Technical Document 1111
July 1987

UNIX Benchmark System

T. Brewer
Integrated Systems Analysts, Inc.

The views and conclusions contained in this report are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Naval Ocean Systems Center or the U.S. government.

Approved for public release. Distribution is unlimited.
NAVAL OCEAN SYSTEMS CENTER
San Diego, California 92152-5000

E. G. SCHWEIZER, CAPT. USN
Commander

R. M. HILLYER
Technical Director

ADMINISTRATIVE INFORMATION

This work was performed for Code 914, NAVOCEANSYSCEN. Contract N66001-84-D-0053 was carried out by Integrated Systems Analysts, Inc., Marina Gateway, 740 Bay Boulevard, Chula Vista, California 92010-5254.

Released by
R.L. Broersma, Head
System Software Branch

Under authority of
J. Gilbreath, Head
Computer Sciences and Simulation Division
A benchmark suite has been consolidated to test and evaluate a variety of computer systems and compare the results. VAX 11/780, under UNIX 4.3 BSD, has been selected as the baseline system to which each target system (procurement candidates) would be compared. The program, BENCH, collects and stores test results from all the target systems, and produces two reports. The first report compares any two systems that the user selects. The second report summarizes all the test data into a single report.

The suite presently has 18 tests and the user can specify which test may be used by modifying an ASCII file called BENCHLIST. Additional tests may be added to the suite by modifying the BENCHLIST and supplying the appropriate code.
## CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>1</td>
</tr>
<tr>
<td>Descriptive Summary</td>
<td>2</td>
</tr>
<tr>
<td>Fortran Tests</td>
<td>2</td>
</tr>
<tr>
<td>Sieve Tests</td>
<td>3</td>
</tr>
<tr>
<td>General Tests</td>
<td>3</td>
</tr>
<tr>
<td>Pre-Award Tests</td>
<td>4</td>
</tr>
<tr>
<td>Database Format</td>
<td>5</td>
</tr>
<tr>
<td>Reports</td>
<td>7</td>
</tr>
<tr>
<td>Comparison Report</td>
<td>9</td>
</tr>
<tr>
<td>UNIX Programmer's Manual</td>
<td>13</td>
</tr>
<tr>
<td>Instructions for Data Collection Using the Bench Program</td>
<td>15</td>
</tr>
<tr>
<td>Prime Numbers</td>
<td>17</td>
</tr>
<tr>
<td>Calling Sequence and Arguments Passing</td>
<td>18</td>
</tr>
<tr>
<td>Random Numbers</td>
<td>19</td>
</tr>
<tr>
<td>Fast Fourier Transform</td>
<td>21</td>
</tr>
<tr>
<td>Matrix Inversion</td>
<td>23</td>
</tr>
<tr>
<td>Polynomial Roots</td>
<td>25</td>
</tr>
<tr>
<td>C Sieve</td>
<td>28</td>
</tr>
<tr>
<td>Fortran Sieve</td>
<td>29</td>
</tr>
<tr>
<td>Pascal Sieve</td>
<td>30</td>
</tr>
<tr>
<td>Whetstone</td>
<td>31</td>
</tr>
<tr>
<td>Dhrystone</td>
<td>36</td>
</tr>
<tr>
<td>Block Write</td>
<td>45</td>
</tr>
<tr>
<td>Block Read</td>
<td>46</td>
</tr>
<tr>
<td>Sort (Part I)</td>
<td>47</td>
</tr>
<tr>
<td>Sort (Part II)</td>
<td>48</td>
</tr>
<tr>
<td>Sort (Part III)</td>
<td>50</td>
</tr>
<tr>
<td>Integer Arithmetic</td>
<td>52</td>
</tr>
<tr>
<td>Real Arithmetic</td>
<td>53</td>
</tr>
<tr>
<td>Large Data Space</td>
<td>54</td>
</tr>
<tr>
<td>Compile</td>
<td>55</td>
</tr>
</tbody>
</table>
Introduction

In March 1986, NAVOCEANSYSCEN tasked ISA to prepare a comprehensive test suite to systematically test and evaluate a variety of computer systems and compare the results. NAVOCEANSYSCEN supplied ISA with their Pre-Award test suite, with instructions to also include certain published tests, such as the Dhrystone and Whetstone benchmarks. It was ISA's task to consolidate all the designated tests into one easily-runnable program which can be used by either Government or Contractor personnel to test all computer systems that are candidates for procurement.

NAVOCEANSYSCEN selected the VAX 11/780 under UNIX 4.3 BSD as the baseline system to which each target system (procurement candidates) would be compared. Our program, bench, collects and stores test results from the baseline system, collects and stores results from all the target systems, and produces two reports. The first report compares any two systems which the user selects, and the second report summarizes all the test data in one report. The user tells the program which of these tasks he wishes performed through the use of options on the input line. All input is in standard UNIX format. For example,

% bench -b

would execute the test suite and store the results as the baseline data. See the enclosed man page for a list of all available options.

The suite includes eighteen tests at present. The user is able to specify which of the eighteen tests will be used by modifying an ASCII file called benchlist. Benchlist includes the names of all the tests. The user adds or deletes the comment indicator to tell the program to include or exclude that particular test. Additional tests may be added to the suite by modifying the benchlist and supplying the appropriate code.
Descriptive Summary
The following is a very brief description of the tests in the benchmark suite.

Fortran Tests

1. Prime Numbers
This program generates the prime numbers from 0 to 8192 (optionally printing out the results).

2. Calling Sequence and Argument Passing
This program initializes nine variables, passes them to a subroutine, which in turn has four assignments. This sequence is repeated one million times.

3. Random Numbers
This program tests the random number generator by generating 12,800 random numbers and checking the randomness.

4. Fast Fourier Transform
This program performs fast fourier transform using the decimation-in-time method (optionally printing out the results).

5. Matrix Inversion
This program performs matrix inversion using the Gauss-Jordan Reduction (optionally printing out the results).

6. Polynomial Roots
Roots of polynomials are calculated using the Bairstow's method (optionally printing out the results).
Sieve Tests

7. C Sieve
C version of the Sieve of Eratosthenes prime number program.

8. Fortran Sieve
Fortran version of the Sieve of Eratosthenes prime number program.

9. Pascal Sieve
Pascal version of the Sieve of Eratosthenes prime number program.

General Tests

10. Whetstone
A C version of the original Algol benchmark, "A Synthetic Benchmark" by H. J. Curnow and B. A. Wichman. Compiler optimization and floating point performance are tested.

11. Dhrystone
This program contains a distribution of statements which are considered to be representative: 53% assignment, 32% control statements, and 15% procedure and function calls.
Pre-Award Tests

12. Block Write
   This program creates a very large file by writing 8K byte blocks one after the other.

13. Block Read
   This program reads the file created by the block write program. The reads are executed in 8K byte blocks.

14. Sort
   A shell script to test the section 1 sort call. A file is sorted on a particular column and the result is compared to a presorted file to test the results of the sort.

15. Integer Arithmetic
   Addition, subtraction, multiplication, and division are performed on integer variables. The group of operations is executed 2.9 million times.

16. Real Arithmetic
   Similar to the integer arithmetic, this program performs addition, subtraction, multiplication, and division on real variables. This group of operations is executed 600,000 times.

17. Large Data Space
   This program references a data area larger than real memory making 20,000 references to random locations.

18. Compile
   This script compiles two C code files and loads the two object files into a single output file.
Database Format

The benchmark test data is stored in a series of files which reside in the current working directory. The file containing the baseline data is called baseline. A file is created for each system tested and is called targetXXX, where XXX is a 3-digit number assigned by the benchmark program and which is unique to each system. The format for both the baseline file and all the target files is identical.

