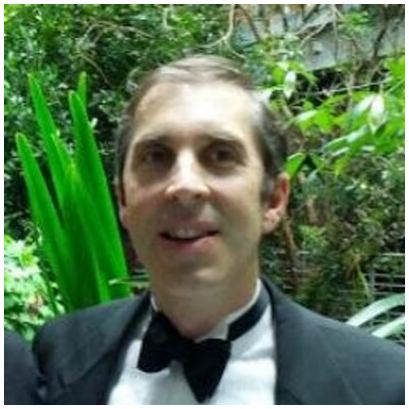
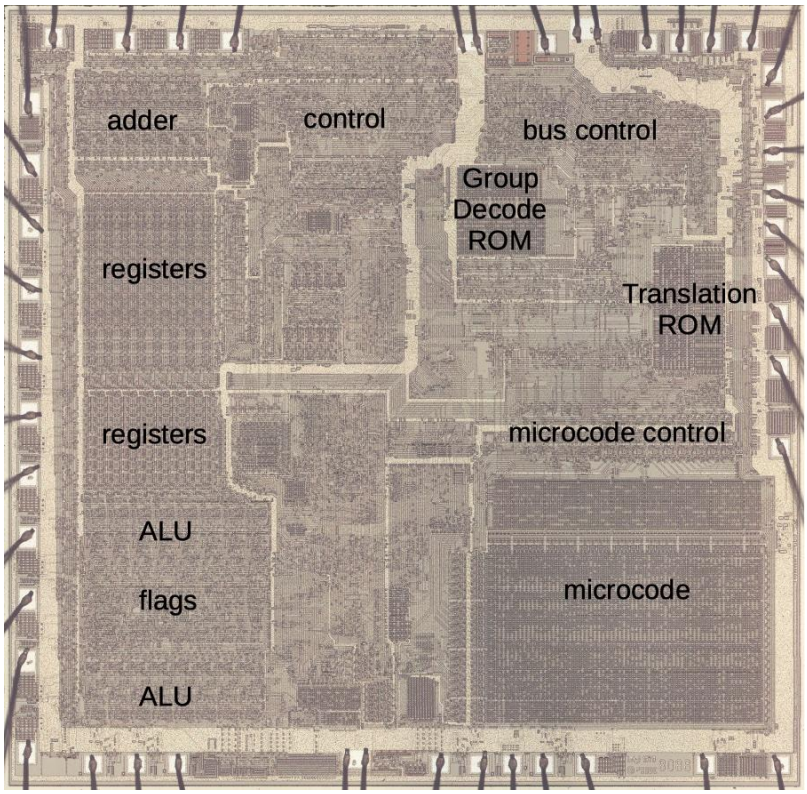


**Ken Shirriff** **@kenshirriff** *Sat Dec 03 19:18:34 +0000 2022*

The 8086 processor (1978) led to the hugely-popular x86 architecture. Internally, the 8086 uses microcode, running a tiny program for each machine instruction.

I'm reverse-engineering the chip from die photos and I can explain exactly how the 8086 microcode engine works.■ **https://t.co/coxeuvvJQS**




The idea of microcode is instead of implementing the processor's control circuitry with logic gates and circuits, you use another software layer. Microcode uses micro-instructions, simple instructions that generate hardware control signals and run in one step (clock cycle).

