COMP 122

Rev 10-4-23

ASSEMBLY Programming

x86 ISA

Dr Jeff Drobman

website → drjeffsoftware.com/classroom.html

email → jeffrey.drobman@csun.edu

Index

Computer Architecture

x86 Intro

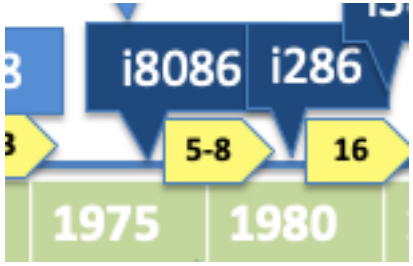
x86

“x86” refers to the ISA architecture that began back in 1978 with the i8086 16-bit microprocessor. it still is used to refer to that ISA through all the upgrades through Pentiums. Intel and AMD both now have 2 ISA’s: a 32-bit (IA-32) and a 64-bit (IA-64). most PC’s and all new Macs use x86–64 (IA-64), but Apple will replace x86–64 with ARMv8 (64-bit) cores.

x86 includes a 32-bit ISA and a 64-bit ISA, called "x86-32" and "x86-64". there are any number of assembly languages supporting those ISA's. "RISC" is a specific CPU architecture that all modern CPU's use. It has a goal of single-cycle execution via deep pipelines, a general register set, separate (Harvard style) L1 I and D caches, and an ISA that restricts memory access to "Load" and "Store".

x86-**32**

IA-32



x86-**64**

IA-64

x86

x86

Designer	Intel, AMD
Bits	16-bit, 32-bit and 64-bit
Introduced	1978 (16-bit), 1985 (32-bit), 2003 (64-bit) i286 i386
Design	CISC
Type	Register–memory
Encoding	Variable (1 to 15 bytes)
Branching	Condition code
Endianness	Little
Page size	8086–i286: None i386, i486: 4 KB pages P5 Pentium: added 4 MB pages (Legacy PAE: 4 KB→2 MB) x86-64: added 1 GB pages
Extensions	x87, IA-32, x86-64, MMX,

x86 is a family of instruction set architectures initially developed by Intel based on the Intel 8086 microprocessor and its 8088 variant. The 8086 was introduced in 1978 as a fully 16-bit extension of Intel's 8-bit 8080 microprocessor, with memory segmentation as a solution for addressing more memo

IBM PC

- Designed and built by IBM (Boca Raton, FL) in 1981
- CPU: **i8088** (i8086 with an 8-bit data bus)
- Speed: 4.77 MHz
- Memory – DRAM: 64KB (48KB soldered, 16KB socketed)
- Disk: 8 inch floppy – 2 drives (A:, B:)
- OS: MS/PC-DOS (derived from CP/M)

The IBM PC

x86 History

Heikki Kultala, M.Sc Computer Science, Tampere University of Technology (2010)

Answered 9h ago

32-bit is NOT called x86.

There are tens of 32-bit architectures such as MIPS, ARM, PowerPC, SPARC which are not called x86.

x86 is a term meaning any instruction set which derived from the instruction set of Intel 8086 processor. It's successors were named 80186, 80286, 80386, 80486, and were all compatible with the original 8086, capable of executing code made for it. Later Intel also released 8086-compatible processors named Pentium , Celeron, Core and Xeon but the name x86 had already stabilized to mean all processors base don the instruction set family.

Of these, 8086, 80186 and 80286 were 16-bit processors. 80386 was a 32-bit processor, with a new 32-bit operating mode. However, it still retained the original 16-bit mode and also added a thid mode, "virtual 86" mode which allowed running 16-bit programs under 32-bit operating system.

Later, 64-bit extension to x86, x86–64 was developed and implemented in AMD K8 and also later intels processors. Also these 64-bit processors based on the x86–64 architecture are called x86 processors

x86 History

So, the correct question is: Why is the 64-bit x86 called x64?

