Enc. 2

AS-1

ACME NOTES

Gio Wiederhold

A Summary of the ACME System

May 26, 1966

(Presentation given at the ONR Computer and Psychobiology Conference May 17, 1966 at the U.S. Navy Postgraduate School, Monterey, California)

The ACME computer system at the Stanford Medical School is designed to provide powerful computing to research laboratories in the Medical School. The type of computation service planned is regular batch processing (mainly at night) and real time interactive capability for on line experiments. The remainder of this description will concern itself with the proposed implementation of this latter facility.

In order to accommodate all the laboratories in the Medical School with their widely varying data rates, the system is designed to share the available computer time. The amount of time allotted to a user, however, is not one fixed unit per period, i.e. 200 ms every 10 seconds, but rather the time required by him to acquire and process one data point.

The data can be obtained in a number of different ways (see appendix CN-2, SK-1):

Data may be typed on a typewriter station. All textual and programming information is entered this way.

Slow to medium speed (up to 1000 samples/second) analog signals can be entered via a subsidiary computer which can either scan input voltages at the 1000/second rate or respond to separate interrupt signals given by the experiment or experimenter. The precision of the conversion can be up to 14 bits (.02%) precise. This computer will convert, reformat and preprocess these data and then transmit them to the main computer.

Digital input up to 16 bits wide can be handled in a manner similar to the analog input with the subsidiary computer, in order to serve users that have their own digitizing equipment or whose experiments actually generate digital output.

Users of the system with much higher rate demands, especially those that currently have small computers installed, will be able to connect directly to the main computer via a high speed (up to 125,000 samples per second) parallel data link. In this case they may program the interactive aspects of their experiments themselves, but have the resultant data processed on a larger machine.

The IBM 360 Model 50 which we propose to use as the main computer has been chosen with a configuration that will support this type of multiple user activity. Its main (core) memory size is one million bytes or characters, or 250,000 words or values. The backup storage is an IBM pie file, which stores data on strips of magnetic tape, 2000 of them, which are all individually retrievable within 0.6 seconds. It has a total capacity of 400 million bytes. The subsidiary computer proposed is an IBM 1800 process control computer, connected via a special direct channel to the main computer.

Results may be listed on the system printer, of course, but the emphasis will be on typing out the results on the typewriters in the laboratories.

Data may also be returned in analog or digital form via the 1800 at rates comparable to those of the input capability; generally within a few seconds after the results have been produced inside the Model 50. The digital lines may be used to drive plotters producing graphical summaries of the experiments. Display equipment of various types can be installed in the laboratories and be driven from the analog or digital 1800 outputs.

The small computers can also be used to distribute output via their typewriter, display tubes and plotters, or they may use the results to automatically control the continuation of the experiments.

It is quite obvious that a system of the described scope is not supported by any computer manufacturer. We are therefore designing and writing a simple but complete support package including an interactive compiler, a supervisory system, input-output procedures and data acquisition and distribution routines. The system design is such that continuous guidance is provided to the user via the typewriter (see ACME note RC-1 and appendix RU-1).

To make such an approach at all economically possible, we are programming using an IBM compiler (FORTRAN H) to allow us to write the system within the allotted time span. This should also make it possible to later share the results of our software development work with others. The language that our system will compile and which we hope will be acceptable to our non-computer specialist users is a subset of PL-1, a FORTRAN like procedural language, defined by IBM for use on 360 systems. Thus we hope that procedures checked and proven on our interactive compiler can be filed and used on ours and other computer libraries under standard systems (see ACME note on PL-1: PL-1).

The input-output system will include file handling and retrieval facilities where all data filed will be automatically labelled with all pertinent information to optimize the usefulness of the collected information. No elaborate buffering procedures will be used to minimize a user's inactive time, rather the time that he cannot use for computation will be turned over to the next user in the queue.

Since for some experiments the system reaction time will be quite critical, we will have to limit the number of these users in a given period. An attempt to utilize every available computer cycle for this work can only result in system overloading and failure. However, a number of users with non-critical problems, routine processing or information retrieval can balance the system to achieve reasonable total utilization, and we plan also to provide facilities for this type of use as soon as the more critical uses are satisfied.