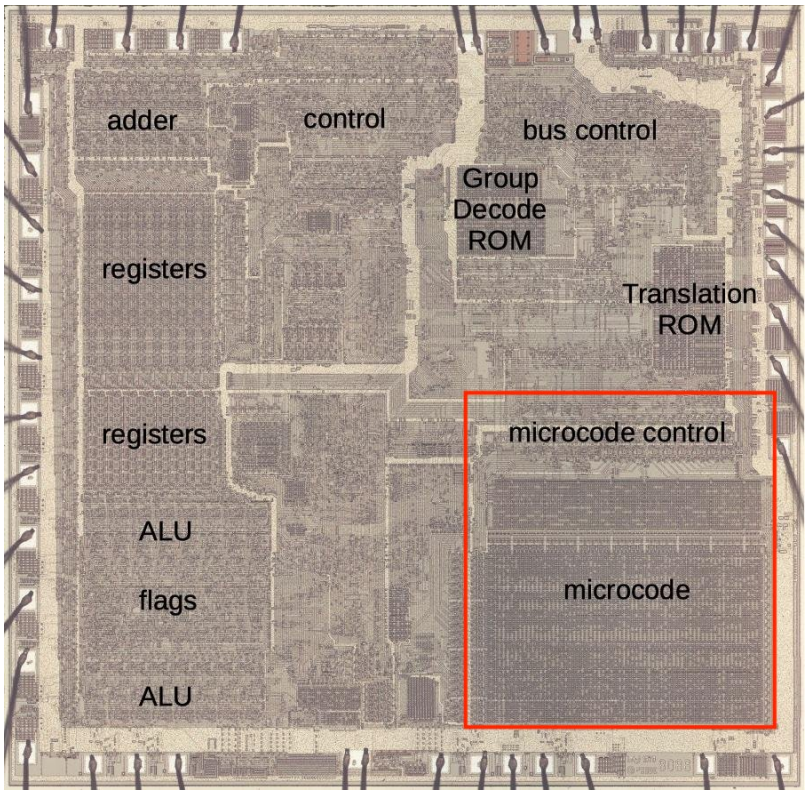
Here's part of the microcode for division. Each micro-instruction (yellow) moves a source register (S) to a destination (D). It also does an operation in parallel, e.g. subtraction (SUBT), left rotate (LRCY), conditional branch (NCY / no carry), (micro) subroutine return (RTN). **https://t.co/0YySEh9gSm**

| | X | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | CR | S | D | Typ | a | b | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CORD | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | | | 1 | SUBT | tmpa | |
| | | | | | | | | | | 1 | Σ | no dest | 4 | MAXC | none | F |
| | | | | | | | | | | 2 | | | 5 | NCY | 7 | |
| | | | | | | | | | | 3 | | | 1 | LRCY | tmpc | |
| | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 4 | Σ | tmpc | 1 | LRCY | tmpa | |
| | | | | | | | | | | 5 | Σ | tmpa | 1 | SUBT | tmpa | |
| | | | | | | | | | | 6 | | | 0 | CY | 13 | |
| | | | | | | | | | | 7 | Σ | no dest | | | | F |
| | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 8 | | | 0 | NCY | 14 | |
| | | | | | | | | | | 9 | | | 0 | NCZ | 3 | |
| | | | | | | | | | | 10 | | | 1 | LRCY | tmpc | |
| | | | | | | | | | | 11 | Σ | tmpc | | | | |
| | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 12 | Σ | no dest | 4 | none | RTN | |
| | | | | | | | | | | 13 | | | 4 | RCY | none | |
| | | | | | | | | | | 14 | Σ | tmpa | 0 | NCZ | 3 | |
| | | | | | | | | | | 15 | | | 0 | UNC | 10 | |

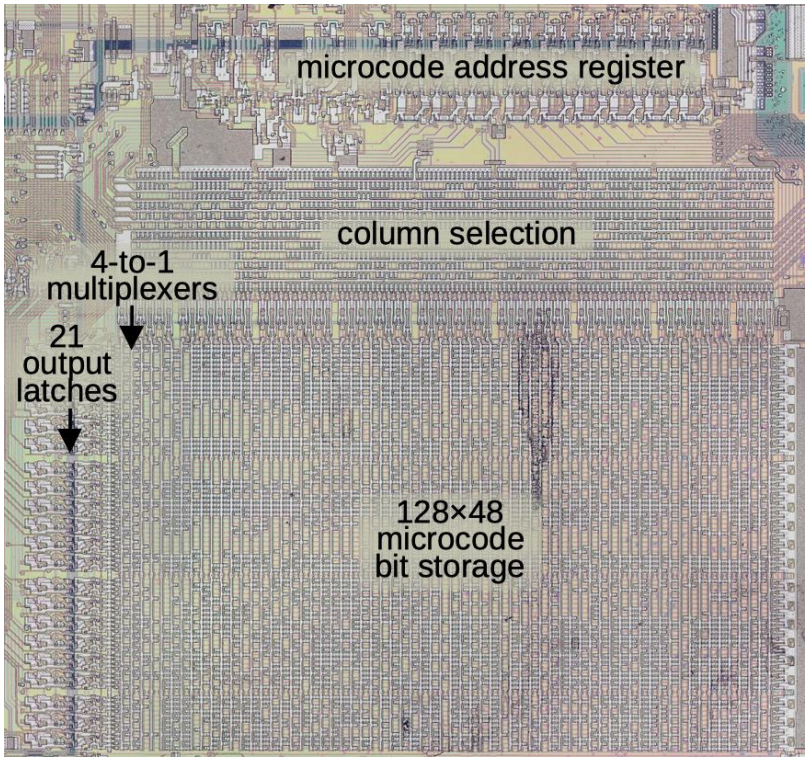
Each micro-instruction is packed into 21 bits. Decoding circuitry generates low-level control signals from it. There are 6 types of micro-instruction. The optimized encoding format depends on the type. (F controls whether the instruction sets the condition flag register.) **https://t.co/OSM9IISR2N**

