Chapter 50

Hewlett-Packard Calculator Architectures

Thomas E. Osborne

Summary *This chapter focues on some of the more important architectural differences between the first Hewlett-Packard electronic calculator, the HP 9100A (c. 1968), and its descendants, the HP 9810/20/30. The architectures of the two generations are so different that the reasons for making the change are, in many ways, as interesting as the differences in the architectures.*

Except for using similar components, the early programmable calculators had surprisingly little in common with the concurrent digital computers. Among the many reasons for the differences, none is as large as the fact that very few, if any, of the early calculator designers were defectors from the ranks of computer designers. Contrary to the old adage, anyone engaged in designing computers could see that the grass was *very* green on his side of the fence— so green that few even acknowledged the existence of other pastures.

A quick glance at some of the objectives of the early electronic calculator manufacturers shows that even if there had been defectors from the computer field, precious little technology would have been directly applicable to the calculator environment.

Would a technology that understood megabit core memories designed to operate in a controlled temperature environment have been capable of stretching enough to design inexpensive kilobit memories to operate from 0 to 55° C? Was there any assurance that those who were skilled in the design of microsecond parallel binary adders would find these skills useful in designing inexpensive serial decimal adders? Who knows? The test was never run.

Because the early calculator designers had so little practical exposure to the inner workings and hidden mechanisms of computers, **it** follows that their designs would not necessarily be an extrapolation from the concurrent computer architectures. Both groups of designers had the same building blocks and shared somewhat similar problems, but in the same way that different life forms sprang from the same primordial soup, early calculator architectures were quite a different species from concurrent computer architectures.

The first Hewlett-Packard programmable calculator, the HP 9100A (c. 1968) [Hewlett-Packard, 1968], was micro-programmed to perform floating-point arithmetic and to evaluate the forward and inverse circular, exponential, and hyperbolic transcendentals. Its I/O was also controlled by its inductive ROM, which contained 512 64-bit words. The ROM's extra-wide micro-words allude to the nonstandard architecture found in the HP 9100A.

Instead of having an arithmetic and logic unit (ALU) connected to various registers by a common bus, the HP 9100A had no ALU per se. Instead, its arithmetic section was distributed throughout the system by assigning small, but specialized, tasks to the various registers within the system. Separate buses then interconnected selected pairs of these registers. As a consequence, several (typically three to seven) micro-instructions were executed simultaneously during each micro-word time. Half of the 64-bit micro-word was used to encode these micro-instructions. The other half of the 64-bit word was used for addressing and to test conditions of both internal registers and asynchronous external signals generated by I/O devices. The results of each test determined the next ROM address and, at the microprogrammer's discretion, whether or not to suppress the action of the micro-instructions in the other half of the ROM microword. Figure 1 pictorially demonstrates the syntax of a typical micro- word used in the HP 9100A.

The micro-programmer would interpret Fig. 1 as follows:

"When the conditions defined by QX are true, execute the set of micro-instructions (IA, IB. IC, . . .) and go to ROM address 'j' for the next micro-word; when QX is false, inhibit all microinstructions [symbolically shown by the shaded corner in the right exit of the diamond] and go to address 'k' for the next microword." Notice that, unlike what happens in a standard flowchart, in which the instructions within the box precede the test designated by the diamond, both actions occurred simultaneously in the HP 9100A.

The advantages of conditionally inhibiting a set of microinstructions are many, but one frequent use stands out. In most

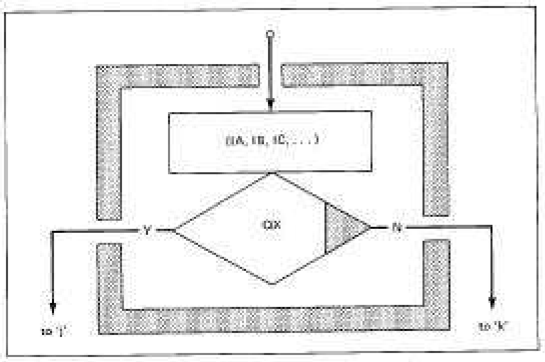

Fig. 1. A typical microword from the HP 9100A.

previous | contents | next

applications, one tests the exit conditions of an iterative loop (1) to re-enter the loop when the exit conditions are not met or (2) to exit when they are met. When the test occurs at the beginning of a loop and the exit conditions are met, none of the loop instructions are executed. In the HP 9100A, loops of this type frequently consisted of a single micro-word to he repeated until the exit conditions were met, at which time the micro-instructions were inhibited and the exit path selected.

As one would expect, this highly parallel architecture resulted in a very fast system. Contrary to what one might expect, the actual micro-coding was not overly difficult, nor was the multi-bus hardware difficult to lay out on PC boards. Although the "wide word" architecture of the HP 9100A was abandoned for reasons to be discussed later, **it** seems that **it** would be worth revisiting. One might find that micro-processor speeds could be enhanced by an order of magnitude without resorting to a higher-speed IC process.

