A Microprocessor for

Part 3:

Terry Ritter and Joel Boney
Motorola Inc
3501 Ed Bluestein Blvd
Austin TX 78721

Photo 1: Processing. Photosensitized wafers are exposed with a particular mask pattern using ultraviolet light. The entire environment is otherwise ultraviolet-free.

Photo 2: Breadboard debug. The gate level TTL model of the processor involves ten boards of 80 to 120 integrated circuits each. Many of the required 10,000 connections will be wrong. The system must be tested to find and correct construction and logic errors.

Crowds are not unusual; here we have Don Tietjen, Katy Miller, James Tietjen, Steve Messinger (almost hidden), Mike Shapiro and Bill Keshlear.

Clock Speed

In part 3 we conclude our discussion of the Motorola 6809 processor with some thoughts on clock speed, timing signals, condition codes and software design philosophy for the 6809.

We expect that our logic and circuit design cohorts will be able to get significant production at a 2 MHz bus rate (and possibly faster) with the 6809. But this value alone means next to nothing as a figure of processor merit (we did consider using a very high frequency on chip oscillator so we could win the clock rate race, but decided at the last minute that a resonant cavity would not be acceptable to most users).

Other processors use an internal state machine to implement the required internal operations. These processors frequently require multiple states and multiple clock edges to implement operations which are done in one cycle on 6800 class processors.

The 6800 class machines are all random logic machines with multiple dynamic sequencers. This method of microprocessor design selects a different set of engineering trade-offs as opposed to the state machine approach. In particular, less critical timing is necessary, but suspending the processor for a long time is difficult. We provide two external methods of stopping the machine: $\overline{\text{DMAREQ}}$ (which has a maximum asynchronous latency of 1.5 bus cycles, and which will recover the bus from DMA (direct memory access) periodically to allow the dynamic microprocessor to perform a refresh cycle) and $\overline{\text{HALT}}$ (which has a maximum latency of 21 cycles, but releases this bus completely).

Signals

The 6809 processor will be made in two versions: the on chip clock version (for small systems) and the off chip clock version (with extra signal lines for additional processor status information). This will allow a cost-effective utilization of pins for each proposed market.

The bus timing signals are E and Q. E is

the Revolution: The 6809

Copyright 1978 by Terry Ritter
and Joel Boney

Final Thoughts

the same as on 6800 systems (previously called Ø2), a square wave clock with a period equal to one bus cycle. Q is the quadrature clock, and leads E by one quarter bus cycle. Good addresses should be available from the processor on the leading edge of Q; data is latched (by the processor or selected memory or peripheral) on the trailing edge of E.

Two signals are used for clock control in the on chip clock version. $\overline{\text{DMAREQ}}$ halts the processor internally (and puts the output lines of the processor in the high impedance state using three state circuitry) but allows E and Q to continue to run to provide system clocks for a DMA transfer. MREADY being low extends a memory access in increments of the high frequency oscillator period until MREADY is brought high.

If BA=0 (the processor is running) BS=1 means that a vector fetch is occurring (IACK). This signal can be used to develop vector-by-interrupting-device hardware that transfers control directly into the desired interrupt handler without polling.

Two signals are available in the off chip clock version to assist in multiprocessor systems. The last instruction cycle (LIC) pin is high during the last execution cycle of any instruction, thus giving bus arbitration a head start. BUSY is high during read modify write, (from the read through to modify) to indicate that memory exclusion is required. Exclusion is required in multiprocessor systems.

Condition Codes

The 6809 condition code flags are the same as those used in the 6800 (N, Z, V and C), and are affected similarly by most operations. Some exceptions are the double byte operations, since the flags are always set to represent the result of the entire operation, whether single or double byte. (This is implied by the fact that both data length operations have the same root mnemonics).