The format of the database files illustrated on the following page.
<table>
<thead>
<tr>
<th>Time</th>
<th>Real Time</th>
<th>User Time</th>
<th>System Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>20.08</td>
<td>5.18</td>
<td>0.36</td>
<td></td>
</tr>
<tr>
<td>20.04</td>
<td>5.09</td>
<td>0.25</td>
<td></td>
</tr>
<tr>
<td>20.04</td>
<td>5.06</td>
<td>0.30</td>
<td></td>
</tr>
<tr>
<td>0.51</td>
<td>0.05</td>
<td>0.10</td>
<td></td>
</tr>
<tr>
<td>0.19</td>
<td>0.03</td>
<td>0.08</td>
<td></td>
</tr>
<tr>
<td>0.50</td>
<td>0.05</td>
<td>0.07</td>
<td></td>
</tr>
<tr>
<td>2.18</td>
<td>1.71</td>
<td>0.07</td>
<td></td>
</tr>
<tr>
<td>1.99</td>
<td>1.67</td>
<td>0.06</td>
<td></td>
</tr>
<tr>
<td>1.97</td>
<td>1.69</td>
<td>0.06</td>
<td></td>
</tr>
<tr>
<td>1.71</td>
<td>1.33</td>
<td>0.05</td>
<td></td>
</tr>
<tr>
<td>1.61</td>
<td>1.30</td>
<td>0.04</td>
<td></td>
</tr>
<tr>
<td>1.61</td>
<td>1.30</td>
<td>0.07</td>
<td></td>
</tr>
<tr>
<td>2.46</td>
<td>2.09</td>
<td>0.08</td>
<td></td>
</tr>
<tr>
<td>2.36</td>
<td>2.08</td>
<td>0.05</td>
<td></td>
</tr>
<tr>
<td>2.43</td>
<td>2.02</td>
<td>0.09</td>
<td></td>
</tr>
<tr>
<td>10.05</td>
<td>5.26</td>
<td>0.38</td>
<td></td>
</tr>
<tr>
<td>10.07</td>
<td>5.13</td>
<td>0.27</td>
<td></td>
</tr>
<tr>
<td>10.09</td>
<td>5.24</td>
<td>0.47</td>
<td></td>
</tr>
<tr>
<td>1.02</td>
<td>0.02</td>
<td>0.07</td>
<td></td>
</tr>
<tr>
<td>0.58</td>
<td>0.05</td>
<td>0.10</td>
<td></td>
</tr>
<tr>
<td>1.28</td>
<td>0.01</td>
<td>1.14</td>
<td></td>
</tr>
<tr>
<td>7.41</td>
<td>1.76</td>
<td>0.10</td>
<td></td>
</tr>
<tr>
<td>5.20</td>
<td>1.69</td>
<td>0.04</td>
<td></td>
</tr>
<tr>
<td>7.78</td>
<td>1.70</td>
<td>0.09</td>
<td></td>
</tr>
<tr>
<td>6.25</td>
<td>1.35</td>
<td>0.08</td>
<td></td>
</tr>
<tr>
<td>4.32</td>
<td>1.29</td>
<td>0.03</td>
<td></td>
</tr>
<tr>
<td>6.56</td>
<td>1.37</td>
<td>0.08</td>
<td></td>
</tr>
<tr>
<td>7.86</td>
<td>2.12</td>
<td>0.06</td>
<td></td>
</tr>
<tr>
<td>5.73</td>
<td>2.10</td>
<td>0.05</td>
<td></td>
</tr>
<tr>
<td>8.66</td>
<td>2.16</td>
<td>0.07</td>
<td></td>
</tr>
<tr>
<td>10.04</td>
<td>5.11</td>
<td>0.30</td>
<td></td>
</tr>
<tr>
<td>10.08</td>
<td>5.09</td>
<td>0.28</td>
<td></td>
</tr>
<tr>
<td>10.04</td>
<td>5.13</td>
<td>0.23</td>
<td></td>
</tr>
<tr>
<td>0.71</td>
<td>0.02</td>
<td>0.09</td>
<td></td>
</tr>
<tr>
<td>0.76</td>
<td>0.01</td>
<td>0.11</td>
<td></td>
</tr>
<tr>
<td>0.76</td>
<td>0.07</td>
<td>0.06</td>
<td></td>
</tr>
<tr>
<td>7.45</td>
<td>1.70</td>
<td>0.05</td>
<td></td>
</tr>
<tr>
<td>6.49</td>
<td>1.69</td>
<td>0.03</td>
<td></td>
</tr>
<tr>
<td>6.70</td>
<td>1.75</td>
<td>0.01</td>
<td></td>
</tr>
<tr>
<td>5.49</td>
<td>1.35</td>
<td>0.03</td>
<td></td>
</tr>
<tr>
<td>4.85</td>
<td>1.27</td>
<td>0.03</td>
<td></td>
</tr>
<tr>
<td>5.03</td>
<td>1.30</td>
<td>0.04</td>
<td></td>
</tr>
<tr>
<td>9.09</td>
<td>2.04</td>
<td>0.07</td>
<td></td>
</tr>
<tr>
<td>8.40</td>
<td>2.12</td>
<td>0.06</td>
<td></td>
</tr>
<tr>
<td>8.21</td>
<td>2.01</td>
<td>0.06</td>
<td></td>
</tr>
</tbody>
</table>

Three flags (0 OFF, 1 ON) for the three ways to execute the suite: singly, all at once, all at once with a load.

Each line contains three numbers:
1. Real time
2. User time
3. System time

Each group of three lines represents the results from one test and each line represents one repetition of the test. This is set in the C code (currently REP = 3).
Reports

*Bench* produces two reports: a comparison report based on two systems of the user's choice, and a summary report which includes all systems tested.

The comparison report is invoked when the user specifies the -p option on the command line. *Bench* displays a list of those systems in its database, and prompts the user to choose two systems from the list. The comparison report displays elapsed time, user time, system time, and percent usage for each test and each system chosen. A composite is also displayed. The composite is a sum of all systems chosen and represented as if it were a separate test. The elapsed time is the total amount of time that is consumed; the "clock" time. The user time is the amount of time the process spent executing nonprivileged instructions (e.g., arithmetic calculations, sorting, searching, etc.). System time is the time the process spent executing privileged (kernel) commands, such as system calls, plus some system-level overhead. The percent usage is that portion of the elapsed time that is actually spent executing the command. It is calculated thusly:

\[
\text{percent usage} = \left( \frac{\text{system time} + \text{user time}}{\text{elapsed time}} \right) \times 100
\]

The lowest elapsed time for each test for each system is indicated on the report by an asterisk (*). A separate column is displayed for the elapsed ratio. The first figure in the elapsed ratio column is the lowest time ratio, which is the ratio of the lowest elapsed time of the second system to the lowest elapsed time of the first system, or

\[
\text{lowest time ratio} = \frac{\text{lowest elapsed time of second system}}{\text{lowest elapsed time of first system}}
\]
The second figure of the elapsed ratio is the average time ratio. The average time ratio is the average elapsed time of the second system divided by the average elapsed time of the first system, or

\[
\text{average time ratio} = \frac{\text{average elapsed time of second system}}{\text{average elapsed time of first system}}
\]

At the bottom of the report, an average elapsed ratio is given, based on all tests except the composite. The average elapsed ratio is the average of each lowest time ratio (first figure) and the average of each average time ratio (second figure).

The summary report displays elapsed time averages for all systems tested. The test results are normalized to the baseline system. The summary report is invoked when the user specifies the -e option on the command line.

Sample output for both comparison report and the summary report follow.
## Comparison Report

Test executed one at a time with no extra load (expressed in seconds)

### DIGITAL VAX 11/780 4.3 BSD

<table>
<thead>
<tr>
<th>Type</th>
<th>Test Number</th>
<th>Elapsed Time</th>
<th>User Time</th>
<th>System Time</th>
<th>Percent Usage</th>
<th>Test Number</th>
<th>Elapsed Time</th>
<th>Elapsed Ratio *</th>
<th>User Time</th>
<th>System Time</th>
<th>Percent Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>composite</td>
<td>1</td>
<td>20.08</td>
<td>5.18</td>
<td>0.36</td>
<td>28</td>
<td>1</td>
<td>20.07*</td>
<td></td>
<td>5.26</td>
<td>0.42</td>
<td>28</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>20.04</td>
<td>5.09</td>
<td>0.25</td>
<td>27</td>
<td>2</td>
<td>20.08</td>
<td></td>
<td>5.18</td>
<td>0.60</td>
<td>29</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>20.04*</td>
<td>5.06</td>
<td>0.30</td>
<td>27</td>
<td>3</td>
<td>20.09</td>
<td></td>
<td>5.20</td>
<td>0.58</td>
<td>29</td>
</tr>
<tr>
<td>avg</td>
<td></td>
<td>20.05</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>20.08</td>
<td>[1.00, 1.00]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>uptime</td>
<td>1</td>
<td>0.51</td>
<td>0.05</td>
<td>0.10</td>
<td>29</td>
<td>1</td>
<td>0.78*</td>
<td></td>
<td>0.04</td>
<td>0.11</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0.91</td>
<td>0.03</td>
<td>0.08</td>
<td>12</td>
<td>2</td>
<td>0.81</td>
<td></td>
<td>0.05</td>
<td>0.11</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.50*</td>
<td>0.05</td>
<td>0.07</td>
<td>24</td>
<td>3</td>
<td>1.42</td>
<td></td>
<td>0.02</td>
<td>0.09</td>
<td>8</td>
</tr>
<tr>
<td>avg</td>
<td></td>
<td>0.64</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.00</td>
<td>[1.52, 1.56]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>whetstone</td>
<td>1</td>
<td>2.18</td>
<td>1.71</td>
<td>0.07</td>
<td>82</td>
<td>1</td>
<td>2.66*</td>
<td></td>
<td>1.67</td>
<td>0.09</td>
<td>86</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1.99</td>
<td>1.67</td>
<td>0.06</td>
<td>87</td>
<td>2</td>
<td>3.24</td>
<td></td>
<td>1.71</td>
<td>0.16</td>
<td>58</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>1.97*</td>
<td>1.69</td>
<td>0.08</td>
<td>89</td>
<td>3</td>
<td>3.07</td>
<td></td>
<td>1.77</td>
<td>0.15</td>
<td>63</td>
</tr>
<tr>
<td>avg</td>
<td></td>
<td>2.05</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2.99</td>
<td>[1.35, 1.46]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>integer arith</td>
<td>1</td>
<td>1.71</td>
<td>1.33</td>
<td>0.05</td>
<td>81</td>
<td>1</td>
<td>2.97</td>
<td></td>
<td>1.44</td>
<td>0.09</td>
<td>52</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1.81</td>
<td>1.30</td>
<td>0.04</td>
<td>83</td>
<td>2</td>
<td>2.62</td>
<td></td>
<td>1.33</td>
<td>0.14</td>
<td>58</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>1.81*</td>
<td>1.30</td>
<td>0.07</td>
<td>85</td>
<td>3</td>
<td>2.41*</td>
<td></td>
<td>1.38</td>
<td>0.11</td>
<td>62</td>
</tr>
<tr>
<td>avg</td>
<td></td>
<td>1.64</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2.67</td>
<td>[1.50, 1.62]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>real arith</td>
<td>1</td>
<td>2.46</td>
<td>2.09</td>
<td>0.08</td>
<td>88</td>
<td>1</td>
<td>2.75*</td>
<td></td>
<td>2.10</td>
<td>0.11</td>
<td>80</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>2.36*</td>
<td>2.08</td>
<td>0.05</td>
<td>90</td>
<td>2</td>
<td>4.01</td>
<td></td>
<td>2.09</td>
<td>0.15</td>
<td>56</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>2.43</td>
<td>2.02</td>
<td>0.09</td>
<td>87</td>
<td>3</td>
<td>3.58</td>
<td></td>
<td>2.12</td>
<td>0.18</td>
<td>64</td>
</tr>
<tr>
<td>avg</td>
<td></td>
<td>2.42</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3.45</td>
<td>[1.17, 1.43]</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Average Elapsed Ratios** **[1.11, 1.21]**

