Branches and Subroutines Calling: 

- **JMP**: 
  - sig = 0001s
  - (jump unconditional)
- **BR**: 
  - bro = brop = 01s
  - (branch unconditional)
- **BNE**: 
  - bro = 02s
  - (equal to zero)
- **BLT**: 
  - bro = 05s
  - (not equal to zero)
- **BGE**: 
  - bro = 04s
  - (less than zero)
- **BLE**: 
  - bro = 06s
  - (greater than or equal to zero)
- **BGT**: 
  - bro = 07s
  - (less than or equal to zero)
- **BCS/BHS**: 
  - bro = 08s
  - (carry set; higher or same (unsigned))
- **BCC/BHS**: 
  - (carry clear; lower (unsigned))
- **BPL**: 
  - bro = 00s
  - (lower or same (unsigned))
- **BVC**: 
  - bro = 04s
  - (overflow)
- **BVT**: 
  - bro = 05s
  - (no overflow)
- **BEQ**: 
  - bro = 03s
  - (minus)
- **BNE**: 
  - bro = 02s
  - (plus)
- **BGT**: 
  - bro = 06s
  - (jump to subroutine by putting R[ss], PC on stack and loading R[ss] with PC, and going to subroutine at D)
- **JSR**: 
  - sig = 0040s
  - (return from subroutine)
- **RTS**: 
  - i = 000200s
  - (return from interrupt)

Miscellaneous processor state modification:

- **RTI**: 
  - i = 2s
  - (return from interrupt)
- **HALT**: 
  - i = 0
  - (trap to M[34s] store status and PC)
- **WAIT**: 
  - i = 1
  - (enter new process)
- **TRAP**: 
  - i = 3
  - (emulator trap)
- **IOT**: 
  - i = 4
  - (I/O trap to M[20s])
- **RESET**: 
  - i = 5
  - (reset to external devices)
- **OPERATE**: 
  - i(3:15) = 5
  - (condition code operate)
  - i(4) → (CC ← CC ∨ i(3:0));
  - ¬i(4) → (CC ← CC ∧ ¬i(3:0));

end Instruction execution
A systems approach to minicomputer I/O

by FRED F. COURY
Hewlett-Packard Company
Cupertino, California

INTRODUCTION

You can tell a lot about a guy by the way he draws a block diagram of a computer system. If he draws the central processor and memory as small boxes off in a corner, then proceeds to fill the page with an elaborate portrait of the input/output system, he is usually referred to as (among other things) an "I/O type".

I have drawn several such diagrams, and I offer this information as a caveat to the reader.

In the pages to follow, I shall outline and attempt to justify some of my views on minicomputer I/O, particularly on "where we should be going from here". If some of the suggestions are already being implemented, I think they are steps in the right direction. If, on the other hand, some of the ideas seem too far out, consider the source.

A BIT OF HISTORY

I guess things started the way they did for several reasons. Hardware (relays, vacuum tubes, power supplies, and air conditioners) was very expensive, especially in the large quantities necessary for computing. The resulting machines were so incredibly complex (literally thousands of relays and vacuum tubes) that just getting one to work was a major accomplishment. In spite of the complexity involved, the actual capability of the early machines was limited to large-scale automatic number-crunching.

It is not hard to understand that hardware optimization was foremost in the designers mind. Unfortunately, programming these first machines was quite difficult due to the limited storage available in the machines, and also due to the fact that no programming frills (such as assemblers) were provided.

I/O was no real problem, since most of the early machines were clearly compute-bound, especially in number-crunching applications, and most I/O was simple card input, line printer (or card) output.

Engineers took advantage of technological developments (core memories, transistors) to build faster, more powerful machines. Programmers began to apply the new machines to a wide variety of problems (such as writing assemblers) and began to explore the true potential of computers.

As the number of machines increased, users (programmers) began to outnumber designers (engineers). They wanted to have something to say about the design of the machines they would be using before it was too late.