While very simple in concept (the condition flags being mere by-products of arithmetic and logic unit [ALU] operations), their use with various data representations

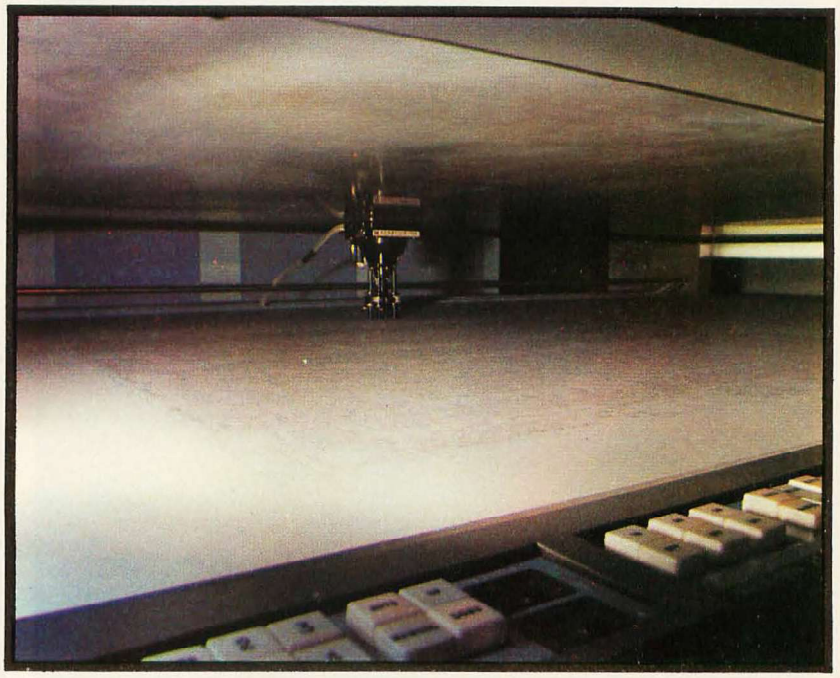


Photo 3: Plotting the circuit layout. Huge precision plotters display the computer data base which will become the chip. The layout plot is then checked by circuit engineers both for proper interconnection and exact transistor sizing. Any problems thus uncovered will be repaired by editing the data base.

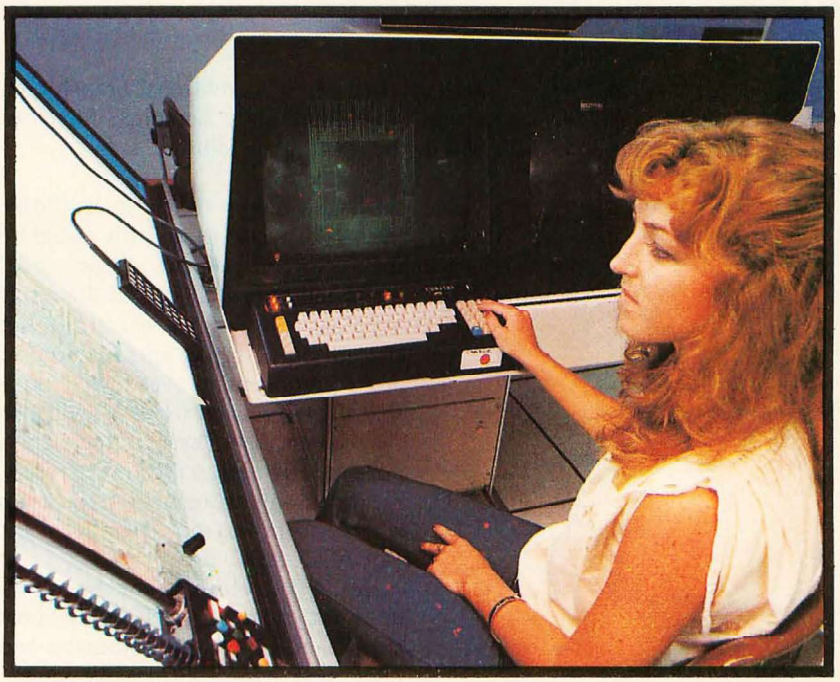


Photo 4: Digitizing. Computer aided design (CAD) technician Lisa Fink enters a cell layout into the data base. The cursor on the light table is used to transfer precision measurements to the computers. An already digitized cell is shown on the video display.