* Marks lowest elapsed time for the particular test
** Averages of all the test ratios
* Ratios are displayed as the [lowest time ratio, average time ratio]
## Comparison Report

Test executed all at once with no extra load (Expressed in seconds)

### DIGITAL VAX 11/780 4.3 BSD

<table>
<thead>
<tr>
<th>Type</th>
<th>Test Number</th>
<th>Elapsed Time</th>
<th>User Time</th>
<th>System Time</th>
<th>Percent</th>
<th>Elapsed Time</th>
<th>System Time</th>
<th>User Time</th>
<th>Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Composite</td>
<td>1</td>
<td>10.05*</td>
<td>5.26</td>
<td>0.38</td>
<td>56</td>
<td>10.09</td>
<td>5.40</td>
<td>0.57</td>
<td>59</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>10.07</td>
<td>5.13</td>
<td>0.27</td>
<td>54</td>
<td>10.08</td>
<td>5.31</td>
<td>0.44</td>
<td>57</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>10.09</td>
<td>5.24</td>
<td>0.47</td>
<td>57</td>
<td>10.07*</td>
<td>5.12</td>
<td>0.42</td>
<td>55</td>
</tr>
<tr>
<td>Avg</td>
<td></td>
<td>10.07</td>
<td></td>
<td></td>
<td></td>
<td>10.08</td>
<td>[1.00, 1.00]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>uptime</td>
<td>1</td>
<td>1.02</td>
<td>0.02</td>
<td>0.09</td>
<td>11</td>
<td>2.57</td>
<td>0.08</td>
<td>0.07</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0.58*</td>
<td>0.05</td>
<td>0.10</td>
<td>26</td>
<td>1.67</td>
<td>0.02</td>
<td>0.11</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>1.28</td>
<td>0.01</td>
<td>0.14</td>
<td>12</td>
<td>1.01*</td>
<td>0.02</td>
<td>0.11</td>
<td>13</td>
</tr>
<tr>
<td>Avg</td>
<td></td>
<td>0.96</td>
<td></td>
<td></td>
<td></td>
<td>1.75</td>
<td>[1.74, 1.82]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Whetstone</td>
<td>1</td>
<td>7.41</td>
<td>1.78</td>
<td>0.10</td>
<td>25</td>
<td>6.89</td>
<td>1.72</td>
<td>0.19</td>
<td>28</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>5.20*</td>
<td>1.69</td>
<td>0.04</td>
<td>33</td>
<td>7.05</td>
<td>1.76</td>
<td>0.13</td>
<td>27</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>7.78</td>
<td>1.70</td>
<td>0.09</td>
<td>23</td>
<td>6.20*</td>
<td>1.70</td>
<td>0.08</td>
<td>29</td>
</tr>
<tr>
<td>Avg</td>
<td></td>
<td>6.80</td>
<td></td>
<td></td>
<td></td>
<td>6.71</td>
<td>[1.19, 0.99]*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Integer arith</td>
<td>1</td>
<td>6.25</td>
<td>1.35</td>
<td>0.08</td>
<td>23</td>
<td>6.26</td>
<td>1.46</td>
<td>0.08</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>4.32*</td>
<td>1.29</td>
<td>0.03</td>
<td>31</td>
<td>5.85</td>
<td>1.36</td>
<td>0.05</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>6.66</td>
<td>1.37</td>
<td>0.08</td>
<td>22</td>
<td>5.34*</td>
<td>1.30</td>
<td>0.10</td>
<td>26</td>
</tr>
<tr>
<td>Avg</td>
<td></td>
<td>5.71</td>
<td></td>
<td></td>
<td></td>
<td>5.82</td>
<td>[1.24, 1.02]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Real arith</td>
<td>1</td>
<td>7.88</td>
<td>2.12</td>
<td>0.08</td>
<td>28</td>
<td>7.78</td>
<td>2.13</td>
<td>0.17</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>5.73*</td>
<td>2.10</td>
<td>0.05</td>
<td>30</td>
<td>7.35</td>
<td>2.17</td>
<td>0.08</td>
<td>31</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>8.66</td>
<td>2.16</td>
<td>0.07</td>
<td>26</td>
<td>6.73*</td>
<td>2.09</td>
<td>0.10</td>
<td>33</td>
</tr>
<tr>
<td>Avg</td>
<td></td>
<td>7.42</td>
<td></td>
<td></td>
<td></td>
<td>7.29</td>
<td>[1.17, 0.98]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Average Elapsed Ratios **

** Marks lowest elapsed time for the particular test
** Averages of all the test ratios
* Ratios are displayed as the [lowest time ratio, average time ratio]
Comparison Report

Test executed all at once with an extra load
(expressed in seconds)

DIGITAL VAX 11/780 4.3 BSD

<table>
<thead>
<tr>
<th>Type</th>
<th>Test Number</th>
<th>Elapsed Time</th>
<th>User Time</th>
<th>System Time</th>
<th>Percent Usage</th>
<th>Test Number</th>
<th>Elapsed Time</th>
<th>Elapsed Ratio*</th>
<th>User Time</th>
<th>System Time</th>
<th>Percent Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>composite</td>
<td>1</td>
<td>10.04</td>
<td>5.11</td>
<td>0.30</td>
<td>54</td>
<td>1</td>
<td>15.11</td>
<td>10.11*</td>
<td>0.42</td>
<td>0.47</td>
<td>59</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>10.08</td>
<td>5.09</td>
<td>0.28</td>
<td>53</td>
<td>2</td>
<td>10.11</td>
<td>10.11*</td>
<td>0.44</td>
<td>0.47</td>
<td>58</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>10.04*</td>
<td>5.13</td>
<td>0.23</td>
<td>53</td>
<td>3</td>
<td>10.11*</td>
<td></td>
<td>0.51</td>
<td>0.51</td>
<td>57</td>
</tr>
<tr>
<td>avg</td>
<td></td>
<td>10.05</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>11.78</td>
<td>[1.01, 1.17]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>uptime</td>
<td>1</td>
<td>0.71*</td>
<td>0.02</td>
<td>0.09</td>
<td>15</td>
<td>1</td>
<td>1.03</td>
<td>0.80*</td>
<td>0.08</td>
<td>0.05</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0.78</td>
<td>0.01</td>
<td>0.11</td>
<td>18</td>
<td>2</td>
<td>1.03</td>
<td>0.04</td>
<td>0.10</td>
<td>0.10</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.76</td>
<td>0.07</td>
<td>0.08</td>
<td>17</td>
<td>3</td>
<td>1.01</td>
<td>0.07</td>
<td>0.08</td>
<td>0.08</td>
<td>13</td>
</tr>
<tr>
<td>avg</td>
<td></td>
<td>0.74</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.95</td>
<td>[1.13, 1.27]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>whetstone</td>
<td>1</td>
<td>7.45</td>
<td>1.70</td>
<td>0.05</td>
<td>23</td>
<td>1</td>
<td>7.92</td>
<td>7.43*</td>
<td>1.76</td>
<td>0.10</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>6.49*</td>
<td>1.69</td>
<td>0.03</td>
<td>27</td>
<td>2</td>
<td>7.92</td>
<td>7.43*</td>
<td>1.80</td>
<td>0.07</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>6.70</td>
<td>1.75</td>
<td>0.01</td>
<td>26</td>
<td>3</td>
<td>7.92</td>
<td>7.43*</td>
<td>1.76</td>
<td>0.10</td>
<td>25</td>
</tr>
<tr>
<td>avg</td>
<td></td>
<td>6.88</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>8.39</td>
<td>[1.14, 1.22]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>integer arith</td>
<td>1</td>
<td>5.49</td>
<td>1.35</td>
<td>0.03</td>
<td>26</td>
<td>1</td>
<td>7.02</td>
<td>6.43*</td>
<td>1.76</td>
<td>0.11</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>4.85*</td>
<td>1.27</td>
<td>0.03</td>
<td>27</td>
<td>2</td>
<td>6.43</td>
<td>6.43*</td>
<td>1.36</td>
<td>0.12</td>
<td>23</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>5.03</td>
<td>1.30</td>
<td>0.04</td>
<td>27</td>
<td>3</td>
<td>6.43</td>
<td>6.43*</td>
<td>1.34</td>
<td>0.10</td>
<td>26</td>
</tr>
<tr>
<td>avg</td>
<td></td>
<td>5.12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6.60</td>
<td>[1.12, 1.29]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>real arith</td>
<td>1</td>
<td>9.09</td>
<td>2.04</td>
<td>0.07</td>
<td>23</td>
<td>1</td>
<td>10.04</td>
<td>8.52*</td>
<td>2.11</td>
<td>0.14</td>
<td>22</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>8.40</td>
<td>2.12</td>
<td>0.06</td>
<td>26</td>
<td>2</td>
<td>8.52</td>
<td>8.52*</td>
<td>2.16</td>
<td>0.12</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>8.21*</td>
<td>2.01</td>
<td>0.06</td>
<td>25</td>
<td>3</td>
<td>8.52</td>
<td>8.52*</td>
<td>2.04</td>
<td>0.17</td>
<td>26</td>
</tr>
<tr>
<td>avg</td>
<td></td>
<td>8.57</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>9.26</td>
<td>[1.04, 1.08]</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* Marks lowest elapsed time for the particular test
** Averages of all the test ratios
+ Ratios are displayed as the [lowest time ratio, average time ratio]
Summary Report

