the Availability Digest

www.availabilitydigest.com
@availabilitydig

High Availability, 1970s Style

October 2016

In 1972, I started a small payroll services company, MiniData Services, Inc. The intent behind MiniData was to provide payroll services to small companies via inexpensive computers. Our billboards read “You pay $9, We pay 15.” In other words, we would do a fifteen-man payroll for $9 per pay period.

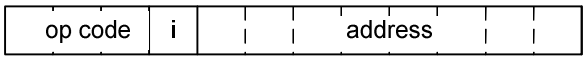
What were inexpensive computers? The choice at the time was the PDP-8 from Digital Equipment Corporation. A basic PDP-8 sat on a desktop. It was a 12-bit machine with 4K of memory (yes, kilo, not mega or giga). It was the first successful commercial minicomputer.

Digital PDP-8

Why a 12-bit word length? At the time of its introduction, the 8-bit byte had not yet been formulated. Characters were stored in six bits, giving enough combinations for the upper and lower case letters, numbers, and a couple of special characters. A PDP-8 word could store two characters. With a 12-bit word length, the addressing capability was 4K words. Thus, the basic PDP-8 was configured with 4K of memory.

Additional 4K memory banks could be added and could be addressed by switching memory banks via special commands.

A PDP-8 instruction included a three-bit operation code, an eight-bit address field, and an indirect address flag (i). The eight-bit address field divided the 4K memory bank into sixteen “pages” of 256 words each. Any word in a page could be addressed directly via an instruction in that page. If the indirect address flag were set, the 12-bit address in that location was used as the operation address, thus allowing access to all 4K words in a memory bank via indirect addressing.



PDP-8 Instruction Format

Of course, to run a payroll service, we needed a bit more than a basic PDP-8. First of all, we needed two systems. Payrolls had to get out on time. If our PDP-8 failed, that was not an excuse to fail to deliver paychecks to the employees of our customers. Therefore, we purchased two PDP-8s. Active/backup systems kept synchronized via data replication were unknown at the time. Rather, we would normally use both systems to process different payrolls. However, if one system failed, we could still get the payrolls out using the surviving system. This was high availability, 1970s style. (Interestingly, in the many years that we used PDP-8s, we never had a system failure.)

Panel Switches for Entering the Bootstrap

We also had to beef up the peripherals on our systems. The basic PDP-8 came with a Model 33 Teletypewriter as a console. It did not come with an operating system – only device drivers. In order to boot the system, one had to enter a 22-word bootstrap routine via the switches on the front of the system and then load the console driver (the Model 33 Teletypewriter driver) via the paper-tape reader of the Teletypewriter.

We needed to add disk memory, magnetic tape, a high-speed line printer, and more memory to our systems. The magnetic tape drives were Digital's DECTape drives. They were unique in that they were formatted in 256-word blocks. Any block on a tape could be located, read, updated, and rewritten to tape in the same location. Thus, the DECTape drives served as an extension to the disk drives.

A Fully Configured PDP-8

We added a second 4K memory block to each system, giving a total of 8K of memory. Digital's disk drives provided 32K words of memory. We added two disk drives to each system, giving a total of 64K of disk capacity to each system.

Then came the software. We had to write a payroll package that would process up to fifty small payrolls simultaneously (this is how we planned to provide payroll services so economically). It was not feasible to write this package in assembly language (which at the time was the only option on the PDP-8), so we developed our own software language – SAIBOL-8. SAIBOL stands for SAI Business Oriented Language (a takeoff on COBOL). SAI was the name of my company – Sombers Associates, Inc. (Digital later came out with a similar language called DIBOL, but it was too late for us).

Fortunately, the PDP-8 provided a convenient mechanism for developing SAIBOL-8. One of its instructions was a JMS instruction – Jump to Subroutine. We defined a language that was COBOL-like. Each verb (such as MOVE or ADD) was implemented as a subroutine. The name of the verb was defined to be a JMS instruction that called the verb subroutine via an indirect address in the following location. Parameters of the call were stored in the words following the JMS instruction so that the verb subroutine had all the information it needed to perform the operation. For instance, to move twenty characters from location A to location B, the SAIBOL-8 statement was MOVE 20 A B. It would be compiled as the assembly language sequence:

```
JMS *+1     (Jump to the subroutine whose address is in the following word)
MOVE        (MOVE subroutine address)
20          (character count)
A           (from address)
B           (to address)
```

where "*" indicates the current address.

We called the SAIBOL-8 compiler an "interpretive compiler." Fortunately, we did not have to write a compiler. Since all the compiler did was to take a SAIBOL-8 statement and convert it to a PDP-8 assembly language sequence, we simply modified the Digital PDP-8 assembler to "interpret" a SAIBOL-8 statement as an assembly language statement.

Fortunately, the entire set of SAIBOL-8 verb subroutines fit within one 4K bank of memory. We used the first bank of memory in our PDP-8s to store the SAIBOL-8 verb subroutines and paged various applications into the second bank of memory. Consequently, we could write all of our payroll applications in a COBOL-like language using only 8K of memory.

One problem we had was that our payrolls were small (an average of seventeen employees), and each company needed its own checks with its name, its bank's name, and the MICR (magnetic ink character recognition) encoding printed on its checks. If we had to break down the printer and change check forms

for every customer, it would take hours each day (1,000 customers per day and one minute to change checks would add sixteen hours per day!).

So what we did was to process fifty customers at a time and print all of their checks with their names and bank names on the checks. We would then burst the checks and run each company's checks through a MICR encoder to print the MICR information on the bottom of the check. We were told by many that this wasn't legal, but we never had any problem with the practice.

Our PDP-8s cranked out payrolls for our customers for many years. However, the onset was almost a disaster. We sold our first payroll service to a local small bank. When we delivered the bank's first payroll, our telephone rang; and we were met with cries of anguish. It turned out that they delivered their payroll information to us as hours worked, but our payroll program took the weekly pay as the hourly pay. Paychecks were issued for thousands of dollars each. We quickly learned from that experience to check each payroll very carefully before delivering it. We also did payrolls manually, comparing them to the computer output, before delivering them. We did this until we were absolutely confident in our payroll processing.

The PDP-8s served us well for many years until we upgraded to Digital's PDP-11 and its more modern 16-bit word size.

Early on, I had hired a superb technologist turned businessman, Ed Bindel, to run MiniData. He grew MiniData over almost twenty years and then arranged the sale of MiniData to Control Data Corporation (CDC) in 1991. At the time, MiniData was processing over 5,000 payrolls.

A final thought. Can you imagine today programming a multi-company payroll on a computer with only 8K of memory? Back then, we didn't know it couldn't be done. So we did it.