Over 10 years later, when the 64-bit extension to x86 instruction set was released, and Microsoft started porting later NT-derived windows to it, some official technical name had to be selected for the version compiled for this architecture. The specification came originally from amd, so some called it "amd64" whereas "intel64" had meant Itanium. But Microsoft did not want to include name of one company to the name they chose for the architecture, and also the name "x86–64" which is later used had not stabilized yet as the common name for the architecture, and also the dash character on "X86–64" name might problematic for some places where the architecture name appears and had to be parsed by some code. So they chose the name "x64", as 64-bit version of x86.

Even later, support for the Itanium architecture was dropped and support for 32- and 64-bit ARM architectures were added to Windows. The 64-bit ARMv8 is typically called either A64 or Aarch64, Im not sure which one is the official technical name for it in Windows.

So now, Windows has support for four architectures: 386 ("x86"), x86–64 ("x64), 32-bit ARMv7 , and 64-bit ARMv8.

So, "x64" currently only means one of these two 64-bit architectures currently supported by Windows.

x86 History

Wikipedia

In the past:

- Transmeta (discontinued its x86 line)
- Rise Technology (acquired by SiS, that sold its x86 (embedded) line to DM&P)
- IDT (Centaur Technology x86 division acquired by VIA)
- Cyrix (acquired by National Semiconductor)
- National Semiconductor (sold the x86 PC designs to VIA and later the x86 embedded designs to AMD)
- NexGen (acquired by AMD)
- Chips and Technologies (acquired by Intel)
- IBM (discontinued its own x86 line)
- UMC (discontinued its x86 line)
- NEC (discontinued its x86 line)

VIA

x86-processors for regular PCs

- Intel
- AMD
- VIA
  - Zhaoxin

x86-processors for embedded designs only [edit]

- DM&P Electronics (continues SiS' Vortex86 line)
- ZF Micro ZFx86,[1] Cx486DX SoC
- RDC Semiconductors[2] 486SX compatible RISC core (R8610 and R8620)
- ao486[3] open source FPGA implementation of the 486SX (currently targets the Terasic Altera DE2-115)
- S80186[4] open source 80186 compatible FPGA implementation
- Montage Jintide[5]

In the past:

- ALi (x86 products went to Nvidia through the ULi sale)
- Nvidia (M6117C - 386SX embedded microcontroller)
- SiS (sold its Vortex86 line to DM&P)
- Zet open source 80186 compatible FPGA implementation targeting the Xilinx ML403 and Altera DE1

x86

The x86 architectures were based on the Intel 8086 microprocessor chip, initially released in 1978.

Intel Core 2 Duo – an example of an x86-compatible, 64-bit multicore processor

AMD Athlon (early version) – a technically different but fully compatible x86 implementation

x86 Architecture

x86 CPU Models

x86 Evolution

4-bit* i4004

8-bit* i80**08** → i80**80** → i80**85** → 16-bit* i80**86**

16-bit address ⟶ **20-24** bit address

16-bit* i80**286** → 32-bit* i80**386** → i80**486** → i80**586**
→ *Pentium* (32/64-bit)

24-bit address ⟶ **40** bit address ⟶ **48** bit address

*word size is for DATA

[Note: the 1st gen IBM PC used a custom **i8088** with an 8-bit data bus]

AMD vs Intel: CPU Families

Market Segment	AMD	Intel
Desktop	Ryzen/Threadripper	Core (10th gen)
Laptop	Athlon	Ice Lake
Gaming	Threadripper +Radeon	Core Extreme
Server/Workstn	Epyc	Xeon