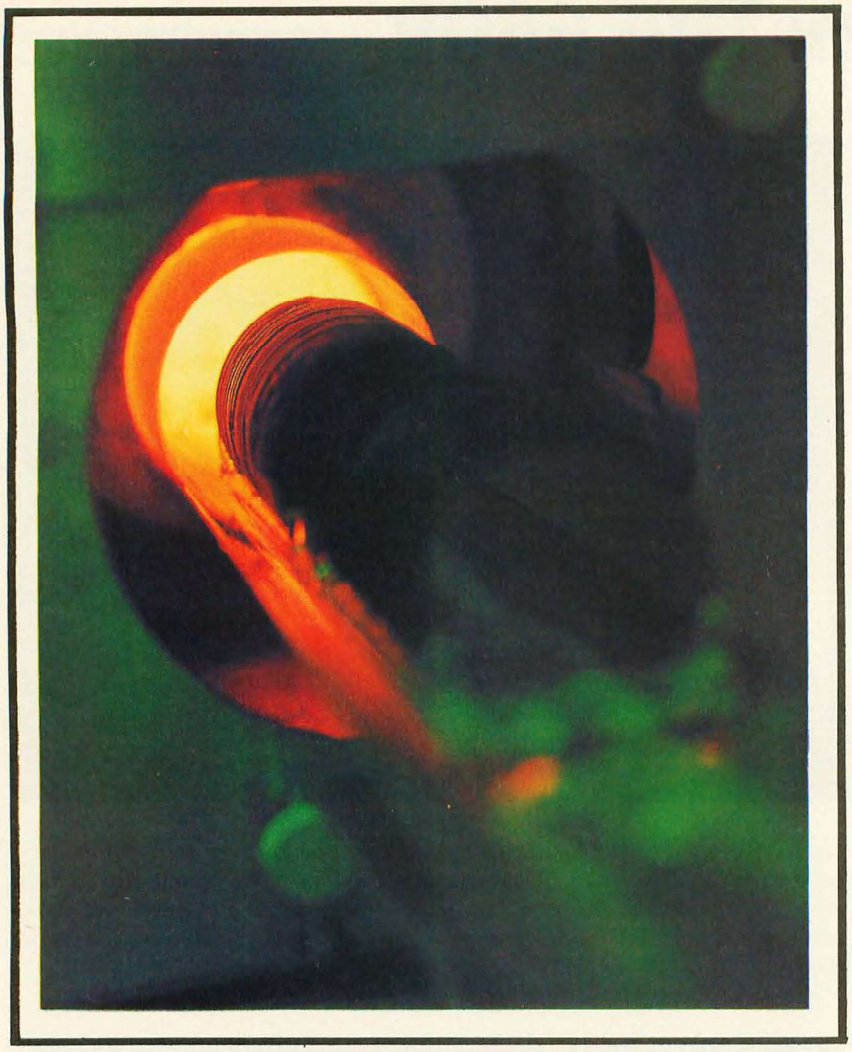


Photo 5: Diffusion. Into the furnace goes another batch of wafers in the process of becoming integrated circuits. Operating near 1000° C, the quartz liner glows incandescent.

and the rich set of conditional branch conditions can seem quite complex. First, we will define the flags as follows.

N: set if and only if the most significant bit of the result is set (this would be the 2's complement "sign" bit).

Z: set if and only if all bits of the result are clear (the result is exactly 0). set

V: if and only if the operation causes a 2's complement overflow. Notice that the expression (N ⊕ V) will give the correct sign, even if the sign is not properly represented in the result.

C: set if and only if the operation causes a carry from the most significant bit (for ADD, ADC) *or*,

set if and only if the operation does not cause a carry from the most significant bit of the arithmetic and logic unit (for subtract-like operations — SUB, SBC, CMP — carry flag represents a borrow) *or*,

set according to rules for rotate or shifts *or*,

set if and only if bit 7 of the result is set (for MUL).

- Notice that the C flag is not the simple result of the carry in the 8 bit arithmetic and logic unit, but depends on the type of operation performed.
- Notice also that the carry flag represents a borrow after subtract-like operations. This was done on the 6800, for convenience.