The engineers made the computers work, but the programmers made the computers do something. It was recognized that the important parameter to optimize was overall system performance. The engineers had to worry not only about how fast a machine could multiply two numbers together, but how efficiently the machine could be programmed to invert a matrix.

It is now common practice for computers to be designed by teams of engineers (with programming experience) and systems programmers (with hardware understanding) in order to optimize the overall performance of the resulting hardware/software system.

Also, the emphasis is shifting from hardware minimization to people optimization. As the cost of hardware goes down, and the cost of people goes up, the way to minimize cost is to maximize the efficiency of people in the design, production, programming, and eventual use of the system.

A GLIMPSE INTO THE FUTURE

In the near future, especially in some of the new minicomputer markets, the vast majority of computer users will not be programmers. As a matter of fact these users will not want to program computers. They won't particularly even want to use computers. They will have questions to be answered, problems to be solved, and things to be done. If a computer offers a better way (or, in some case, the only way) to do it,
people will consider using a computer. Otherwise, they will choose another method, or not do it at all.

Let's face it...the novelty is wearing off. The small computer industry must come of age. We are approaching the same position as the commercial airlines are in now. People don't fly just because they want a plane ride. They want to get somewhere, and flying happens to be the best (fastest, cheapest, most convenient) way to get there. If it's not, they will choose a better way to go or they will stay at home.

And most people are no longer interested in "roughing it" (wearing goggles and helping to start the engines). In most cases, the less they are aware of the fact that they are flying, the better they like it. This attitude is reflected in boarding ramps at the airports, and music, drinks, dinners, and movies while in flight.

And people are only interested in new developments insofar as they are directly affected. A revolutionary new jet aircraft design is of interest only if it means a faster, quieter, or more comfortable trip. Note what is stressed in the Boeing 747 advertisements. New navigation, propulsion, and control systems are ignored in favor of winding staircases and plush accommodations. Pilots fly planes, people pay to ride in them, and there are a lot more people than pilots.

The same rules will apply to minicomputers. New architectures, bussing structures, and addressing modes are only appreciated in terms of benefits which the user can see. Applications programs will be written for the user, not by him, and he will only be interested in the performance of the entire system as it affects his particular problem.

THE MYTH OF THE ULTIMATE PROCESSOR

But we are continually improving our machines. We are coming up with better performing hardware/software systems every day.

I don't think that faster processors and more powerful languages are the whole solution. Let me illustrate by carrying the current trends to their ultimate goal.

Suppose a man wants to generate an amortization schedule for a home loan. State of the art in minicomputers has reached the point where he can get a zero-cost infinitely-fast processor with 4K of memory and a super-powerful new compiler called "ENGLISH". The steps he goes through to generate the amortization schedule may be familiar to many readers:

1. He sits down to tell the computer (in "ENGLISH") to generate a loan amortization schedule. He discovers that no I/O device was provided. So he buys a teletype (with controller) for $2,000.

2. He tries to load the "ENGLISH" compiler paper tape into the machine. Discovers that "ENGLISH" requires 8K of core; he only has 4K. So he buys another 4K of core for $5,000.

3. He is about to load "ENGLISH" when he discovers that the MTBF on the teletype is shorter than the time it takes to load the tape. So he buys a high-speed photoelectric paper tape reader for $3,000.

4. He loads the "ENGLISH" compiler.

5. He types (in "ENGLISH") "GENERATE AMORTIZATION SCHEDULE (CR, LF)"

6. Immediately, the system starts to punch a binary tape. However, halfway through, the teletype punch breaks down. So he buys a high speed punch for $2,000.

7. He punches the binary tape.

8. He loads the binary tape.

9. He starts the program and types in the amount of loan, interest rate, and term.

10. Immediately, the system starts printing output, one line for each monthly payment. It takes a total of forty-five minutes to print all 360 lines. Meanwhile, the man stands there, with his fingers in his ears, hoping that the teletype printer will not break down before all the output has been printed.