The project currently uses several different computers around Stanford to check out parts of the system and is preparing to do a simulation study of the queuing algorithm. A small technical group is building prototype interconnection equipment for the various data interfaces.

The current planning work is sponsored by a Macy foundation planning grant and further funding has been requested from NIH. Much credit for ideas and procedures goes to other computer installations and other people, notably project MAC at M.I.T., MEDLAB at the Latter-Day Saints Hospital in Salt Lake City, the University of California at Berkeley Computation Center and ARPA project, U.C. San Francisco medical school, U.C.L.A. Health Sciences, etc., and of course the Computation Center and the Computer Science Department of Stanford itself.

Appendix 1

Current Work in Progress

Design and building of interconnection between LINC and PDP-8 computers to the IBM 360/50 with an IBM 2701 interface (see ACME Note HL-1).

Design and building of a status display box to be used at typewriter stations (see ACME Notes LI-2 ~~and LJ-1~~).

Design and checkout of the interactive compiler (see ACME Note PL-1).

Design of file system (see ACME Notes FI-1 and FL-1).

Programming of the input-output supervisor (see ACME Note IO-1).

Design and checkout of typewriter control system (see ACME Note KA-1).

Collection of information of possible display devices (Goethe Link).

Definition of parameters for simulation of user interaction (see ACME Note QP-1).

1800 computer software system evaluation and design.