x86 Timeline

Processor	Series Nomenclature	Code Name	Production Date	Clock Rate	Socket	Fabrication	Bus Speed
4004	N/A	N/A	1971 - Nov 15	740 kHz	DIP	10-micron	N/A
8008	N/A	N/A	1972 - April	200 kHz - 800 kHz	DIP	10-micron	200 kHz
8080	N/A	N/A	1974 - April	2 MHz - 3.125 MHz	DIP	6-micron	2 MHz
8085	N/A	N/A	1976 - March	3 MHz, 5 MHz, 6 MHz	DIP	3-micron	2 MHz
8086	N/A	N/A	1978 - June 8	10 MHz, 8 MHz, 4.77 MHz	DIP	3-micron	10 MHz, 8 MHz, 4.77 MHz
8088	N/A	N/A	1979 - June	8 MHz, 4.77 MHz	DIP	3-micron	8 MHz, 4.77 MHz
80286	N/A	N/A	1982 - Feb	12 MHz, 10 MHz, 6 MHz	DLPP	1.5-micron	12 MHz, 10 MHz, 6 MHz
i80386	DX, SX, SL	N/A	1985 - 1990	33 MHz, 25 MHz, 20 MHz, 16 MHz	DLPP	1 - 1.5-micron	33 MHz, 25 MHz, 20 MHz, 16 MHz
i80486	DX, SX, DX2, DX4, SL	N/A	1989 - 1992	25 MHz - 100 MHz	Socket 1, Socket 2, Socket 3	1 - 0.6-micron	25 MHz - 50 MHz

x86 Timeline

Processor	Series Nomenclature	Code Name	Production Date		Clock Rate
Intel Pentium	N/A	P5, P54C, P54CTB, P54CS	1993 - 1999		65 MHz - 250 MHz
Intel Pentium MMX	N/A	P55C, Tillamook	1996 - 1999		120 MHz - 300 MHz
Intel Atom	Z5xx, Z6xx, N2xx, 2xx, 3xx, N4xx, D4xx, D5xx, N5xx, D2xxx, N2xxx	Diamondville, Pineview, Silverthorne, Lincroft, Cedarview, Medfield, Clover Trail	2008 - 2009 (as Centrino Atom) 2008–present (as Atom)		800 MHz - 2.13 GHz

x86 Timeline

Processor	Series Nomenclature	Code Name	Production Date	Clock Rate	
				Bus Clock	CPU Clock
Intel Pentium	N/A	P5, P54C, P54CTB, P54CS	1993 - 1999	50 MHz - 66 MHz	65 MHz - 250 MHz
Intel Pentium MMX	N/A	P55C, Tillamook	1996 - 1999	60 MHz - 66 MHz	120 MHz - 300 MHz
Intel Atom	Z5xx, Z6xx, N2xx, 2xx, 3xx, N4xx, D4xx, D5xx, N5xx, D2xxx, N2xxx	Diamondville, Pineview, Silverthorne, Lincroft, Cedarview, Medfield, Clover Trail	2008 - 2009 (as Centrino Atom) 2008–present (as Atom)	400 MHz, 533 MHz, 667 MHz, 2.5 GT/s	800 MHz - 2.13 GHz

x86 Timeline

Celeron

Intel Celeron	3xx, 4xx, 5xx	Banias, Cedar Mill, Conroe, Coppermine, Covington, Dothan, Mendocino, Northwood, Prescott, Tualatin, Willamette, Yonah, Merom, Penryn, Arrandale, Sandy Bridge, Ivy Bridge, Haswell, Broadwell, Bay Trail-M, Braswell, Skylake	1998–present

Xeon

Intel Xeon	n3xxx, n5xxx, n7xxx	Allendale, Cascades, Clovertown, Conroe, Cranford, Dempsey, Drake, Dunnington, Foster, Gainestown, Gallatin, Harpertown, Irwindale, Kentsfield, Nocona, Paxville, Potomac, Prestonia, Sossaman, Tanner, Tigerton, Tulsa, Wolfdale, Woodcrest	1998–present

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE

COMP122

x86 Timeline

DR JEFF SOFTWARE
INDIE APP DEVELOPER

i3/i5

Intel Core i3	i3-xxx, i3-2xxx, i3-3xxx, i3-4xxx, i3-61xx, i3-63xx	Arrandale, Clarkdale, Sandy Bridge, Ivy Bridge, Haswell, Skylake, Kaby Lake, Coffee Lake	2010–present
Intel Core i5	i5-7xx, i5-6xx, i5-2xxx, i5-3xxx, i5-4xxx, i5-64xx, i5-65xx, i5-66xx, i5-74xx, i5-75xx, i5-76xx, i5-84xx, i5-85xx, i5-86xx	Arrandale, Clarkdale, Clarksfield, Lynnfield, Sandy Bridge, Ivy Bridge, Haswell, Broadwell, Skylake, Kaby Lake, Coffee Lake	2009–present

