</tr>
<tr>
<td>9</td>
<td>17</td>
<td>22</td>
<td>18</td>
</tr>
<tr>
<td>10</td>
<td>19</td>
<td>23</td>
<td>20</td>
</tr>
<tr>
<td>11</td>
<td>21</td>
<td>24</td>
<td>22</td>
</tr>
<tr>
<td>12</td>
<td>23</td>
<td>25</td>
<td>24</td>
</tr>
</tbody>
</table>

The INTEGER function takes the parameter and truncates the fraction and \( \text{MOD} 26 \) is a modulo divide.
The data read from the disk is a string of ASCII characters in paper tape format. The string is stored in a buffer called BUFF. Once in BUFF the data is then converted and stored in memory using a paper tape read routine called READIO.

DSKBIN is a stand alone program starting execution at 5000H.
Binary Read From Disk (DSKBIN)

SUBROUTINES: MESSAGE, INPINT, DECHEX, READONE, RESULT, ERROR, READIO, CHARIN, DMA1, VFYDRVRDY, COMMANDBUSY, PARMREG, BYTEI, WORDI, CONVHEX, OUTHEX, CHAROT

REGISTERS: SI - message pointer
AK - block number, general purpose, track number, hex converted number, input, output
CX - counter
BX - general purpose

VARIABLES: 5300H BLK
5302H LEN
5304H I
5305H TS
5306H TRK
5307H SEC
5308H CRITL
5309H DRV
530AH DRIVET
530BH TEMP
530DH-531CH HEX
531DH-5324H NUM
5325H-5332H SECI
5333H-533EH SECI2
533FH-5363H MESS1 read object code
5364H-5376H MESS2 starting block
5377H-538BH MESS3 length in blocks
538CH-5394H MESS4 Done?
5395H-53BDH MESS5 disk error
53B1H-53C7H MESS6 checksum error
53C8H BUFF

INPUTS: Values from disk in PDP/11 block format

OUTPUTS: Messages to data terminal. Data to proper location in memory.

LOCATION: 5000H

SIZE: 24BH

COMMENTS: Preceded by Master loop. Data is in standard paper tape format.

VERSION: 0

188
DSKBIN Procedure

1. Get into the 86/12 monitor. If in the program mode, type in a Q and two carriage returns.

2. Load transfer disk into drive #1 making sure that the metallic strip is removed from the disk.

3. Type G5000[CR].

4. The system will prompt for the starting block on the transfer disk. Enter the proper block corresponding to the program to be loaded and type [CR].

5. The system will prompt for the number of blocks used to store the program. Read this number from the directory listings and type [CR].

6. The system will respond with Done? Type a 'Y' if you wish to enter more programs from the transfer disk, otherwise type 'N' and exit the program. The program will automatically load into the correct position in memory.
Read (RDDSK)/Write (WRDSK) Disk Picture

4.5 DISPIC transfers picture data (256x244 bytes) between upper memory and one of four locations on disk drive #1. The starting track and sector addresses for the four locations (pictures) on a disk are as follows:

<table>
<thead>
<tr>
<th>PICTURE NUMBER</th>
<th>STARTING TRACK</th>
<th>STARTING SECTOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>20</td>
<td>19</td>
</tr>
<tr>
<td>2</td>
<td>39</td>
<td>13</td>
</tr>
<tr>
<td>3</td>
<td>58</td>
<td>07</td>
</tr>
<tr>
<td>4</td>
<td>01</td>
<td>25</td>
</tr>
</tbody>
</table>

These addresses are indexed in table TRKSET by the inputted picture number. The picture is then read or written sequentially by sector and by track.

DISPIC has three entry points:

- **RDDSK** - This routine inputs a picture number from the keyboard and transfers the corresponding picture from the disk to memory.

- **WRDSK** - This routine writes the picture number found in variable CNT3. CNT3 is incremented for sequential writing onto the disk.

- **NSWRDSK** - This routine inputs a picture number from the keyboard and transfers memory to the corresponding picture location on the disk.
Read (RDDS/Write (WRDSK) Disk Picture

SUBROUTINES: MESSAGE, CRLF, CHAROT, CHARIN, SETPIC, PICT, VFYDRVRDY,
             DMASET, RUNSEC, DMAI, COMMANDBUSY, PARMREG, RESULT,
             ERROR, OUTHEX

REGISTERS:   SI - message pointer
             AX - input, output, general purpose, track and sector
             CX - counter
             DX - general purpose

VARIABLES:   0F17 HEX hex characters
             7500-7517 TRKSET

INPUTS:      Values from/to disk.

OUTPUTS:     Values to/from picture memory.