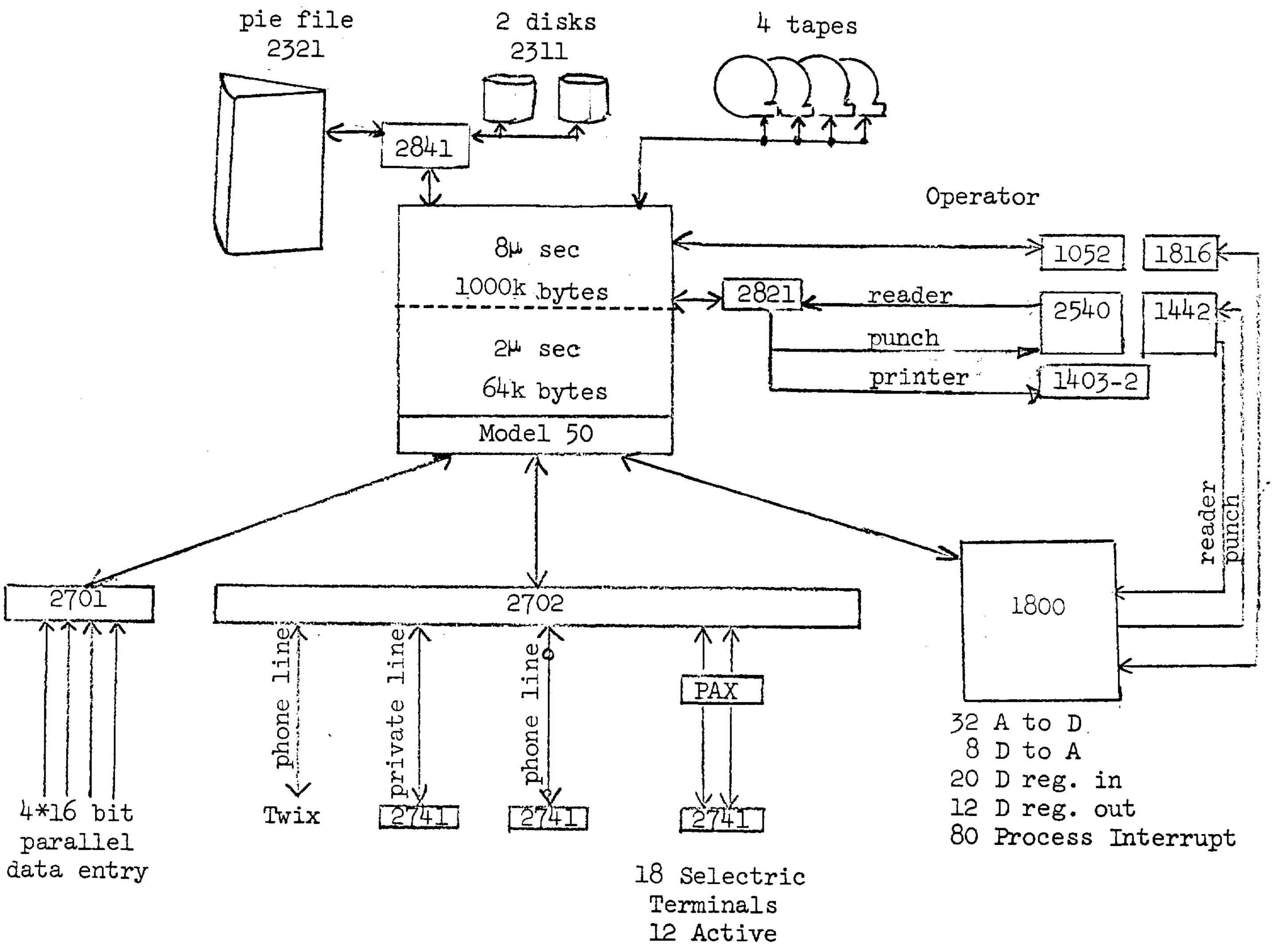
CN-2

ACME NOTES

Gio Wiederhold

April 21, 1966

Configuration of Machine



LINCs
PDP-8s
Other Digital
Input

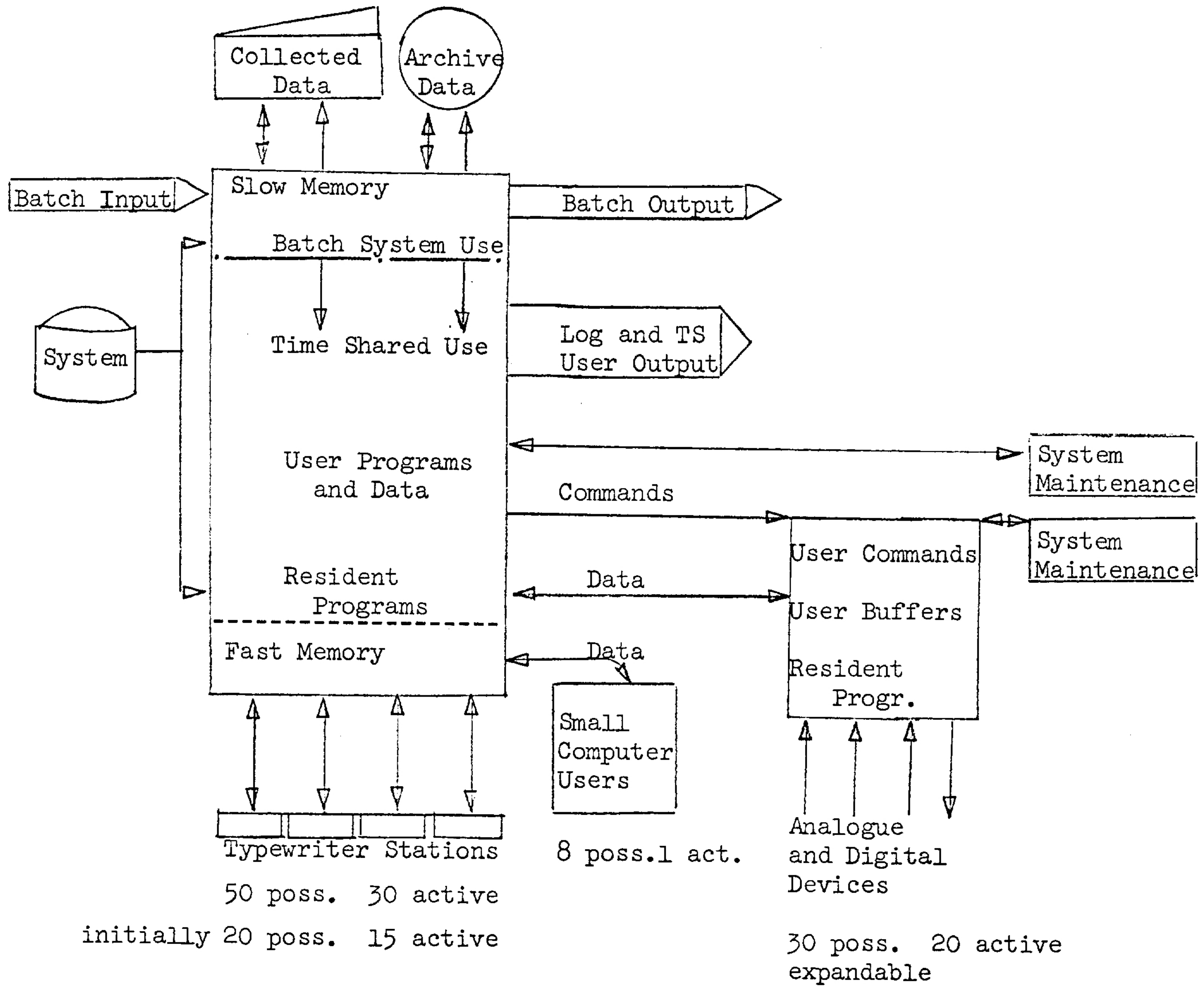
Revision of CN-1 dated January 12, 1966

TS NOTES

Gio Wiederhold

ACME - Sketch of Proposed System

January 17, 1966



Model 50

1800

Selector Channels: 1800

Pie, disk, tape, small computers

Multiplexer Channels: Teletypes (2)

Batch I/O (1 - 3)

System Maintenance

Revision of SK-1 dated December 13, 1965

ACME NOTES

J. Miller/G. Breitbard

Example of an ACME Run

April 22, 1966

The user turns on the terminal and ACME types
NAME?
All statements typed by ACME are followed by a question mark.

The user then types his name.
NAME? BEN CASEY!

Except when entering a program, all statements typed by a user must be followed by an exclamation point. Program statements are followed by a semicolon.

The run continues as follows:

```
PROJECT?  RATS!
YOU CAN BEGIN, 13:30  11 MAY,  NAME OF PROGRAM?  SAMPLE!

1.  ?        Stats:  PROCEDURE;
2.  ? loop:  GET FILE (Instrument)  (value);
3.  ?        n = n + 1;
4.  ?        sum = sum + value;
5.  ?        sum_sqr = sum_sqr + value**2;
6.  ?        Mean = sum/n;
7.  ?        std_dev = SQRT ((sum_sqr - (sum**2/n))/(n-1));
8.  ?        PUT FILE (Console) (n, mean, std_dev);
9.  ?        PUT FILE (Save) (value, mean, std_dev);
10. ?        GO TO loop;
11. ?        END Stats;
PROGRAM SIZE 431 BYTES, PROGRAM LINES 11.
THIS ENDS 'Stats'.  RUN?
```

The program Stats reads an experimental instrument, calculates the mean and standard deviation of the observations, types these quantities on the experimenter's console, and also saves them. GET and PUT are respectively the input and output statements. At this point the program has been compiled into core memory and also saved in the user's file of programs.

Before executing this program the user must set the initial values of n, sum, and std_dev. ACME has asked whether immediate execution is desired by the question RUN? A response of YES! would start the program running. Whenever the user does not wish to follow ACME's prompting, he indicates this by giving a carriage return instead of typing a response. In this case the user gives a carriage return and types a command to set the initial value of n.

```
n = 0!
```

All PL/ACME statements followed by exclamation points are executed immediately, and are not saved on the user's file of programs.

```
? sum = 0!
? sum_sqr = 0!
? CALL Stats!
```

The CALL statement causes ACME to begin execution of the program 'Stats'.