Elapsed time averages normalized to the baseline

<table>
<thead>
<tr>
<th></th>
<th>Digital VAX 11/780 4.3 BSD</th>
<th>Digital VAX 11/730 4.3 BSD</th>
<th>DEC VAX 8600 4.3 BSD</th>
<th>DEC VAX 11/750 4.1 BSD</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Composite</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>one at a time</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>all at once, no load</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>all at once, w/ load</td>
<td>1.0</td>
<td>1.2</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td><strong>Uptime</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>one at a time</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>all at once, no load</td>
<td>1.0</td>
<td>1.6</td>
<td>0.7</td>
<td>3.2</td>
</tr>
<tr>
<td>all at once, w/ load</td>
<td>1.0</td>
<td>1.3</td>
<td>1.5</td>
<td>3.4</td>
</tr>
<tr>
<td><strong>Wetstone</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>one at a time</td>
<td>1.0</td>
<td>1.5</td>
<td>1.1</td>
<td>1.4</td>
</tr>
<tr>
<td>all at once, no load</td>
<td>1.0</td>
<td>1.0</td>
<td>0.8</td>
<td>1.8</td>
</tr>
<tr>
<td>all at once, w/ load</td>
<td>1.0</td>
<td>1.2</td>
<td>1.2</td>
<td>1.3</td>
</tr>
<tr>
<td><strong>Integer Arith</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>one at a time</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.5</td>
</tr>
<tr>
<td>all at once, no load</td>
<td>1.0</td>
<td>1.0</td>
<td>0.8</td>
<td>1.6</td>
</tr>
<tr>
<td>all at once, w/ load</td>
<td>1.0</td>
<td>1.3</td>
<td>1.3</td>
<td>1.5</td>
</tr>
<tr>
<td><strong>Real Arith</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>one at a time</td>
<td>1.0</td>
<td>1.4</td>
<td>1.3</td>
<td>1.6</td>
</tr>
<tr>
<td>all at once, no load</td>
<td>1.0</td>
<td>1.0</td>
<td>0.8</td>
<td>1.8</td>
</tr>
<tr>
<td>all at once, w/ load</td>
<td>1.0</td>
<td>1.1</td>
<td>1.0</td>
<td>1.2</td>
</tr>
</tbody>
</table>
NAME

bench - benchmark driver and result comparison generator

SYNOPSIS

bench -b [-v] [-s] [-l] [-n "make, model, version"]
bench -p [-s] [-f] [-l] [-o outfile]
bench -e [-s] [-f] [-l] [-o outfile]

DESCRIPTION

Bench is a benchmark driver program to time the execution of a suite of tests specified
in the ASCII file benchlist. The defaults to bench are designed to allow a user with
little or no understanding of the options to establish a baseline system and create
comparisons of other systems with the baseline.

The user can override the defaults by using the options. For example: data can be
collected without generating a comparison report; an output filename can be specified
for the comparison report; a summary table of all the system results can be generated;
and selected groups of tests can be executed without running the complete suite.

The available flags are:

-b Execute the test suite and add the results to the database as the baseline from
which comparisons will be produced.

-a Execute the test suite and add the results to the database. Unless used with the -o
option, no comparison report will be generated.

-p Prepare a comparison report between two systems of the user's choice. May be
used with the -o option; default is to the line printer.

-e Generate a summary table containing normalized elapsed times for all systems in
the database. If no system has been assigned as the baseline, the user will be
prompted for a system to use as a baseline. Default is to the line printer.

-v Verbose: causes output to be generated to standard output. This information is
helpful when trying to follow the progress of the driver. Default is off.

-n "make, model, version"
Use the make, model, and version of the system to identify the results. This
option is useful when executing the driver in a batch mode. If this is not specified
on the command line, the user will be prompted for make, model, and version.

-o outfile
Name the formatted output file outfile. By default the output file is created by
adding the last three digits of the process id to /tmp/bench.
The presence of any of the -s, -f, and -l flags cause the execution to be limited to only what is specified. (If -s, -f, or -l are not specified, the default sets all three flags.)

- s  This flag causes tests to be executed one after the other with no extra load added to the system.

- f  This flag causes simultaneous execution of the tests with no extra load added to the system.

- l  This flag causes simultaneous execution of the tests with extra load added to the system at the same time.

EXAMPLES

Execute the test suite on the current machine and store the results as the baseline.

% bench -b

Execute the test suite on the current machine and store the results. Also input the make, model, and version from the command line.

% bench -a -n "DEC, VAX 8600, 4.3 BSD"

Print a summary of current database into outfile.

% bench -e -o outfile

FILES

<table>
<thead>
<tr>
<th>Path</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/tmp/benchXXX</td>
<td>formatted output of the comparison</td>
</tr>
<tr>
<td>baseline</td>
<td>result data of baseline system</td>
</tr>
<tr>
<td>benchlist</td>
<td>the path of test and the printable name</td>
</tr>
<tr>
<td>targetXXX</td>
<td>result data of system to be compared to baseline</td>
</tr>
</tbody>
</table>

BUGS
Instructions for Data Collection Using the Bench Program

Before bench can be used on any system, instructions 1 through 4 must be completed. All tar instructions are assuming 1600 bpi on drive 0.

1. Mount tape on drive 0 at 1600 bpi.
2. Change to a working directory with at least 3,000 blocks free.
3. To unload the tape, type:
   % tar xv
4. To compile the driver program and test suite, type:
   % make
   It may be necessary to edit the bench.mk file to alter the names for the different compilers with optimizers on.

After completing steps 1-4 above, any of the remaining sections can be followed to collect data or display previously collected data.

To establish a baseline and store the results on the tape:
1. Type:
   % bench -b
   a. Enter the make, model, and version of the system when prompted by the program.
   b. Sit back and relax.
2. To store the baseline file on the tape, type:
   % tar u baseline
3. Remove all working files and directories from the disk if desired.

To add a target system to the database:
1. Type:
   % bench -a
   a. Enter the system description when prompted by the program.
   b. Sit back and relax.
2. To store target system results on tape, type:
   % tar u target*
   If a print-out is desired, skip to one of the last two sections.
3. Remove all working files and directories if desired.
To print a comparison report between two systems in the database:
1. Type:
   
   \% bench -p
   
   A list of systems in the database will appear preceded by a number. The system will prompt you for two numbers to indicate the two systems to be compared. The output will be sent to the line printer.
2. Remove all working files and directories if desired.

To print a summary report of all systems in the database:
1. Type:
   
   \% bench -e
   
   The output will be sent to the line printer.
2. Remove all working files and directories if desired.
Prime Numbers

C prime.f
C C PROGRAM TO GENERATE PRIME NUMBERS
C Compile by: fort -0 prime.f -o prime
C
C PROGRAM PRIME
C COMMON/DAT/VALUE(8192)
C
C INITIALIZE DATA STRUCTURES
C ILUM=6
IPRT=0
ICNT=512
CUR=2.
TOP=3.
I=1

C CHECK REMAINDER
C 5 IF(AMOD(TOP,CUR).EQ.0.)GO TO 10
CUR=CUR+1.
IF(CUR.LT.TOP)GO TO 5

C IF WE SCAN FROM 2 THRU TOP, THEN TOP IS A PRIME NUMBER
C VALUE(I)=TOP
I=I+1

C SET UP FOR NEXT PRIME NUMBER
C 10 TOP=TOP+2.
CUR=2.
IF(I.LE.ICNT)GO TO 5

C PRINT THE PRIME NUMBERS WEVE GENERATED
C IF(IPRT.EQ.0)STOP
DO 15 I=1,ICNT
WRITE(ILUM,9010)(VALUE(J),J=I,I+7)
9010 FORMAT(6F10.0)
15 CONTINUE
STOP
END
Calling Sequence and Arguments Passing

C calseq.f
C
C PROGRAM TO TEST CALLING SEQUENCE AND ARGUMENT PASSING
C
C Compile by:    fort -O calseq -o calseq
C
C
PROGRAM CALSEQ
C
Z=0.
10 I=0
    J=1
    K=2
    L=3
    A=0.
    B=1.
    C=2.
    D=3.
    CALL CALSEQ1(A,I,B,J,C,K,D,L)
    Z=Z+1.
    IF(Z.LT.1.E6)GO TO 10
    STOP
END
C
SUBROUTINE CALSEQ1(A,I,B,J,C,K,D,L)
A=D
B=C
I=J
K=L
RETURN
END
Random Numbers

PROGRAM TO PERFORM A CHECK OF THE RANDOM NUMBER GENERATOR BY PERFORMING DIRECT ACCESS TO A DISK FILE. THE SUBROUTINE WILL USE A RANDOM NUMBER FROM 1 TO 256 AS THE KEY TO READ A RECORD, INCREMENT THE VALUE READ, AND WRITE THE NEW VALUE.