i7

Intel Core i7	i7-6xx, i7-7xx, i7-8xx, i7-9xx, i7-2xxx, i7-37xx, i7-38xx, i7-47xx, i7-48xx, i7-58xx, i7-59xx, i7-67xx, i7-68xx, i7-69xx, i7-7700K	Bloomfield, Nehalem, Clarksfield, Clarksfield XM, Lynnfield, Sandy Bridge, Sandy Bridge-E, Ivy Bridge, Ivy Bridge-E, Haswell, Haswell Refresh, Devil's Canyon, Broadwell, Skylake, Kaby Lake, Coffee Lake	2008–present

M

Pentium M	7xx	Banias, Dothan	2003 - 2008

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP122

x86 Timeline Generations

Chronology of x86 Processors

Generation		Introduction	Prominent CPU models	Address space			Notable features
				Linear	Virtual	Physical	
x86	1st	1978	Intel 8086, Intel 8088(1979)	16-bit	NA	20-bit	16-bit ISA, IBM PC (8088), IBM PC/XT (8088)
		1982	Intel 80186, Intel 80188 NEC V20/V30(1983)				8086-2 ISA, embedded (80186/80188)
	2nd		Intel 80286 and clones		30-bit	24-bit	protected mode, IBM PC XT 286, IBM PC AT
	3rd (IA-32)	1985	Intel 80386, AMD Am386 (1991)	32-bit	46-bit	32-bit	32-bit ISA, paging, IBM PS/2
	4th (pipelining, cache)	1989	Intel 80486 Cyrix Cx486S/DLC(1992) AMD Am486(1993)/Am5x86(1995)				pipelining, on-die x87 FPU (486DX), on-die cache
	5th (Superscalar)	1993	Intel Pentium, Pentium MMX(1996)				Superscalar, 64-bit databus, faster FPU, MMX (Pentium MMX), APIC, SMP
		1994	NexGen Nx586 AMD 5k86/K5 (1996)				Discrete microarchitecture (μ-op translation)
		1995	Cyrix Cx5x86 Cyrix 6x86/MX(1997)/MII(1998)				dynamic execution
	6th (PAE, μ-op translation)	1995	Intel Pentium Pro			36-bit (PAE)	μ-op translation, conditional move instructions, dynamic execution, speculative execution, 3-way x86 superscalar, superscalar FPU, PAE, on-chip L2 cache
		1997	Intel Pentium II, Pentium III (1999) Celeron(1998), Xeon(1998)				on-package (Pentium II) or on-die (Celeron) L2 Cache, SSE (Pentium III), SLOT 1, Socket 370 or SLOT 2 (Xeon)
		1997	AMD K6/K6-2(1998)/K6-III(1999)			32-bit	3DNow!, 3-level cache system (K6-III)
	Enhanced Platform	1999	AMD Athlon, Athlon XP/MP(2001) Duron(2000), Sempron(2004)			36-bit	MMX+, 3DNow!+, double-pumped bus, Slot A or Socket A
		2000	Transmeta Crusoe			32-bit	CMS powered x86 platform processor, VLIW-128 core, on-die memory controller, on-die PCI bridge logic