Quantifiers are tested to determine which of the two next micro-address fields should be used. In Fig. 2 a single signal, IQAM, connects the QUALIFIER SELECTION **LOGIC** to the MAIN CONTROL LOGIC. This signal is actually the output of a comparator whose right inputs come from the QUALIFIER subfield of the ROM and whose left inputs come from an encoder whose inputs are, in turn, all of the individual conditions to be tested. An unobvious, but very beneficial, byproduct of this technique is the fact that the entire system is free from problems resulting from *any* external qualifiers' asynchronously changing state between clock times. The signal IQAM either is effective upon the CONTROL LOGIC or isn't. In either case the CONTROL LOGIC is deterministic in its response to IQAM.

It is interesting to notice that the patents obtained for the HP 9100 did not use the words *micro-program* and *micro instruction.* We were not aware of either term until after the HP 9100A was introduced, even though **it** was a micro-programmed machine.

Before discussing some of the aspects of the stack as seen by the

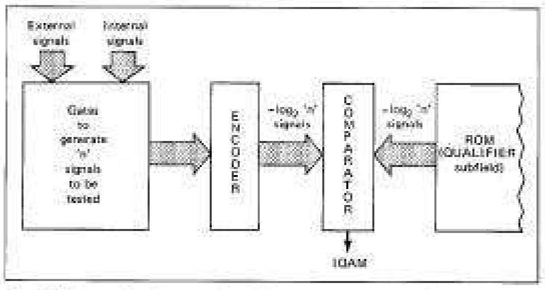

Fig. 2. Next microinstruction address select logic.

HP 9100A user (i.e., the X and Y registers), one must first appreciate the fact that calculator designers and computer designers are in violent disagreement as to the way in which a stack is visualized. Since there are about 100 times as many calculator users visualizing the last entry at the bottom of the stack as there are computer users visualizing the last entry at the top of the stack, we will side with the majority. We justify this position with the attitude that the same person who "inverted" the computer stack was responsible for deciding that "trees" have their "roots" above their "leaves." Regardless of how the minority view came about, the calculator stack is as **it** is because we wanted the numerator to be above the denominator when a division is performed.

For nontechnical reasons, the HP 9100A presented a unique stack to its users. Prior to the introduction of the HP 9100A, one of the well-established mechanical calculator users introduced an electronic calculator having a classic stack (with its last entry at the bottom). To avert any possible patent infringements, we invented a stack in which a dyadic operator overwrote the first entered operand (the second operand in the stack) and left the bottom of the stack unchanged. (A classic stack could be thought of as performing the same operation but "popping" the stack following the dyadic operator.) In the HP 9100A, an operand that followed any operator simply overwrote the bottom of the stack, with the result that there was no difference between the classic stack and our version in executing a series of dyadic operators. In fact, when an evaluation required multiple uses of the last stack operand, as often occurred, our version was superior to the classic version. Nonetheless, our stack had one shortcoming. When a monadic operator was to he performed upon the result of a dyadic operator, **it** was necessary to manually "drop" the stack prior to performing the monadic.

We found that the public adapted very well to the concept of a stack. The fact that **it** was displayed in its entirety on a CRT helped, but the fundamental nature of the concept was even more important. The HP 35 only displayed the bottom of the stack, and **it** was well received.

Even today, if these were the only choices of stacks, **it** would be difficult to choose between the two. Fortunately a third choice exists which has the advantages of both. It functions as a classic stack with the feature that the bottom of the stack, which is consumed by an operation, is saved prior to performing the operation and made available for recall as the next operand. Those familiar with any of the later versions of HP hand-held calculators will recognize this feature as the "LAST X" operator and can testify to its usefulness.

As mentioned earlier, the wide-word architecture of the HP 9100A was not used in other HP calculators. No single factor contributed to the demise of the HP 9100A architecture as did the HP 2114 mini-computer. And for good reason. An investigation of its architecture showed that **it** could be reduced to an MSI

previous | contents | next

previous | contents | next

micro-processor and that its software could he locked up in ROM. By simulating the calculator on a copy of the HP 2114, an order-of-magnitude increase in performance was realized over the hand-soldered diode-ROM simulator used to develop the HP 9100A. On top of that we thought we could use the software and operating system that had been developed for the HP 2114. As it turned out, little of either was used, hut the fact that it existed weighed heavily in the decision to abandon the HP 9100A architecture. Finally, it seemed wise to use the same I/O protocols adopted by the HP 2114.

Shortly after the decision was made to use the HP 2114 architecture, it became evident that portions of the system had to be enhanced to meet the speed required by the three calculators that were to use the 2114 "micro-processor." The MSI version of HP 2114 was enhanced by including a decimal adder in the ALU, and by expanding the instruction set to include an extensive set of macro-instructions whose principal use was to assist floating-point decimal operations.