Compile by: fort -O rndisk.f -o rndsk

PROGRAM RNDISK

ILUN=6
IPRT=0
ICNT=128
FCNT=FLOAT(ICNT)
FCHK=FCNT*100.
IRAN=0
isize = 4
B=rand(IRAN)
OPEN(ACCESS='DIRECT',
1FILE='TEST',
2FORM='UNFORMATTED',
3MAXREC=ICNT+1,
4RECL=isize*2,
5STATUS='UNKNOWN',
6UNIT=4)

CREATE FILE WITH EACH RECORD CONTAINING ALL ZEROS

DO 10 I=1,ICNT
IERC=I
WRITE(4,rec=IERC)FLOAT(IERC),0.
10 ::CONTINUE

GENERATE ICNT=100 RANDOM NUMBERS

A=0.
20 IREC=IFIX(FCNT*rand(IRAN))+1
IF(IREC.GE.1.AND.IREC.LE.ICNT)GO TO 25
WRITE(ILUN,9010)IERC
9010 FORMAT( ' RANDOM NUMBER OUT OF RANGE',I6)
25 I=IERC
READ(4,rec=I)RNUM,COUNT
COUNT=COUNT+1.
I=IERC
WRITE(4,rec=I)RNUM,COUNT
A=A+1.
IF(A.LT.FCHK)GO TO 20

READ FILE, GET MIN, MAX AND AVERAGE OF RANDOM NUMBER GENERATOR

AMIN=9999.
AMAX=0.
Random Numbers

AVE=0.
DO 30 I=1, ICNT
IREC=I
READ(4,rec=IREC) RNUM, COUNT
IF(COUNT.GT.AMAX) AMAX=COUNT
IF(COUNT.LT.AMIN) AMIN=COUNT
AVE=AVE+COUNT
30 CONTINUE
CLOSE(UNIT=4)
AVE=AVE/ICNT
IF(IPRT.EQ.0) STOP
WRITE(ILUM,9000) AMIN, AMAX, AVE
9000 FORMAT(3F15.0)
STOP
END
Fast Fourier Transform

C fft.f
C
C PROGRAM TO PERFORM A FAST FOURIER TRANSFORM USING THE
C DECIMATION-IN-TIME METHOD.
C
C Compile by: fort -O fft.f -o fft
C
PROGRAM FFT
C
COMMON/DAT/A(4096)
COMPLEX A,U,W,T
C
INITIALIZE
C
ILUM=6
IPRT=0
DO 25 LOOP=1,10
M=12
ICNT=2**M
PI=3.141592653589793
DO 1 I=1,ICNT
B=SIN(2.*PI*FLOAT(I)/PER)
A(I)=CMPLX(B,0.)
1 CONTINUE
NWV2=N/2
NM1=N-1
J=1
DO 7 I=1,NM1
IF(I.GE.J)GO TO 5
T=A(J)
A(J)=A(I)
A(I)=T
5 T=A(J)
K=NW2
IF(K.GE.J)GO TO 7
J=J-K
K=K/2
GO TO 6
7 J=J+K
PI=3.141592653589793
DO 20 L=1,M
LE=2**L
LE1=LE/2
U=(1.,0.)
W=CMPLX(COS(PI/FLOAT(LE1)),SIM(PI/FLOAT(LE1)))
DO 20 J=1,LE1
DO 10 I=J,N,LE
IP=I+LE1
T=A(IP)*U
A(IP)=A(I)-T
A(I)=A(I)+T
10 CONTINUE
20 U=W
25 CONTINUE
IF(IPRT.EQ.0)STOP
Fast Fourier Transform

DO 30 I=1,128,4
WRITE(ILUN,9000)(A(J),J=I,I+3)
9000 FORMAT(4G15.6)
30 CONTINUE
STOP
END
Matrix Inversion

matrix.f

MATRIX INVERSION USING GAUSS-JORDAN REDUCTION
INVERTED MATRIX OVERLAYS ORIGINAL MATRIX IN MEMORY
PARITAL PIVOTING IS NOT USED

Compile by; fort -O matrix -o matrix

PROGRAM MATRIX

COMMON/DAT/A(15,15) DOUBLE PRECISION A
ILUM=6
IPRT=0
DO 10 LOOP=1,10000
N=4
A(1,1)=1.
A(1,2)=1.
A(1,3)=1.
A(1,4)=1.
A(2,1)=4.
A(2,2)=5.
A(2,3)=6.
A(2,4)=7.
A(3,1)=6.
A(3,2)=10.
A(3,3)=15.
A(3,4)=21.
A(4,1)=12.
A(4,2)=30.
A(4,3)=60.
A(4,4)=105.

CALCULATE ELEMENTS OF REDUCED MATRIX

DO 6 K=1,N

CALCULATE NEW ELEMENTS OF PIVOT ROW

DO 4 J=1,N
IF(J.EQ.K)GO TO 4
A(K,J)=A(K,J)/A(K,K)
CONTINUE

CALCULATE ELEMENT REPLACING PIVOT ELEMENT

A(K,K)=1./A(K,K)

CALCULATE NEW ELEMENTS NOT IN PIVOT ROW OR PIVOT COLUMN

DO 5 I=1,N
IF(I.EQ.K)GO TO 5
DO 5 J=1,N
IF(J.EQ.K)GO TO 5
Matrix Inversion

\[ A(I,J) = A(I,J) - A(K,J) \times A(I,K) \]

5 CONTINUE

C

C CALCULATE REPLACEMENT ELEMENTS FOR PIVOT COLUMN-EXCEPT PIVOT ELEMENT
C

DO 6 I = 1, N
IF(I .EQ. K) GO TO 6
A(I,K) = -A(I,K) \times A(K,K)
6 CONTINUE
10 CONTINUE

C

C OUTPUT INVERTED MATRIX
C

IF(IPRT .EQ. 0) STOP
WRITE(ILUN,8)((A(I,J), J = 1, N), I = 1, N)
8 FORMAT(4F16.4)
STOP
END
Polynomial Roots

C
C roots.f
C
C ROOTS OF POLYNOMIAL BY BAIRSTOWS METHOD
C
C Compile by: fort -O roots.f -o roots
C
C
PROGRAM ROOTS
C
DIMENSION A(30),B(30),C(30)
ILUN=6
IPRT=0
IF(IPRT.EQ.0)GO TO 200
IPRT=0
JPRT=1
200 DO 100 LOOP=1,10000
IF(LOOP.NE.10000)GO TO 220
IF(IPRT.EQ.0)GO TO 220
IPRT=1
220 U=0.
V=0.
EPSI=1.E-6
N=5
A(1)=-3.
A(2)=-10.
A(3)=10.
A(4)=44.
A(5)=48.
C
C SEE IF M=0,1, OR GREATER THAN 1
C
40 IF(N-1).LE.0,5,7
5 P=A(1)
Q=0.
IT=1
IF(IPRT.NE.0)WRITE(ILUN,6)N,P,Q,IT
6 FORMAT('X('',12,'') = ',',2X,F8.4,6X,F8.4,10X,I3)
GO TO 100
C
C SEE IF M=2 OR IF M IS GREATER THEN 2
C
7 IF(M.EQ.2)GO TO 8
GO TO 13
8 U=A(1)
V=A(2)
IT=1
9 P=-U/2.
RAD=U**2-4.*V
C
C CHECK THE SIGN OF U**2-4.*V
C
IF(RAD.GT.0.)GO TO 12
RAD=-RAD
Q=SQRT(RAD)/2.
IF(IPRT.NE.0)WRITE(ILUN,6)N,P,Q,IT
Polynomial Roots

N=N-1
Q=Q
90 IF(IPRT.NE.0)WRITE(ILUN,6)N,P,Q,IT
10 N=N-1
C CHECK TO SEE IF N IS GREATER THEN ZERO
C
IF(N.LE.0)GO TO 100
DO 11 I=1,N
11 A(I)=B(I)
GO TO 40
12 Q=SQRT(RAD)/2.
W=P
Z=Q
P=P+Q
Q=0.
IF(IPRT.NE.0)WRITE(ILUN,6)N,P,Q,IT
N=N-1
P=W-2
GO TO 90
13 U=UI
V=VI
IT=1

C CALCULATE THE B VALUES
C
50 B(1)=A(1)-U
B(2)=A(2)-B(1)*U-V
DO 14 K=3,N
14 B(K)=A(K)-B(K-1)*U-B(K-2)*V
C CALCULATE THE C VALUES
C
C(1)=B(1)-U
C(2)=B(2)-C(1)*U-V
M=M-1
DO 15 K=3,N
15 C(K)=B(K)-C(K-1)*U-C(K-2)*V
C CALCULATE DELU AND DELV
C
IF(N.GT.3)GO TO 17
DENOM=C(N-1)-C(N-2)**2
IF(DENOM.EQ.0.)GO TO 30
DELU=(B(N)-B(N-1)*C(N-2))/DENOM
16 DELV=(C(N-1)*B(N-1)-C(N-2)*B(N))/DENOM
GO TO 18
17 DENOM=C(N-1)*C(N-3)-C(N-2)**2
IF(DENOM.EQ.0.)GO TO 30
DELU=(B(N)*C(N-3)-B(N-1)*C(N-2))/DENOM
GO TO 16
C CALCULATE NEW U AND V VALUES
C
Polynomial Roots