Next, let's define the use of the branches. Simple conditional branches:

Test	True	False
Z=1	BEQ	BNE
N=1	BMI	BPL
C=1	BCS	BCC
V=1	BVS	BVC

Signed conditional branches:

Test	True	False
$\overline{(N \oplus V)} \wedge \overline{Z}=1$	BGT	BLE
$\overline{(N \oplus V)} = 1$	BGE	BLT
Z=1	BEQ	BNE
$\overline{(N \oplus V)} \vee Z=1$	BLE	BGT
$\overline{(N \oplus V)} = 1$	BLT	BGE

Unsigned conditional branches:

Test	True	False
$\overline{C} \wedge \overline{Z}=1$	BHI	BLS
$\overline{C}=1$	BHS	BLO
$\overline{Z}=1$	BEQ	BNE
$C \vee Z=1$	BLS	BHI
C=1	BLO	BHS

Note: The unsigned branches are not, in general, useful after INC, DEC, LD, ST, TST, CLR or COM.

And finally, the flag results of known conditions of comparison are as follows.

After SUB, SBC, CMP:

If register is less than memory value (2's complement values) $(N \oplus V)=1$.
If register is lower than memory value (unsigned values) C=1
If register is equal to memory value (signed or unsigned) Z=1.

Because some instructions do not (and should not) affect carry, only the equal and not equal branch tests (BEQ, BNE) are useful after these instructions (INC, DEC, LD, ST, TST, CLR, COM) operate on unsigned values. When operating on 2's complement

values, all signed branches are correctly available.

Some Software Design Philosophy

The design of successful software differs from other types of engineering design in that good software can be easily changed, but is exceedingly unforgiving. The creation of working software involves intimate contact with *quality*.

Any program, working or unworking, is a representative of the philosophy of *truth*; the machine will execute the program, good or bad. Only applicable programs are useful, however, and utility is where we encounter quality. Many individuals indoctrinated into a society founded upon truth can scarcely understand why such truthful programs do not work, for isn't one truth just as good as another?

Any program that is to be fixed or changed must be analyzed: the written code must be read and understood. Reading is a problem — most computer languages are very difficult to read simply because so many options are possible from each statement. Finding the coherent design of a program is nearly impossible when, as it is being read, thousands of options exist. It is the paradox of programming that a disciplined, restricted, structured programming language gives programmers greater freedom to understand their programs.

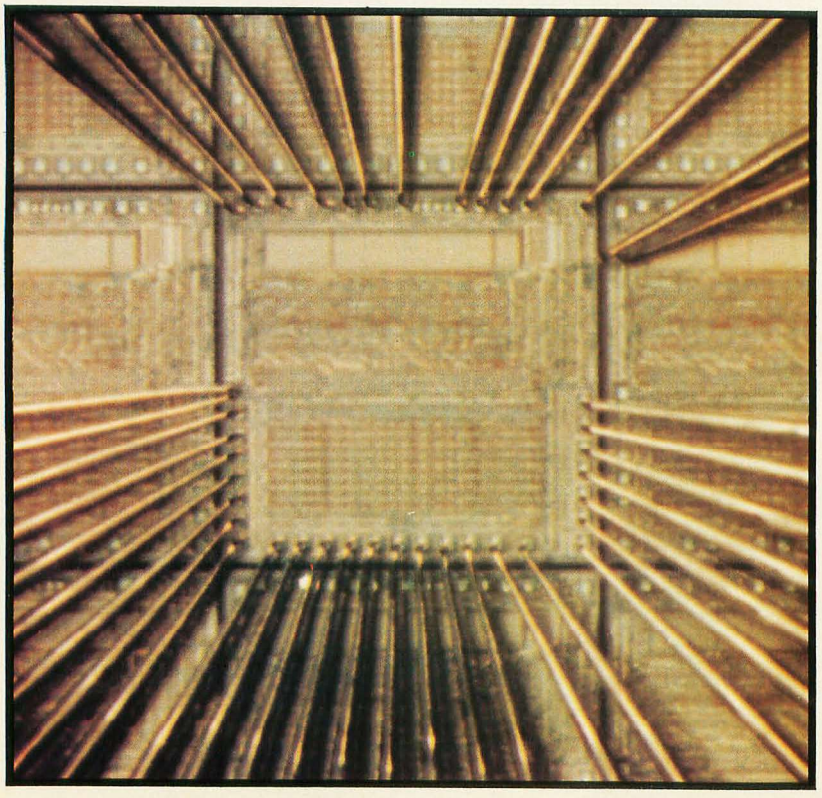
Consider the analysis of programs: any program segment having multiple conditional branches that cannot be separated must be analyzed for all possible conditions of input data before we can be assured that the program will operate correctly.