| | | | Type | | a | | | | b | | | Flag |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Short jump | Source | Destination | 0 | 0 | Condition | | | | jump target | | | F |
| ALU operation | Source | Destination | 0 | 1 | ALU operation | | | | tmp | | NXT | F |
| Bookkeeping | Source | Destination | 1 | 0 | 0 | Operation 1 | | | Operation 2 | | | F |
| Memory read/write | Source | Destination | 1 | 1 | 0 | r/w | IAK | RNI | Seg Reg | Addr Factor | | F |
| Long jump | Source | Destination | 1 | 0 | 1 | Condition | | | target tag | | | F |
| Long call | Source | Destination | 1 | 1 | 1 | Condition | | | target tag | | | F |
| | 1 2 3 4 5 | 6 7 8 9 10 | 11 | 12 | 13 | 14 15 16 | | | 17 18 19 20 | | | 21 |

Mainframes started using microcode in the 1960s, but microcode took up too much space for the first microprocessors. The 8086 was the first Intel chip to use microcode (I think); microcode took up a large fraction of the chip. Lots of optimizations were needed to make it fit. **https://t.co/uh6Hw6hqHa**

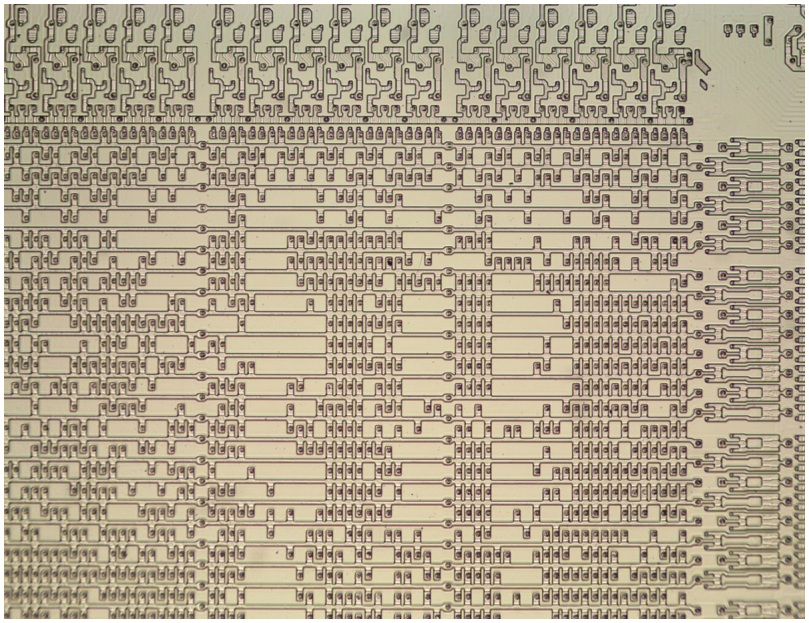


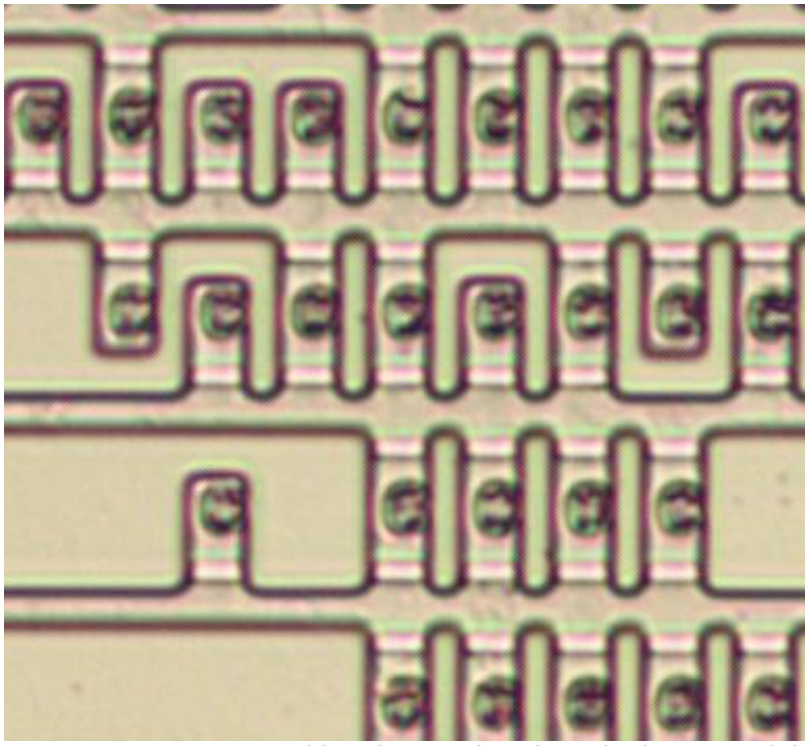

Here's a closeup of the microcode on the die. The address register holds a 13-bit micro-address. Column selection circuitry selects one column of the microcode ROM. There are 512 micro-instructions, stored four per column to improve the layout. **https://t.co/VOgdgLvZ5z**




Zooming in, you can see individual transistors in the microcode ROM. The silicon doping pattern defines the 0's and 1's. A few years ago, Andrew Jenner disassembled the 8086 microcode from my die photos. See his blog post to see the complete microcode.

**https://www.reenigne.org/blog/8086-microcode-disassembled/ https://t.co/JOpm6V9VrQ**





8086 supports many addressing modes through the second byte of instructions, the ModR/M byte. These are implemented by subroutines in the microcode. The "Translation ROM" (die photo below) holds the micro-addresses for subroutines. **https://t.co/9UeuZV2GMC**