The following chart compares the price/performance characteristics at the beginning and at the end of the example:

<table>
<thead>
<tr>
<th></th>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>Price</td>
<td>$0</td>
<td>$12,000</td>
</tr>
<tr>
<td>Speed</td>
<td>Infinite</td>
<td>10 char/sec.</td>
</tr>
</tbody>
</table>

Some may say that the example is an exaggeration. It may be, but I wonder if they have ever tried to generate an amortization schedule using a minicomputer in its "basic configuration".

The point is, that if one were to substitute zero-cost, infinitely-fast processors into most existing minicomputer systems, the total system cost and overall system throughput would not be significantly affected.

A CALL FOR UNITY

So far, I have tried to make three points:

1. Computers should be designed for the user, not for the designers. The user wants a system to solve his problems, not a computer to program.

2. The best way to optimize the overall performance of a system is to take a unified approach in the design of the system's components in order to optimize their performance together.
3. I/O is by far the weakest link in current minicomputers. The total cost and overall performance of most existing minicomputer systems would not be greatly affected if we substituted a zero-cost, infinitely-fast processor and a super-powerful programming language.

The conclusion I draw from these points is that if we are to improve the overall performance of minicomputers, we must concentrate more on I/O. However, I don't think that faster, cheaper I/O devices are the whole answer. There is no question that we need such devices, but we need something more.

We need to include I/O in the design process from preliminary specification through actual construction. I/O is an integral part of system performance and it should be an integral part of the design process. Processor architecture, instruction set, and I/O scheme should be developed together, from scratch, in order to truly optimize total system performance.

I don't think we should discuss minicomputer I/O as an isolated topic; rather it should be treated as an integral part of the whole system. As soon as we look at I/O in this light, several very interesting possibilities appear.

A BIT OF PHILOSOPHY

Before we approach the problem of new I/O schemes, let us approach the problem of approaching problems. I think that we often misdirect our efforts due to taking too narrow a view of a given problem. It's like struggling to climb over a wall when, if we had stepped back and looked at the whole scene, we would have seen the open gate a short distance away.

The important thing is to define the real problem (in this case, to get to the other side of the wall, not to climb over the wall) and to take a sufficiently broad view of the problem so as to include several alternative paths from which to select the best.

Don't look for a way to improve existing methods. Rather, carefully define the real problem, then try to find the best way to solve that problem. The best solution may be to improve upon existing methods, but then it may be a totally different approach.

Rapid advances in technology necessitate constant reevaluation of goals and methods. Decisions which were valid two years ago may have lost their validity due to technological developments.

Let us try to reanalyze some of the basic characteristics of I/O and perhaps suggest some new approaches to minicomputer I/O design in the light of current (and projected) technology.

A VERY BASIC DISTINCTION

I/O operations can be divided into two groups:

1. Those which are intrinsic to the solution of the problem at hand, and
2. Those which are incidental to the solution.

Let us analyze the loan amortization problem discussed earlier, and classify the I/O operations performed according to the above criteria. The problem, as you recall, was to generate a loan amortization schedule (not to program a computer to generate the schedule. The difference here is important as will be seen).

The I/O operations involved are classified as follows:

Intrinsic
1. Input loan description
2. Output amortization schedule

Incidental
1. Load compiler
2. Type program
3. Punch object tape
4. Load program

Note that the division would have been quite different if the problem had been defined in terms of programming a computer to generate the schedule. Unfortunately, we “computer types” have grown so accustomed to this rigmarole that we accept it as a part of problem solving. It is difficult for us to distinguish between the two because we are so used to working with machines. (If you have a hard time categorizing the I/O steps in a particular application, try describing the sequence of operations to your wife. Those operations which she accepts and understands without further explanation are intrinsic, the others are incidental.)

The goal of new I/O design approaches should be to streamline the intrinsics and to eliminate the incindentals. If an incidental operation cannot be eliminated, it should be made transparent or at least as painless as possible.