Program segments having branch paths that cross may be impossible to analyze rigorously due to the combinatorially larger number of paths that the program may execute. Where control structures are always properly nested, crossed branch paths cannot occur and analysis is easier.

Programming structures which have basically one entry point and one exit are easily detached from surrounding code and are easier to understand and test. This is the fundamental tenet of structured programming.

Every attempt should be made to code in modules. Modules are self-contained entities (usually subroutines) which allocate and deallocate their own local storage. Naturally, the actual code should be heavily commented to allow a reader to understand what is being attempted. But one mark of a good module is that it contains a header block which fully describes *all* aspects of the inputs to the module and results from it. This description should be so detailed as to allow the module to be totally recoded from this information alone. We hope that the description was arrived at *before* the module was written. It is a mark of good software design that the actual coding is but a minor part of the project; it occurs after all modules have been completely described. The finished modules should be individually tested for all possible input values, and should demonstrate that error handlers will operate when a supposedly invalid input value occurs. Modules which are recoded at a later date must pass the original tests.

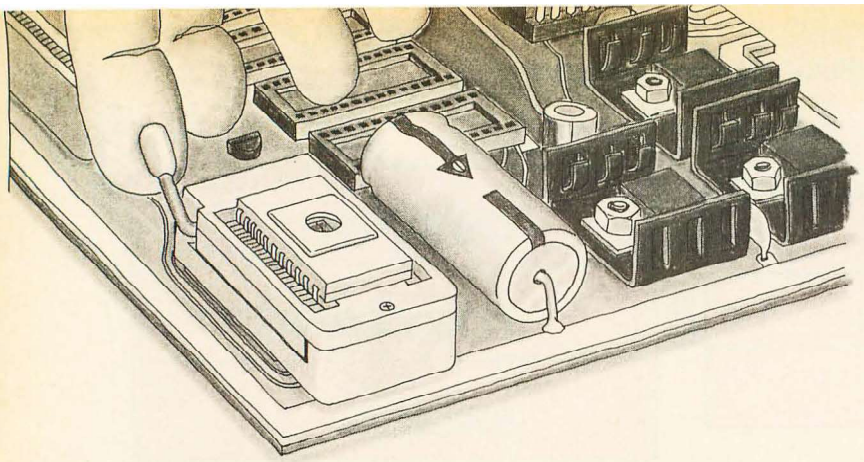
Photo 6: Wafer probe. Each circuit is separately checked while still on the wafer. This equipment automatically steps to the next chip after any bad result or when all tests are good. A production 6800 is shown.



Software in the Revolution

The microprocessor revolution is fueled by continual technical advancement that produces hardware with ever higher capability and ever lower cost. Yet, it is a requirement of the revolution that software be written to make that cheap hardware *do* anything.

Most present microprocessor software is custom software written for a specific project. Project specific software is rarely published, partly in the (unreasonable) hope of maintaining trade secret protection, and partly because finished project software is rarely of publication quality. Commercial software is rare for a number of reasons:



EPROMs out at the touch of a finger.

After programming a 2708 or 2716 EPROM you won't need a screwdriver to pry it out of **SSM's new PB1 board** equipped with Textool sockets. Just flip the lever and lift it out. And on the same board there are 4 sockets waiting for 2708 or 2716 EPROMs that can be independently addressed to any 4k or 8k boundary above 8000 hex. Two boards in one.