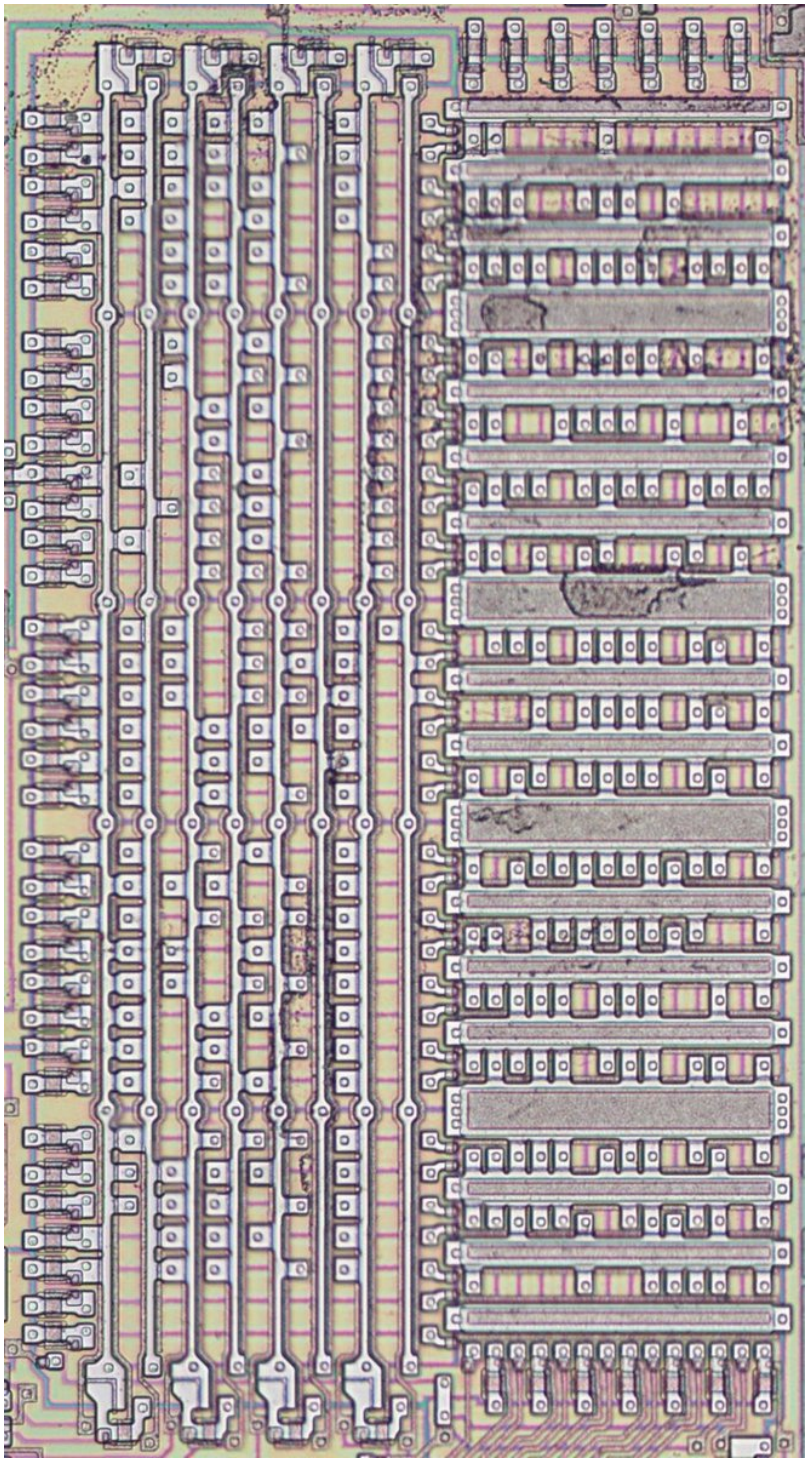
| mod | reg | r/m |
|---|---|---|

| mod | Displacement |
|---|---|
| 00 | DISP = 0*, disp-low and disp-high are absent |
| 01 | DISP = disp-low sign-extended to 16-bits, disp-high is absent |
| 10 | DISP = disp-high: disp-low |
| 11 | r/m is treated as a "reg" field |

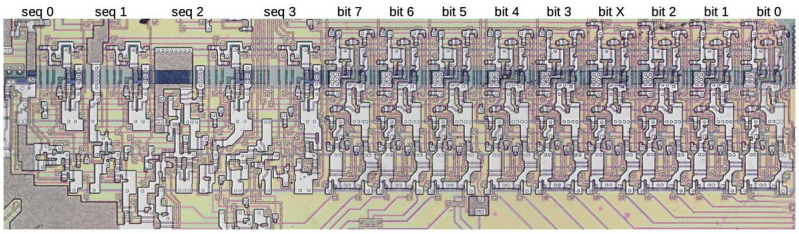