18  U=U+DELU
    V=V+DELV
    SUM=ABS(DELU)+ABS(DELV)
C
C STORE THE FIRST SUM CALCULATED
C
    IF(IT.EQ.1)GO TO 19
    GO TO 20
19  STORE=SUM
    GO TO 21
20  IF(IT.EQ.50)GO TO 28
    IF(IT.GE.200)GO TO 26
21  IF(SUM.LE.EPSI)GO TO 9
    IF(IT.EQ.100)GO TO 23
    IT=IT+1
    GO TO 50
22  IF(IPRT.NE.0)WRITE(ILUN,24)
23  FORMAT(' CONVERGENCE IS SLOW')
    IF(IPRT.NE.0)WRITE(ILUN,25)U,V
24  FORMAT(' U=',E14.7,' V=',E14.7)
    GO TO 22
25  IF(IPRT.NE.0)WRITE(ILUN,27)
26  FORMAT(' STOPPED AFTER 200 ITERATIONS')
    IF(IPRT.NE.0)WRITE(ILUN,25)U,V
    GO TO 100
C
C SEE IF SUM AFTER 50 ITERATIONS EXCEEDS FIRST SUM STORED
C
28  IF(SUM.LT.STORE)GO TO 21
    IF(IPRT.NE.0)WRITE(ILUN,29)
29  FORMAT(' DIVERGENCE OCCURRING')
    IF(IPRT.NE.0)WRITE(ILUN,25)U,V
    GO TO 100
30  IF(IPRT.NE.0)WRITE(ILUN,31)
31  FORMAT(' DENOMINATOR IS ZERO')
    GO TO 100
100  CONTINUE
    STOP
END
C Sieve

/*
 * sieve.c
 * Eratosthenes Sieve Prime Number Program in C */
 * Compile by: cc -O sieve.c -o csieve
 */

#define true 1
#define false 0
#define size 8190

char flags[size + 1];

main() {
    int i, prime, k, count, iter;

    printf("100 iterations\n");
    for(iter = 1; iter <= 100; iter++) {
        count = 0;
        for(i = 0; i <= size; i++)
            flags[i] = true;
        for(i = 0; i <= size; i++) {
            if(flags[i]) {
                prime = i + i + 3;
                for(k = i + prime; k <= size; k += prime)
                    flags[k] = false;
                count++;
            }
        }
    }
    printf("%d is largest of %d primes.\n", prime, count);
}
Fortran Sieve

c c sieve.f
c c eratosthenes sieve with Knuth's optimization
c c Compile by: fort -O sieve.f -o fsieve

c integer i,j,k,iter,prime,count
logical flags(8191),last

write(6,10)
10 format (' 100 iterations')
do 20 iter = 1, 100
    count = 0
    do 30 i = 1, 8191
        flags(i) = .true.
        last = .false.
        do 40 i = 1, 8191
            if (.not. flags(i)) go to 50
            prime = i + i + 1
            count = count + 1
            write(6,11) prime
11 format (lx,i6)
            if (last) go to 50
            k = (prime*prime - 1) / 2
            k = i + prime
            do 60 j = k, 8191, prime
                flags(j) = .false.
                if (prime .ge. 127) last = .true.
30  continue
40  continue
20  continue

write(6,12) count
12 format (lx, i6, ' primes')
end
Pascal Sieve

(* sieve.p *)

(* Eratosthenes Sieve Prime Number Program in Pascal *)

(* Compile by: pi sieve.p *)

program prime(output);

const
  size = 8190;

var
  flags : array [0..size] of boolean;
  i,prime,k,cnt,iter : integer;

begin
  writeln('100 iterations');
  for iter := 1 to 100 do begin
    cnt := 0;
    for i := 0 to size do flags[i] := true;
    for i := 0 to size do
      if flags[i] then begin
        prime := i+i+3;
        k := i + prime;
        while k <= size do begin
          flags[k] := false;
          k := k + prime
        end;
        cnt := cnt + 1
      end;
  end;
  writeln(cnt,' primes')
end.

Whetstone

/*
  Whetstone benchmark in C. This program is a translation of the
  original Algol version in "A Synthetic Benchmark" by H.J. Curnow
  Used to test compiler optimization and floating point performance.

  Compile by:
    cc -O -s -o whet whet.c
  or:
    cc -O -DPOUT -s -o whet whet.c
  if output is desired.
*/

#define ITERATIONS 10 /* 1 Million Whetstone instructions */

#include "math.h"

double x1, x2, x3, x4, x, y, z, t, t1, t2;
double e1[4];
int i, j, k, l, n1, n2, n3, n4, n6, n7, n8, n9, n10, n11;

main()

  /* initialize constants */
  t = 0.499975;
  t1 = 0.50025;
  t2 = 2.0;

  /* set values of module weights */
  n1 = 0 * ITERATIONS;
  n2 = 12 * ITERATIONS;
  n3 = 14 * ITERATIONS;
  n4 = 345 * ITERATIONS;
  n6 = 210 * ITERATIONS;
  n7 = 32 * ITERATIONS;
  n8 = 899 * ITERATIONS;
  n9 = 616 * ITERATIONS;
  n10 = 0 * ITERATIONS;
  n11 = 93 * ITERATIONS;

  /* MODULE 1: simple identifiers */
  x1 = 1.0;
  x2 = x3 = x4 = -1.0;

  for(i = 1; i <= n1; i += 1) /*
    x1 = ( x1 + x2 + x3 - x4 ) * t;
    x2 = ( x1 + x2 - x3 - x4 ) * t;
    x3 = ( x1 - x2 + x3 + x4 ) * t;
    x4 = (-x1 + x2 + x3 + x4 ) * t;*/
/* MODULE 2: array elements */

#define POUT

pout(n1, n1, n1, x1, x2, x3, x4);
#endif

pout(n1, n1, n1, x1, x2, x3, x4);
#endif

/* MODULE 3: array as parameter */

#define POUT

pout(n2, n2, n2, el[0], el[1], el[2], el[3]);
#endif

/* MODULE 4: conditional jumps */

#define POUT

pout(n2, n3, n2, el[0], el[1], el[2], el[3]);
#endif

/* MODULE 5: omitted */

/* MODULE 6: integer arithmetic */
j = 1;
k = 2;
l = 3;

for (i = 1; i <= n6; i += 1) {
    j = j * (k - j) * (l - k);
    k = l * k - (l - j) * k;
    l = (l - k) * (k + j);
    e1[j - 2] = j + k + l;
    e1[k - 2] = j * k * l;
} /* C arrays are zero based */

#define POUT
pout(n6, j, k, e1[0], e1[1], e1[2], e1[3]);
#endif

/* MODULE 7: trig. functions */

x = y = 0.5;

for (i = 1; i <= n7; i += 1)
    x = t * atan(t2*t2*sin(x)*cos(x)/(cos(x+y)+cos(x-y)-1.0));
    y = t * atan(t2*t2*sin(y)*cos(y)/(cos(x+y)+cos(x-y)-1.0));

#define POUT
pout(n7, j, k, x, x, y, y);
#endif

/* MODULE 8: procedure calls */

x = y = z = 1.0;

for (i = 1; i <= n8; i += 1)
p3(x, y, &z);

#define POUT
pout(n8, j, k, x, y, z, z);
#endif

/* MODULE 9: array references */

j = 1;
k = 2;
l = 3;

el[0] = 1.0;
el[1] = 2.0;
el[2] = 3.0;

for (i = 1; i <= n9; i += 1)
p0();

#define POUT
pout(n9, j, k, el[0], el[1], el[2], el[3]);
#endif
/* MODULE10: integer arithmetic */

j = 2;
k = 3;

for(i = 1; i <= n10; i += 1) {
    j = j + k;
k = j + k;
    j = k - j;
k = k - j - j;
}

#ifdef POUT
    pout(n10, j, k, x1, x2, x3, x4);
#endif

/* MODULE11: standard functions */

x = 0.75;
for(i = 1; i <= n11; i += 1)
x = sqrt( exp( log(x) / t1));

#ifdef POUT
    pout(n11, j, k, x, x, x);
#endif

exit(0);

ps(e)
double e[4];
{
    register int j;
    j = 0;
lab:
j += 1;
    if (j < 6)
goto lab;
}

p3(z, y, z)
double x, y, *z;
{
x = t * (x + y);
y = t * (x + y);
*z = (x + y) / t2;
}
Whetstone

p0()
{
    e1[j] = e1[k];
    e1[k] = e1[l];
    e1[l] = e1[j];
}
#endif

#define POUT
pout(n, j, k, x1, x2, x3, x4)
int n, j, k;
double x1, x2, x3, x4;
{
    printf("%6d%6d%6d %5e %5e %5e\n",
            n, j, k, x1, x2, x3, x4);
}
#endif
Dhrystone

/*
 * dry.c
 * "DHRYSTONE" Benchmark Program
 * Version: C/1.1, 12/01/84
 * Date: PROGRAM updated 01/06/86, RESULTS updated 02/17/86
 * Author: Reinhold P. Weicker, CACM Vol 27, No 10, 10/84 pg. 1013
 * Translated from ADA by Rick Richardson
 * Every method to preserve ADA-likeness has been used,
 * at the expense of C-ness.
 * Compile: cc -O dry.c -o drynr : No registers
 * cc -O -DREG=register dry.c -o dryr : Registers
 * Run: drynr; dryr
 *
 * The following program contains statements of a high-level programming
 * language (C) in a distribution considered representative:
 * assignments 53%
 * control statements 32%
 * procedure, function calls 15%
 * 100 statements are dynamically executed. The program is balanced with
 * respect to the three aspects:
 * - statement type
 * - operand type (for simple data types)
 * - operand access
 * operand global, local, parameter, or constant.
 * The combination of these three aspects is balanced only approximately.
 * The program does not compute anything meaningful, but it is
 * syntactically and semantically correct.
 */
Dhrystone

/* Accuracy of timings and human fatigue controlled by next two lines */
#define LOOPS 500000 /* Use this for slow or 16 bit machines */
#define LOOPS 5000000 /* Use this for faster machines */

/* Compiler dependent options */
#undef NOENUM /* Define if compiler has no enum's */
#undef NOSTRUCTASSIGN /* Define if compiler can't assign structures */

/* define only one of the next two defines */
#define TIMES /* Use times(2) time function */
/*#define TIME /* Use time(2) time function */