LOCATION:    2400H

SIZE:        13EH

VERSION:     3 13 10.32
References


Test Patterns

A.1 Two basic test patterns are stored on floppy disk to test programs such as Area acquisition, Box distribution, Max point and others. The patterns may also be used for program modification.

The test patterns are provided to help the operator load in a known shape quickly instead of entering picture data point by point.

The first pattern shown (#2 on Test Pattern disk) was used for testing Max point and Fat pieces, Fig. 4.1-1. It can be seen that there are 6 fat pieces as well as 1 section of background.

From the top left point 1F65 one can find the address of any point in the display, e.g., the top most boundary point is located at 1F84.

- top most point 1F84H
- left most point 2C6AH
- bottom most point 356FH
- right most point 2890H

There is a total of 104 decimal boundary points.

The second pattern (pattern #3 on Test pattern disk) was used mainly for Area acquisition, Max point, and Box distribution. Fig. A.1-2.

<table>
<thead>
<tr>
<th>hex</th>
<th>decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Meat in LEA</td>
<td>47H</td>
</tr>
<tr>
<td>Fat in LEA</td>
<td>13H</td>
</tr>
<tr>
<td>BKGD LEA</td>
<td>10H</td>
</tr>
<tr>
<td>Total Meat Count LEA</td>
<td>57H</td>
</tr>
<tr>
<td>Maximum top point</td>
<td>7971H</td>
</tr>
<tr>
<td>Maximum bottom point</td>
<td>8679H</td>
</tr>
<tr>
<td>Center point</td>
<td>7F76H</td>
</tr>
</tbody>
</table>
FIG. A.1-1. Test Pattern #2.

FIG. A.1-2. Test Pattern #3.
A.2 Start Up Procedure

The operator types what is indicated above the asterisks.

1. Plug in box.
2. Turn on fan and power switches.
3. Type the indicated
   a) UU
      **
   b) M9000,97FF,1000[CR]
      ********************* *
4. Wait for the prompt. (.)
5. Insert a blank disk in drive #0.
6. Insert PROGRAM DISK in drive #1.
7. Type the indicated
   a) G0000- XX 1000[CR]
      *     ***** *
8. This prompt will return
    READ
    ?
9. Type 1
10. Wait for the prompt. (.)
11. Type the indicated
    a) M1000:400,7FF,400[CR]
       ********************* *
12. Wait for the prompt. (.)
13. Insert current data disk in drive #0.
15. Type the indicated
    a) GXXXX- XX 6200[CR]
       ***** *
    b) MONTH 91-12) ? 5[CR] (CURRENT MONTH)
       * *
c) DAY (1-31)? 25[CR] (CURRENT DAY)
   ** *

d) YEAR? 82[CR] (CURRENT YEAR)
   ** *

e) LOCATION? CCN[CR] (PROPER LOCATION)
   *** *

f) GXXXX- XX 1E00[CR]
   *  **** *

g) PROGRAMS?
   * ************
START UP PROCEDURE EXAMPLE

ACTUAL SAMPLE OF STARTUP:

{
ISBC 86/12 MONITOR, VI.2
.M9000,97FF,1000
.G 0000- 50 1000
READ .
?1
@0000:1062
.M1000:400,7FF,400
.G 2434- B8 6200
MONTH (1-12)? 5
DAY (1-31)? 27
YEAR? 81
LOCATION: MAN
@0000:624D 0D
.G 624D- OD 1E00
PROGRAMS

200
A.3 Procedure For Making A New Program Disk

1. Using Binary Read from Disk, load into memory all programs to be stored on the new program disk.

2. Make any variable changes necessary.

3. Type M400, 7FF,1000:400[CR]. This command moves all the programs from lower or program memory to picture memory.

4. Use Read/Write Disk Picture to write the picture image (now programs) to the disk by typing:

   G1E00[CR]

   System → PROGRAMS?

   TQ[CR]

   System → WRITE?

   1 or whatever section on the disk you wish

   System → ,

   [CR] to write on.

   [CR] you should now be back in the 86/12 monitor.
SOFTWARE OPERATIONS MANUAL OF A
COMPUTERIZED BEEF GRADING INSTRUMENT

by

DON A. GILLILAND

B.S., Kansas State University, 1977

_____________________

AN ABSTRACT OF A MASTER'S THESIS

Submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Electrical Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1982
ABSTRACT

The computerized beef grading instrument was developed as an aid to the beef grader in making a more accurate analysis of the exposed cut surface of a carcass. The analysis includes a determination of fat area and lean meat area over the surface as well as the fat thickness covering the longissimus dorsi muscle. The software manual herein describes the algorithms and procedures necessary to perform the data acquisition and analysis.