**"reg" field Bit Assignments:**

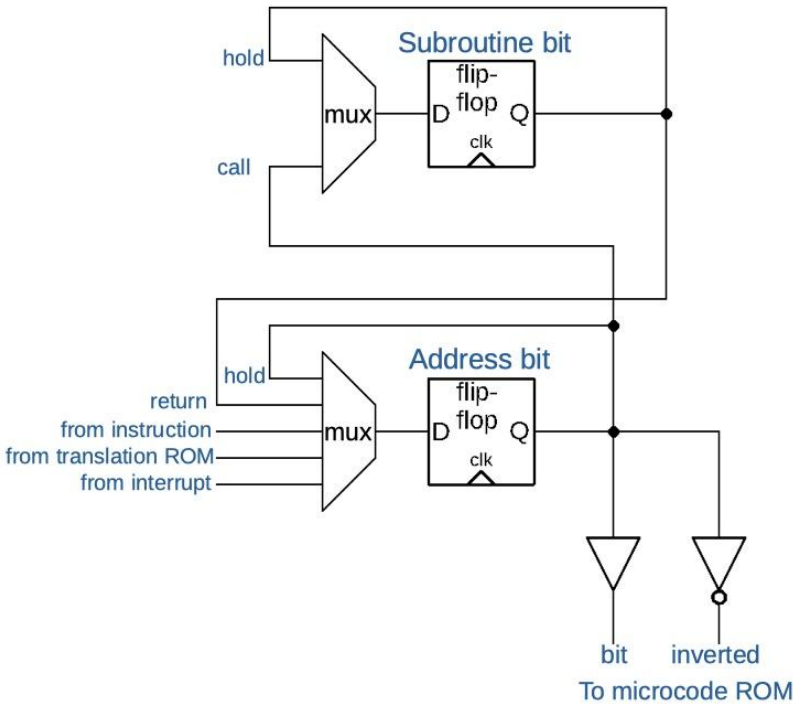
| 16-Bit (w = 1) | 8-Bit (w = 0) | Segment |
|---|---|---|
| 000 AX | 000 AL | 00 ES |
| 001 CX | 001 CL | 01 CS |
| 010 DX | 010 DL | 10 SS |
| 011 BX | 011 BL | 11 DS |
| 100 SP | 100 AH | |
| 101 BP | 101 CH | |
| 110 SI | 110 DH | |
| 111 DI | 111 BH | |