/* define the granularity of your times(2) function (when used) */
#define HZ 60 /* times(2) returns 1/60 second (most) */
/*#define HZ 100 /* times(2) returns 1/100 second (WECO) */

/* for compatibility with goofed up version */
/*#define GOOF /* Define if you want the goofed up version */
#endif GOOF
char Version[] = "1.0";
#else
cchar Version[] = "1.1";
#endif

#endif NOSTRUCTASSIGN
#define structassign(d, s) memcpy(&(d), &(s), sizeof(d))
#else
#define structassign(d, s) d = s
#endif

#undef NOENUM
#define Ident1 1
#define Ident2 2
#define Ident3 3
#define Ident4 4
#define Ident5 5
typedef int Enumeration;
#else
typedef enum {Ident1, Ident2, Ident3, Ident4, Ident5} Enumeration;
#endif
typedef int OneToThirty;
typedef int OneToFifty;
typedef char CapitalLetter;
typedef char String30[31];
typedef int Array1Dim[51];
typedef int Array2Dim[51][51];

typedef struct Record
{
    struct Record *PtrComp;
    Enumeration Discr;
    Enumeration EnumComp;
} Record;
Dhrystone

OneToFifty
String30

};

typedef struct Record RecordType;
typedef RecordType * RecordPtr;
typedef int boolean;

#define NULL 0
#define TRUE 1
#define FALSE 0

ifndef REG
#define REG
endif

extern Enumeration Func1();
extern boolean Func2();

ifndef TIMES
#include <sys/types.h>
#include <sys/times.h>
endif

main()
{
    Proc0();
    exit(0);
}

/*
 * Package 1
 */

int
boolean
char
char
Array1Dim
Array2Dim
RecordPtr
RecordPtr

Proc0()
{
    OneToFifty
    REG OneToFifty
    REG OneToFifty
    REG char
    REG char
    Enumeration
    String30
    String30
    extern char
    IntLoc1;
    IntLoc2;
    IntLoc3;
    CharLoc;
    CharIndex;
    EnumLoc;
    String1Loc;
    String2Loc;
    *malloc();
Dhrystone

#ifndef TIME
long
long
long
long
register unsigned int
starttime = time((long *) 0);
for (i = 0; i < LOOPS; ++i);
nulitime = time((long *) 0) - starttime; /* Computes overhead of loop */
#endif

#ifndef TIMES

ifdef TIMES

time_t
for (i = 0; i < LOOPS; ++i);
times(&tms);
nulitime = tms.tms_utime - starttime; /* Computes overhead of looping */
#endif

PtrGlbNext = (RecordPtr) malloc(sizeof(RecordType));
PtrGlb = (RecordPtr) malloc(sizeof(RecordType));
PtrGlb->PtrComp = PtrGlbNext;
PtrGlb->Discr = Ident1;
PtrGlb->EnumComp = Ident3;
PtrGlb->IntComp = 40;
strcpy(PtrGlb->StringComp, "DHRYSTONE PROGRAM, SOME STRING");
#ifndef GOOF
strcpy(String1Loc, "DHRYSTONE PROGRAM, 1'ST STRING");  /* GOOF */
#endif
Array2Glob[8][7] = 10; /* Was missing in published program */

/**********************************************************************
-- Start Timer --
**********************************************************************/

#ifndef TIME

ifdef TIME

ifdef TIMES

times(&tms);
#endif
for (i = 0; i < LOOPS; ++i)
{

Proc5();
Proc4();
IntLoc1 = 2;
IntLoc2 = 3;
strcpy(String2Loc, "DHRYSTONE PROGRAM, 2'ND STRING");
EnumLoc = Ident2;

}
Dhrystone

BoolGlob = ! Func2(String1Loc, String2Loc);
while (IntLoc1 < IntLoc2)
{
    IntLoc3 = 5 * IntLoc1 - IntLoc2;
    Proc7(IntLoc1, IntLoc2, &IntLoc3);
    ++IntLoc1;
}
Proc8(Array1Glob, Array2Glob, IntLoc1, IntLoc3);
Proc1(PtrGlb);
for (CharIndex = 'A'; CharIndex <= Char2Glob; ++CharIndex)
    if (EnumLoc == Func1(CharIndex, 'C'))
        Proc6(Ident1, &EnumLoc);
IntLoc3 = IntLoc2 * IntLoc1;
IntLoc2 = IntLoc3 / IntLoc1;
IntLoc2 = 7 * (IntLoc3 - IntLoc2) - IntLoc1;
Proc2(&IntLoc1);

/*********************/
-- Stop Timer --
/*********************/

#define TIME
benchtime = time( (long *) 0) - starttime - nulltime;
printf("Dhrystone(\%s) time for \%ld passes = \%ld\n", 
    Version, 
    (long) LOOPS, benchtime);
printf("This machine benchmarks at \%ld dhrystones/second\n", 
    ((long) LOOPS) / benchtime);

#define TIMES
	times(&tms);
	benchtime = tms.tms_utime - starttime - nulltime;
printf("Dhrystone(\%s) time for \%ld passes = \%ld\n", 
    Version, 
    (long) LOOPS, benchtime/HZ);
printf("This machine benchmarks at \%ld dhrystones/second\n", 
    ((long) LOOPS) * HZ / benchtime);

PROC1(PtrParIn)
REG RecordPtr  PtrParIn;
{
#define NextRecord  (*(PtrParIn->PtrComp))
structassign(NextRecord, *PtrGlb);
PtrParIn->IntComp = 5;
NextRecord.IntComp = PtrParIn->IntComp;
NextRecord.PtrComp = PtrParIn->PtrComp;
Proc3(NextRecord.PtrComp);
if (NextRecord.Discr == Ident1)
{...}
Dhrystone

NextRecord.IntComp = 6;
Proc6(PtrParIn - EnumComp, &NextRecord.EnumComp);
NextRecord.PtrComp = PtrGlb - PtrComp;
Proc7(NextRecord.IntComp, 10, &NextRecord.IntComp);
}
else
struct assign(*PtrParIn, NextRecord);

} undefined NextRecord
}

Proc2(IntParIO)
OneToFifty *IntParIO;
{
REG OneToFifty IntLoc;
REG Enumeration EnumLoc;

IntLoc = *IntParIO + 10;
for(;;)
{
if (CharGlob == 'A')
{
    IntLoc;
    *IntParIO = IntLoc - IntGlob;
    EnumLoc = Ident1;
}
if (EnumLoc == Ident1)
    break;
}

Proc3(PtrParOut)
RecordPtr *PtrParOut;
{
    if (PtrGlb != NULL)
        *PtrParOut = PtrGlb - PtrComp;
    else
        IntGlob = 100;
    Proc7(10, IntGlob, &PtrGlb - IntComp);
}

Proc4()
{
REG boolean BoolLoc;

BoolLoc = CharGlob == 'A';
BoolLoc |= BoolGlob;
CharGlob = 'A';
}

Proc5()
{
CharGlob = 'A';
BoolGlob = FALSE;
extern boolean Func3();

Proc6(EnumParIn, EnumParOut)
REG Enumeration EnumParIn;
REG Enumeration *EnumParOut;
{
    *EnumParOut = EnumParIn;
    if ( ! Func3(EnumParIn) )
        *EnumParOut = Ident4;
    switch (EnumParIn)
    {
        case Ident1:  *EnumParOut = Ident1; break;
        case Ident2:  if (IntGlob > 100) *EnumParOut = Ident1;
                      else *EnumParOut = Ident4;
                      break;
        case Ident3:  *EnumParOut = Ident2; break;
        case Ident4:  break;
        case Ident5:  *EnumParOut = Ident3;
    }
}

Proc7(IntParI1, IntParI2, IntParOut)
OneToFifty IntParI1;
OneToFifty IntParI2;
OneToFifty *IntParOut;
{
    REG OneToFifty IntLoc;
    IntLoc = IntParI1 + 2;
    *IntParOut = IntParI2 + IntLoc;
}

Proc8(Array1Par, Array2Par, IntParI1, IntParI2)
Array1Dim Array1Par;
Array2Dim Array2Par;
OneToFifty IntParI1;
OneToFifty IntParI2;
{
    REG OneToFifty IntLoc;
    REG OneToFifty IntIndex;
    IntLoc = IntParI1 + 5;
    Array1Par[IntLoc] = IntParI2;
    Array1Par[IntLoc+1] = Array1Par[IntLoc];
    Array1Par[IntLoc+30] = IntLoc;
    for (IntIndex = IntLoc; IntIndex <= (IntLoc+1); ++IntIndex)
        Array2Par[IntLoc][IntIndex] = IntLoc;
    ++Array2Par[IntLoc][IntLoc-1];
    Array2Par[IntLoc+20][IntLoc] = Array1Par[IntLoc];
    IntGlob = 5;
}
Dhrystone

Enumeration Func1(CharPar1, CharPar2)
CapitalLetter CharPar1;
CapitalLetter CharPar2;
{
    REG CapitalLetter CharLoc1;
    REG CapitalLetter CharLoc2;

    CharLoc1 = CharPar1;
    CharLoc2 = CharLoc1;
    if (CharLoc2 != CharPar2)
        return (Ident1);
    else
        return (Ident2);
}

boolean Func2(StrParIl, StrParI2)
String30 StrParIl;
String30 StrParI2;
{
    REG OneToThirty IntLoc;
    REG CapitalLetter CharLoc;

    IntLoc = 1;
    while (IntLoc <= 1)
        if (Func1(StrParIl[IntLoc], StrParI2[IntLoc+1]) == Ident1)
            CharLoc = 'A';
            ++IntLoc;
        
    if (CharLoc >= 'W' && CharLoc <= 'Z')
        IntLoc = 7;
    if (CharLoc == 'X')
        return (TRUE);
    else
        if (strcmp(StrParIl, StrPanIZ) > 0)
            