One of the more interesting consequences of the first generation RAM's was the dramatic impact they had upon scientific desk-top calculators like the HP *9810/20/30* [Hewlett-Packard, 1972]. Most, if not all, of the early RAM's were aimed at replacing large core memories found in digital computers. To achieve this end, the LSI RAM manufacturers were aiming at a price of about two cents per bit. As anyone who has designed a core memory knows, its driving and sensing circuitry is inherently expensive. When this cost is prorated among the few bits found in a small memory like the one used in the HP 9100A, one could easily encounter costs of 20 cents per bit. Fortunately for the calculator designers, the per bit cost of LSI RAM was relatively independent of the ultimate size of the memory. Whereas two cents per bit was a break-even point for the computer environment, it represented an order-of-magnitude improvement for the calculator environment. It was no coincidence that many of the first production orders for RAM's went into programmable desk-top calculators.

Unlike the operating systems of digital computers, which share user RAM, the operating systems of the HP 9810/20/30 were all committed to ROM. As one of the HP engineers recently said, "If you think it takes nerves of steel to release a software operating system, imagine how it feels to release one in firmware."

The firmware of the 9800 Series calculators supports some rather sophisticated features. At the low end of the line, the HP 9810 performed indirect register arithmetic. For example, the sequence

$$\boxed{X \rightarrow (\ )} \quad \boxed{X} \quad \boxed{\text{ind}} \quad \boxed{50}$$

will multiply the contents of the X register by the contents of the register whose address is found in register 50 and place the product in that register.

The HP 9820 was Hewlett-Packard's first algebraic (non-Polish) machine. Instead of running interpretively, it partially compiled its source programs into reverse Polish strings, which accomplished two important objectives. First, the lexical and parsing phases of the compilation were only performed once, but second and more important, a "decompiler" was able to reconstruct the source code from the reverse Polish strings, thereby giving the user unprecedented on-line editing of his source code without storing the source code in user memory.

The HP 9820 firmware also supported recursive functions. For example, one could define a function like MAX and then use it as one of its own arguments. For example, MAX(A, MAX (B,C)) is a valid

HP 9820 statement.

At the upper end of the calculator spectrum, the HP 9830 supports BASIC. In its maximum configuration, it offers 16K words of 16 bits to its users. Remember that this is all user memory. The operating system is in ROM. An additional two-and- a-half million words of memory is available in the HP 9880 mass (disc) memory.

Since the introduction of the HP 9830, LSI micro-processor versions of the HP 9810/20/30 have been introduced. As one might expect, they run faster, cost less, and do more than their ancestors. How they achieve these ends will be left to the authors of future research papers.

So far the reader has been told why things are as they are and very little about why they aren't something else. Why, for example, was the HP 9830 not given a CRT? Although one was considered, the designers did not want their product to be confused with the competition of the day, terminals driven from remote computers. It was thought that the combination of a 32-character LED display and an 80-column, 300 line per minute thermal printer would be an adequate solution and at the same time prevent confusion in the minds of potential customers as to what a terminal was and what a stand-alone computing station was. Whether or not this was the best decision remains open. However, one thing is clearly obvious: it was not a bad decision. The HP 9830 is a very successful product.

Negative experiences, though painful, can be helpful in the long run. Such was the case with the HP 9100A I/O. The signals emanating from its I/O connector were fondly referred to as "semi-modulated white lightning." The peripherals that connected to the I/O connector stand as a tribute to engineering. The obstreperous nature of the HP 9100A **I/O** was a strong contributor to the fact that its descendants have excellent I/O characteristics.

Another disturbing fact of life surrounds the desire to maintain the same fundamental architecture between successive generations so that the software from previous generations can be used

previous | contents | next

in the generation being developed. Unfortunately, two factors team up to defeat this admirable goal. First, it seems that regardless of how carefully the previous software was documented, the (new) personnel who will be responsible for the next generation find the ancestor software's documentation dull and uninspired. Second, the performance objectives of the new generation automatically render much of the ancestral code obsolete. Once Pandora's box is opened, good intentions escape and most of the code is rewritten. The paradoxical result of this process is that an architecture is retained because one wishes to capitalize upon its software when, in fact, very little of the software is actually used. The measurable effect of this logic is that an architecture may persist much longer than it would if its software were removed from the decision process.

It has been more than a decade since the first scientific desk-top calculator was introduced. Since that time the public has had enough time to appraise many systems and use their respective languages. To me it appears that as we add more deep structure to the grammars upon which the languages are based, the users become more confused. In an odd way, the more powerful we make our grammars, the less useful they become to all but a few. The following BASIC statement is an example:

```
1 − 1 OR NOT Ø AND Ø = 2 ↑ SQR Ø * Ø
```

Unless you have committed to memory the nine levels of hierarchy involved, there is little chance that you would parse the sentence as:

```
(1 − 1) OR ((NOT Ø) AND (Ø = ((2↑ (SQR Ø)) * Ø)))
```

My point is that while the first sentence is derived from what many consider to be a trivial grammar, the sentence can only be understood if the user has memorized the proper grammatical rules or has access to a manual in which they are described. The entire meaning of what is being said is lost if the rules of the language are forgotten or improperly used.

Based upon my experiences, a language that is easily learned and seldom misused would:

1 Be free from hierarchy, except for parentheses

2 Be left-associative

3 Use the right assignment operator "→ "

In other words, it would be much like APL except that it would execute left to right rather than right to left.

References

Hewlett-Packard [1968]; Hewlett-Packard [1972].