| r/m | Operand Address |
|---|---|
| 000 | (BX) + (SI) + DISP |
| 001 | (BX) + (DI) + DISP |
| 010 | (BP) + (SI) + DISP |
| 011 | (BP) + (DI) + DISP |
| 100 | (SI) + DISP |
| 101 | (DI) + DISP |
| 110 | (BP) + DISP* |
| 111 | (BX) + DISP |



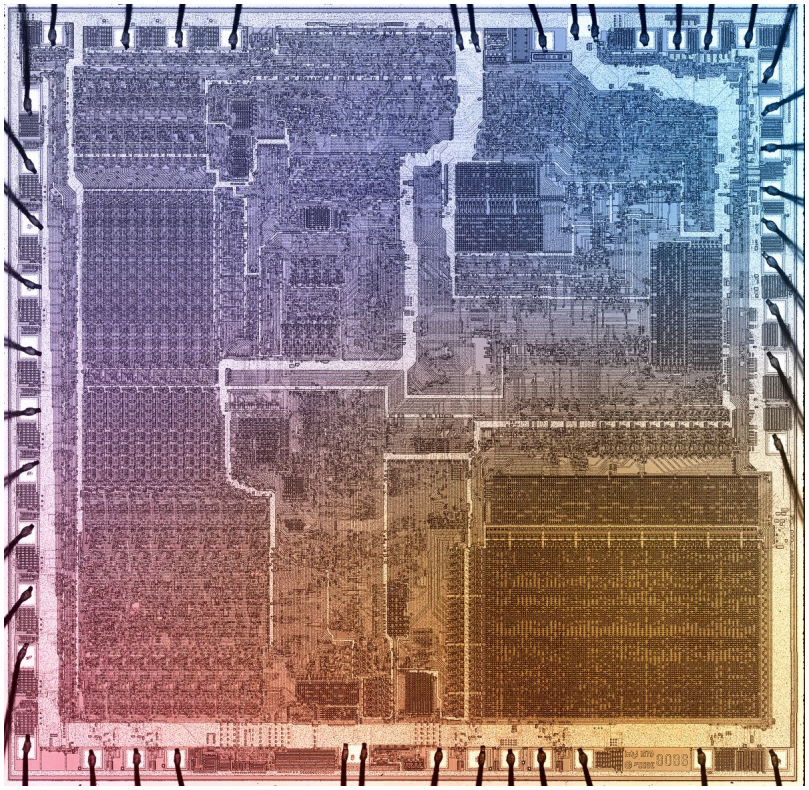
The microcode address register steps through microcode with a 13-bit address: 8 bits from the instruction, 4 sequential bits, and an extra X bit. It has a register to hold the subroutine return address. Multiplexers select the address for call/return/jump/interrupt/etc. **https://t.co/P096whrvrx**






To summarize, you might think that machine instructions are what processors execute. But microcode forms a lower layer: the processor runs multiple micro-instructions to implement each machine instruction. Microcode simplifies processor design making much of it software.

Here's a die photo of the 8086 using the color scheme from Apple's M1 chip. It amuses me to parody Apple's gradient by applying it to vintage chips. **https://t.co/EVFqrbtCCY**



For more details on 8086 microcode, see my blog post:
**https://www.righto.com/2022/11/how-8086-processors-microcode-engine.html**