In the case of this example ACME will immediately type:

```
ZERO DIVISOR,  LINE 7.  IN 'Stats'.  ATTN?
```

The user failed to consider the special case n = 1. The program continues to execute. If the user wishes to take corrective action, he may get ACME's attention by hitting the ATTN key. He may then stop his program, reset the initial values, and restart.

```
? STOP!
? n = 1!
? sum = 0!
? sum_sqr = 0!
? CALL Stats
```

or, better, he can change the program so it can handle the special case. To insert a line in his program, he may hit ATTN, stop his program, and type a line number for the line to be inserted, followed by a statement:

```
? STOP!
? 5.5 IF n = 1 GO TO loop;
```

This statement will cause the computation and output to be skipped when n = 1. ACME will insert Line 5.5 between Lines 5. and 6. (Any line number between 5. and 6. could have been used, e.g. 5.015).

The user now commands ACME to resume execution of his program.

```
?  CALL Stats!
         n = 2.
      mean = 3.5
   std_dev = 0.713
         n = 3.
      mean = 4.16
   std_dev = 0.6195
         n = 4.
And so on ...
```

The user may now decide he wants to type out these quantities only every hundredth time. He may also decide he wants to see the current value as well as the mean and standard deviation. This will require changing line 8. To do this he simply types the line number, 8., followed by a statement. This statement will replace the previous statement at line 8. However ...

ACME is typing output at full speed.

To get ACME's attention, the user pushes the ATTN button on his console. ACME immediately interrupts what it is doing and types:

```
?
```

The user can then modify the program. While he is entering the new statement, the program will continue to run, so no data will be lost. Programs are stopped only by the PAUSE or STOP statements.

```
? 8.  IF MOD (n,100) = 0 THEN PUT (n, value, mean, std_dev);
```

If no file is specified in the PUT statement, ACME assumes the console is meant.

```
       n = 100.
   value = 4.01
    mean = 4.2751
 std_dev = 0.3271
       n = 200.
   value = 5.62
    mean = 4.9135
 std_dev = 0.7216
LIMIT CHECK ON 'Instrument', VOLTAGE OF 6.2173 EXCEEDS LIMIT OF 6.0000.  ATTN?
```

ACME has detected an error in the input from the user's experimental equipment. If the user does not want to wait while ACME types out the entire message, he may hit the ATTN key as soon as he recognizes the message, and ACME will type an ellipsis followed by a question mark, and stop.

```
LIMIT CHECK ON "Ins ... ?
```

The user might now stop his program and go check his equipment. Upon returning to the console he may decide to modify his program and restart it.

```
STOP!
? 1.1  PUT  ('INITIAL VALUES');
? 1.2  GET  (n, sum, sum_sqr);
?      CALL Stats!
```

The program has now been modified to prompt the user to enter the initial values.

```
INITIAL VALUES
      n = ? 0
    sum = ? 0
sum_sqr = ? 0
      n = 100
  value = 3.17
         .
         .
         .
```

However, if he wants to execute his program without modification, he may immediately CALL it.

When the user is finished he logs off:

```
? LOGOFF!
LOGOFF 01:15  12 MAY.  LOGON?
```

If he resumes his console work another time the following sequence might occur.

```
NAME?  BEN CASEY!
PROJECT?  RATS!
YOU CAN BEGIN, 9:20  13 MAY.  PROGRAM NAME?  Sample!
YOU ALREADY HAVE 'Sample' FROM 16.15, 11 MAY 67.
RUN?  YES!
INITIAL VALUES
n = ?
```

The user might wish to modify his program before running it. He may then type NO!

```
RUN?  NO!
MODIFY?
```

He may now type a line number and a statement. Or he may again type NO!

```
MODIFY?  NO!
DELETE?  YES!
DO YOU WANT TO DELETE 'Sample' of 16:15, 11 MAY 67?  YES!
NAME OF PROGRAM?
```

ACME checks to make sure he wants to DELETE the program "Sample' from his file of programs.

[A continuation of this note will be published soon, giving further examples of the program debugging capabilities of the ACME system.]