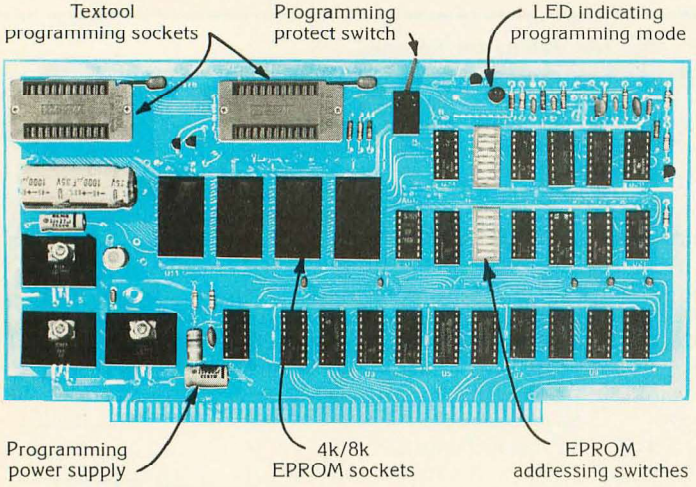
PB1 has two separate programming circuits so 2708 or 2716 (5v) type of EPROMs can be programmed without modifying the board. Programming voltage is generated on-board by a DC-DC converter; no need for an external power supply. Programming sockets are Dip Switch addressable to any 4k boundary. And complete software is provided for programming and verifying EPROMs.

With our Magic Mapping™ feature, unused EPROM sockets don't take memory space, so you are never committed to the full 4k or 8k of memory. The board can be configured for 0 to 4 wait states. Use fast or slow EPROMs. All lines are buffered.

The PB1 kit is available at over 150 retail locations or directly from SSM for $139.95 (with Textool sockets) or $119.95 (without Textool sockets). All SSM kits are backed by a 90 day warranty. Assembled, one year warranty.

SSM manufactures a full line of S-100 boards, including CPU, Video, I/O, RAM, EPROM, Music, Prototyping, Terminator, Extender and Mother boards. For complete details just send for our new, free brochure.

PB1 2708/2716 Programmer & 4k/8k EPROM Board



SSM
2116 Walsh Ave., Santa Clara, CA 95050
(408) 246-2707

We used to be Solid State Music. We still make the blue boards.

there must be a market for the (machine specific) software before the investment in program development is made, but the customer base may not exist until good programs are available. It is also difficult to consider investing in software that can be so easily copied (stolen) and used.

The copying problem is not new; musical reproductions have long coexisted with the possibility of consumer recording and reproduction for a close circle of friends. This occasionally happens, but it is usually too much bother to tape the music you want (assuming that the original product is available at a reasonable cost). Software should be distributed as a reasonably priced physical product that is useful to a broad consumer base.

This is an old idea, but it just hasn't worked. The problem is not in the idea, but in the second generation microcomputer architectures which limit the applicability of any particular program read only memory. The 6809 microprocessor is designed specifically — through the use of position independent code, stack indexing, and indirect addressing — to allow the creation of standard program read only memories. This creates a market opportunity for a brand new standard software industry. We knew this when we included these features; you're welcome, entrepreneurs!

Summary

We wrote this series of articles not only to disclose the 6809 but mainly to put down in print the rationale and reasoning behind the 6809. It would have benefited us if the designers of the 6800 had documented their rationale. We would also like to think we have stimulated some interest in the personal computing community for solutions to the software problem and for the study of computer architecture. The big challenge for architects in the next decade and beyond will be to design computers that can effectively utilize the huge number of devices — 1,000,000 transistors by 1985 — that semiconductor technology will be able to put on one 25 mm^2 piece of silicon.

No computer is designed in a vacuum, and we would like to thank all of our customers and the people at Motorola who gave us valuable input. Special thanks go to the dozens of people — too many to enumerate — who have been or are still actively involved in the design, implementation and production of the MC6809. Without their individual talents and dedication to what seemed to be impossible tasks and impossible schedules, the MC6809 could not have been realized.■

Circle 335 on inquiry card.