A Little Non-standard Intel format (BPNF)

Stanley Mazor

December 13 2011

Summary

In 1970 Intel developed a proprietary coding scheme for customers to transmit their orders for ROM chips. The formatting scheme was called **BPNF,** and its usage is explained here. Although it was an ad hoc specification, and not a generally important computer science element, it was an external interface that became a standard format for object code to be *burned* into EPROM chips..

Background

In 1968 Intel started as the first semiconductor memory company, in Mt. View, CA. The MOS technology offered higher density, but slower speed than conventional (bipolar) semiconductor technology, so Intel made two static memory IC chips, the fast 64-bit I3101, and the large 256-bit I1101. I joined as the Intel applications engineer working for Ted Hoff, the manager of Applications Research in 1969. There was effectively no market for semiconductor memories, and their advantage was in *small scale* applications since core memories did not *scale down* very effectively. Our job was to find applications and promote the use of our memory chips. As a computer designer I was aware of the need of scratch pad memories, page tables, and other small memories inside main frame computers, and also the use of Read Only Memory (ROM) as a control store to hold microcode. Accordingly Ted and I requested development of two other types of high speed memory chips for use in large computers: associative memory (for page tables), and ROM's. The associative memory chip was designed and produced and had an infinitesimal market. On the other hand, the bipolar ROM was somewhat successful. Below are some details about the *data format* developed for that product.

Read Only Memory

A read only memory chip contained a custom pattern as specified by the customer, and created by Intel using a custom *mask* that established each bit permanently as a 1 or a 0. Since most ROM's are used in a wide word, it was practical to organize a ROM chip wider than RAM chips—which were a one-bit wide *slice* of a memory. Consequently, Intel's bipolar ROM chip (I3301), was organized as 256 4-bit words. Since the data pattern is fixed in the IC factory, there is no need for an input data bus (as required in a RAM chip); and consequently this chip had 8-address pins, 4 data output pins, and 2 power pins. As 16-pins were available, two select pins were also provided. External address decoders used these to *enable* particular chips within a large array of ROM chips. Unlike RAM chips there is a need for closer communication between the vendor and customer; it is imperative to have both unambiguous and error free transmission of the customer's data patterns. To that end I developed a proprietary data format that I named BPNF for reasons explained here.

Custom Data Communication

Regrettably there is not common agreement on what voltage represents a logic 1 or 0. There are both positive and negative conventions in companies, because many used inverting logic components. So it would be ambiguous to have a customer specify a pattern containing one's and zero's. For that reason I adopted a notation referring to the electrical polarity of **p**ositive and **n**egative, with the meaning that the pattern corresponded to the voltage level. Bipolar chips typically ran on a 5-volt supply and ground, and the ground could also be more negative. So the P corresponded to a positive voltage (nearly 5 volts), and the N specified a low, zero or negative, voltage.

There is another issue with a ROM chip. One needs to define the polarity of the address lines, and also the order of the output bits. RAM chips are more forgiving, since any data line is interchangeable with any other data line; address line polarity doesn't matter; and chips are simply one-bit wide. For ordering a ROM chip, the customer needs an unambiguous way of specifying the address and the data-bit order for the 4-bit wide ROM chip output bus. Accordingly we developed a paper form for the customer to fill out to specify all 1024 data bits as being either a P or N.

In 1970 there weren't many ways for communicating between customer and factory, so the forms were either physically mailed or faxed to Intel. Frequently the data had to be transcribed as the customer usually had the patterns for his ROM (e.g., microcode) stored on a computer system. Another option was using paper tape, as it was a common and suitable transmission means for sending data that was computer stored.

Paper Tape

A paper tape could be generated automatically from a computer system without human intervention. The customer could then mail the physical tape to Intel; however, since most customers had Telex machines, it was more expedient to send a message (TWX) containing the data. In this case no form was used and the paper tape contained some indicative information for the customer's order such as a chip designation.

However, when you look at the ASCII encoding of the character one and the character zero, you'll notice that they are *adjacent* and somewhat error prone as a dropped bit or picked bit can convert or invert the requested data. Thus instead of sending a zero or one character, we used the letters P and N, as they are *not* a single bit-change apart. A dropped or picked bit results in an invalid character. For additional redundancy I chose to put *brackets* around each 4-bit word, namely the letter **B** for begin and the letter **F** or Finish. Thus the Intel proprietary format used 6 characters for each 4-bit data word. For example, a typical data word looked like

BPPPPF (1111) or BNNPNF (0010)

Again for redundancy, each word was prefixed with the memory address, separated by a punctuation mark; and the addresses were always in sequence from 0 to 255, in decimal ASCII. There were no other parity or longitudinal data checks. The customer needed to specify all 256 data words, in the correct order, with no gaps, and with all data present.

Conclusion

As a semiconductor memory company, Intel made both RAM and ROM IC chips, and the making of custom ROM chips required transmission of data patterns using the available infrastructure. We adopted paper tape transmission, and developed a proprietary data representation for efficient and error reducing data patterns. As Intel developed software *assemblers and compilers* that produced binary object code, we chose the BPNF format as the output format to feed into *EPROM programmers* and to represent data patterns in external media. I understand that the BPNF format evolved for other uses, but it humbly started with the *need* for sending custom ROM data.

Electronic data communications and standards have evolved considerably since then.