COMPUTERS TALK TO PEOPLE

Until recently, man/minicomputer communications have been rather poor. The teletype has been by far the predominate minicomputer I/O device, primarily due to an unapproachably low cost for a combined keyboard, printer, tape punch, and tape reader facility.

Rather than ask how we can improve upon the teletype, let us ask “What is the best way to talk to
computers?" The answer is contained in the question. Most interpersonal information is conveyed by speech. Even "HAL", the ultimate computer talked and listened to people. ("Yes", you might say, "but look what happened to him.") Notice, however, that not a single teletype was to be seen (or heard) throughout the entire Space Odyssey.

Unfortunately, inexpensive spoken communication with minicomputers is not (yet) within the state of the art. So we must ask what is the next best method. Obviously it is visual communications.

Man can assimilate visual information very rapidly. Ten characters per second is much too slow, one hundred per second is adequate, and a picture is worth a thousand and twenty-four words. I think that we are on the right track with some of the low-cost CRT terminals which have been and are being developed. One objection which is usually raised about CRT output is the lack of hard copy. True, this may be a limitation in some instances, but how often do you really need hard copy? Suppose you could store scrolls of output in a file somewhere and call them back for CRT display and manipulation very rapidly? Again, the solution space is different for different statements of a problem.

Now, how should man talk to a computer? Remember, most new users will be non-technically oriented. We should attempt to tailor the computer to the people, not vice versa. Let the machine do the work. This is in keeping with the trend toward less expensive machines and more expensive people.

I firmly believe that the human finger is much better for pointing than for typing. Given a fast CRT output, a very efficient input method is the selection of a reply from a computer-generated "menu". Let the computer guide the user and help, rather than hinder, in the solution of his problems.

COMPUTERS ALSO TALK TO MACHINES

Peripheral device interfacing is the area where we have had more experience, since we have long been attacking such problems as "How can we make our machine talk to a teletype?" (instead of "How can we make our machine talk to the person sitting at the teletype?").

I think new developments in technology and new applications areas warrant a new look at the area of interfacing peripheral devices to minicomputers. I think we can find ways to design better device controllers, faster and at a much lower cost.

We spend most of our time trying to develop integrated processor/software systems. We take advantage of quantity production techniques to lower hardware costs. We do everything we can to minimize engineering and programming time for the basic system, then we design a unique controller (and write new support software) for each new peripheral device.

We are very interested in statistics concerning the amount of time our CPU's are busy, but do we realize how inefficiently our device controllers are used? Most integrated circuit devices can easily run at a ten megacycle clock rate. Yet an I.C. teletype interface runs on a 110 cps clock. A typical photoelectric reader reads 300 characters per second. This means that such device controllers are only active on the order of 0.001 percent of the time. The remaining 99.999 percent of the time, the high speed logic gates are idle and only a few flip-flops are needed to hold some logical state information.

To me, this clearly suggests multiplexing, or in some way time-sharing the control logic among several devices.

PARTITIONING OF I/O FUNCTIONS

The inclusion of I/O design as an intrinsic part of the overall computer system design provides a much larger space over which to distribute the functions necessary for I/O operations.

For example, we could choose to implement a full duplex teletype controller using only one flip-flop, a clock, and two level converters, and provide timing and control functions in software.

To add a photoreader merely requires device addressing capability, perhaps another flip-flop, and an addition to the software.

This argument begins to fall apart when we add too many devices, or hang on a fast device (such as a magnetic tape unit). But does it really fall apart? How much I/O could your minicomputer handle if all it had to do was I/O? How much more could it handle if the I/O routines were in read-only-memory, rather than in core? Minicomputers are commonly being used to handle I/O for larger machines.

"But", you say, "separate I/O processors are only warranted for very large machines. They are much too expensive to be included in a minicomputer."

A WAY-OUT IDEA(?)

Let us consider this approach before we dismiss it as unrealistic. Suppose we were designing a minicomputer as an integrated CPU/software/I/O system. We could