            IntLoc += 7;
            return (TRUE);
        } else
            return (FALSE);
}

boolean Func3(EnumParIn)
REG Enumeration EnumParIn;
{
    REG Enumeration EnumLoc;

    EnumLoc = EnumParIn;
    if (EnumLoc == Ident3) return (TRUE);
    return (FALSE);
Dhrystone

#ifdef NOSTRUCTASSIGN
memcpy(d, s, l)
register char *d;
register char *s;
register int l;
{
    while (l--)*d++ = *s++;
}
#endif
Block Write

/*
 * blockwrite.c
 *
 * This program creates a very large file.
 *
 * Compile by: cc -o blockwrite.c -o blockwrite
 */

#define NAME "BLOCKWRITE"
#define FILENAME "bigfile"
#define BSIZE 8096 /* 8K block */
#define BLOCKS 128 /* number of 8K blocks in file (total 1 Mbyte) */

#include <sys/file.h>

int main()

int fileflags = O_CREAT | O_TRUNC | O_APPEND | O_WRONLY;
int filemode = 0777;
int f;
int lcount = 0;
char buffer[BSIZE];

printf("%s: beginning (%d bytes in file)\n", NAME, BSIZE * BLOCKS);
fflush(1);
for(lcount = 0; lcount < 23; lcount++) {
    if ((f = open(FILENAME, fileflags, filemode)) < 0) {
        printf("%s: unable to create '%s'\n", NAME, FILENAME);
        exit(1);
    }
    for (i=1;i<=BLOCKS;i++) write(f, buffer, BSIZE);
close(f);
}

printf("%s: complete (%d bytes in file)\n", NAME, BSIZE * BLOCKS);
fflush(1);
Block Read

/*
 * blockread.c
 * This program reads a very large file.
 * Compile by cc -O blockread.c -o blockread
 */
#define NAME "BLOCKREAD"
#define FNNAME "bigfile"
#define BSIZE 8096 /* 8k block */
#define BLOCKS 128 /* number of 8k blocks in file (total 1 Mbyte) */

#include <sys/file.h>

int main() {
    int fileflags = O_RDWR | O_LARGEFILE;
    int filemode = 0444;
    int f,
    char buffer[BSIZE];
    int i,
    int lcount;

    printf("%s: beginning (%d bytes in file))\n", NAME, BSIZE * BLOCKS);
    fflush(1);
    for(lcount = 0; lcount < 39; lcount++)
        if ((f = open(FNNAME, fileflags, filemode)) <= 0)
            printf("%s: unable to open '%s'\n", NAME, FNNAME);
            exit(1);

    for (i = 1; i <= BLOCKS; i++) read(f, buffer, BSIZE);
    close(f);

    printf("%s: complete (%d bytes in file))\n", NAME, BSIZE * BLOCKS);
    fflush(1);
}

46
csh script to run timing on sort

sorttest
sortfile - input to sort
sortstandard - presorted output for checking

echo Start of sort
sort -4 +5 ./preaward/sortfile > sortout
echo End of sort. Start of compare.
diff ./preaward/sortstandard sortout
echo End of compare.
rm -f sortout
.de sh
.br
.ne 5
.PP

**SYNOPSIS**

.csh
[ .B -cefe instvVxx ]
[ .B . . . ]

**DESCRIPTION**

.csh
is a first implementation of a command language interpreter incorporating a history mechanism (see
.8 "History Substitutions"]
job control facilities (see
.8 Jobs)
and a C-like syntax.
So as to be able to use its job control facilities, users of
.csh
must (and automatically) use the new tty driver fully described in
.1R tty (4).
This new tty driver allows generation of interrupt characters from the keyboard to tell jobs to stop. See
.1R stty (1)
for details on setting options in the new tty driver.

An instance of
.csh
begins by executing commands from the file `.cshrc' in the
.I home
directory of the invoker.
If this is a login shell then it also executes commands from the file
.1R login' there.
It is typical for users on crts to put the command 'stty crt' in their
.1R .login
file, and to also invoke
.1R tset (1)
there.

PP
In the normal case, the shell will then begin reading commands from the terminal, prompting with `%'.

Processing of arguments and the use of the shell to process files containing command scripts will be described later.

The shell then repeatedly performs the following actions:

- A line of command input is read and broken into words.
- This sequence of words is placed on the command history list and then parsed.
- Finally each command in the current line is executed.

When a login shell terminates it executes commands from the file `.logout' in the users home directory.

The shell splits input lines into words at blanks and tabs with the following exceptions. The characters

', ', ' ', ', ' ', '
form separate words.

If doubled in ', ', ' ', ' ' or ' ', these pairs form single words.

These parser metacharacters may be made part of other words, or prevented their special meaning, by preceding them with `$e'.

A newline preceded by a `$e' is equivalent to a blank.

In addition strings enclosed in matched pairs of quotations,

's*(aa', 's*(ga' or 's',
form parts of a word; metacharacters in these strings, including blanks and tabs, do not form separate words.

These quotations have semantics to be described subsequently.

etc.
Sort (Part 111)

$ last argument
& Repeat the previous substitution.
0 first (command) word
10 ex write.c
11 cat oldwrite.c
12 diff *write.c
[1] 1234
#(ua first argument, i.e. 'l'
#-slfyfR abbreviates '0-sslfysfRl'
#09 write michael
#sIn#fr sIn#fr#h argument
#sfixfr#f abbreviates sfixfr#h-S'
sfixfr#f,s-slfysfR range of words
d directory
e existence
f plain file
o ownership
r read access
s/sfixfr#f/sfrfrfrfr/# Substitute sfrfrfrfr for sfixfrfR
w write access
x execute access
z zero size

$\#name
$S
$X
$0
$?0
$?name
$Name
$Name[selector]
$number
${$name}
${?name}
${name[selector]}${name}
$\{number\}
(As in
Both
(See the description of
The
(The words
(as in
(e.g. '$shell').
(second form).
.. 
.B v-V
.B v-X
.B v-c
.B v-e
.B v-f
.B v-i
.B v-n
.B v-s
.B v-t
Sort (Part 111)

.9 $v
.9 $x
.9 alias
.9 alloc
.9 break
.9 breaksw
.9 breaksw
.9 breaksw
.9 cd
.9 chdir
.9 continue
.9 default:
.9 default:
.9 else
.9 else
.9 end
.9 end
.9 end
.9 endif
.9 endif
.9 endsw
.9 endsw
.9 endsw
.9 exit
.9 history
.9 login
.9 logout
.9 nice


etc.
Integer Arithmetic

/*
 * integer.c
 *
 * This program does integer arithmetic
 *
 * Compile by: cc -O integer.c -o integer
 *
 */

#define NAME "INTEGER"
#define COUNT 2900000 /* number of iterations */

main()
{
    long i;     /* iteration counter */
    long a, b, c, d;    /* integer variables for arithmetic */

    a = 1234; b = 2345; c = 3456; d = 4567;
    printf("%s: beginning (%d iterations)\n", NAME, COUNT);
    for ( i = 0; i<COUNT; i++ ) {    /* do some arithmetic */
        a = b + c - d;
        b = a * b / d;
    }
    printf("%s: complete (%d iterations)\n", NAME, COUNT);
}
Real Arithmetic

/*
 * real.c
 * This program does real arithmetic.
 * Compile by: cc -O real.c -o real
 */

#define NAME "REAL"
#define COUNT 600000 /* number of iterations */
float aa, bb, cc, dd;

int ii, jj, kk;

main()
{
    printf("%s: beginning (%d iterations)\n", NAME, COUNT);
    for(ii = 1; ii < COUNT; ii++)
    {
        aa = ii;
        bb = aa * aa;
        cc = (bb - aa - .137526)/aa;
    }
    printf("%s: complete (%d iterations)\n", NAME, COUNT);
}
Large Data Space

/*
 * largedata.c
 * This program has a data space larger than real memory.
 * Compile by: cc -O largedata.c -o largedata
 */

#define NAME "LARGEDATA"
define COUNT 20000 /* number of iterations */
define BLOCK 1024 /* 1K block */
define BSIZE 4000 /* large buffer size (blocks) */
define ADDR 0xe000 /* base address of array */

main()
{
    register char *curptr; /* current pointer */
    register long i; /* iteration counter */
    register long pagecount; /* number of new pages */
    register long limit; /* number of references */
    register long size; /* size of array */

    limit = COUNT;
    size = BSIZE;

    sbrk(ADDR + BSIZE * BLOCK); /* increase data space */
    srand(1); /* init random generator */
    i = 0;
    pagecount = 0;
    printf("%s: beginning (%d iterations, size %d)\n", NAME, COUNT, size * BLOCK);
    while ( ++i < limit ) /* make COUNT memory references */
    {
        curptr = (char *)(ADDR + ((rand() % size) * BLOCK));
        if ( *curptr == 0 )
        {
            pagecount++;
            /* increase new page count */
            *curptr = 1;
        }
    }

    printf("%s: complete (%d pages referenced, %d for the first time)\n", NAME, COUNT, pagecount);
}
Compile

// Compile and load of to routine
// compiletest
// to.c - C source
// subs.c - C source
//
// echo cc -O -c preaward/to.c
c -O -c preaward/to.c
echo cc -O -c preaward/subs.c
c -O -c preaward/subs.c
echo cc -O -o to.o subs.o
c -O -o to.o subs.o
r -f to.o subs.o
END DATE FILMED

8-12-87