x86-64 Timeline Generations

x86-64	64-bit Extended since 2001	2011	AMD APU C, E and Z Series (Bobcat)	36-bit	low power smart device APU
			Intel Core i3, Core i5 and Core i7 (Sandy Bridge/Ivy Bridge)		Internal Ring connection, decoded μ-op cache, LGA 1155 socket.
		2012	AMD APU A Series (Bulldozer, Trinity and later)	48-bit	AVX, Bulldozer based APU, Socket FM2 or Socket FM2+
			Intel Xeon Phi (Knights Corner)	48-bit	coprocessor OS powered PCI-E Card Formed coprocessor for XEON based system, Many Core Chip, In-order P54C, very wide VPU (512-bit SSE), LRBni instructions (8× 64-bit)
		2013	AMD Jaguar (Athlon, Sempron)	48-bit	SoC, game console and low power smart device processor
			Intel Silvermont (Atom, Celeron, Pentium)	36-bit	SoC, low/ultra-low power smart device processor
			Intel Core i3, Core i5 and Core i7 (Haswell/Broadwell)	39-bit	AVX2, FMA3, TSX, BMI1, and BMI2 instructions, LGA 1150 socket
		2015	Intel Broadwell-U (Intel Core i3, Core i5, Core i7, Core M, Pentium, Celeron)		SoC, on-chip Broadwell-U PCH-LP (Multi-chip module)
		2015/2016	Intel Skylake/Kaby Lake/Cannon Lake (Intel Core i3, Core i5, Core i7)	46-bit	AVX-512 (restricted to Cannon Lake-U and workstation/server variants of Skylake)
		2016	Intel Xeon Phi (Knights Landing)	48-bit	Many-core CPU and coprocessor for Xeon systems, Airmont (Atom) core based
		2016	AMD Bristol Ridge (AMD (Pro) A6/A8/A10/A12)	48-bit	Integrated FCH on die, SoC, AM4 socket
		2017	AMD Ryzen Series/AMD Epyc Series		AMD's implementation of SMT, on-chip multiple dies.
		2017	Zhaoxin WuDaoKou (KX-5000, KH-20000)		Zhaoxin's first brand new x86-64 architecture
		2018/2019	Intel Sunny Cove (Ice Lake-U and Y)		Intel's first implementation of AVX-512 for the consumer segment. Addition of Vector Neural Network Instructions
Software Emulation					Cooperation between Microsoft and Qualcomm bringing Windows 10 onto ARM64 platform with

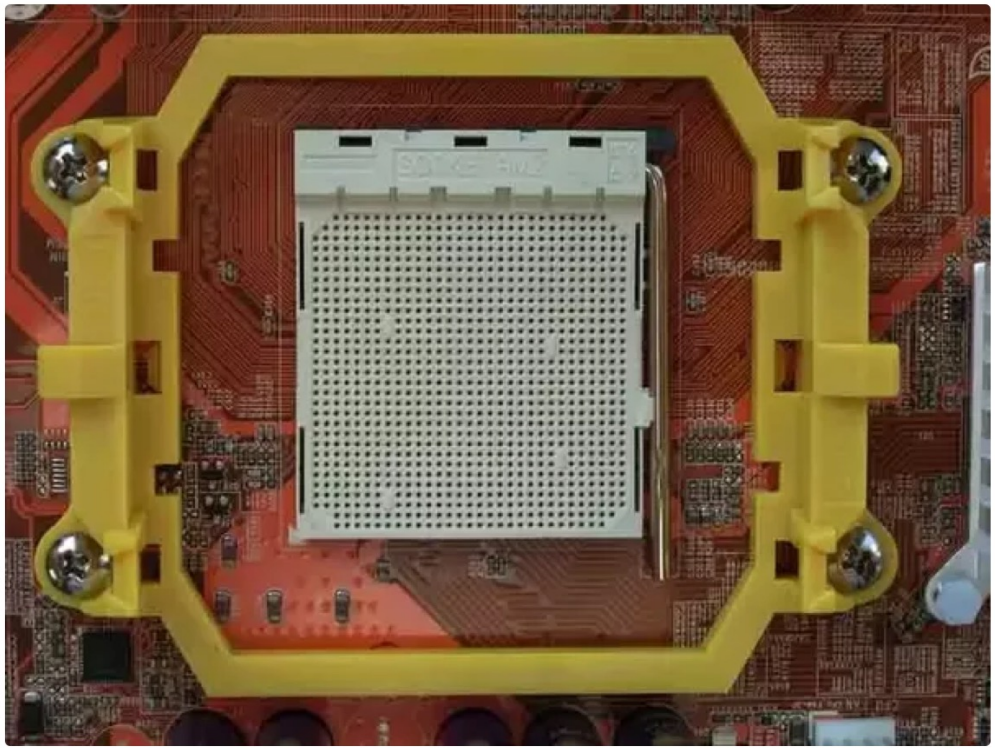
x86 Support Chips

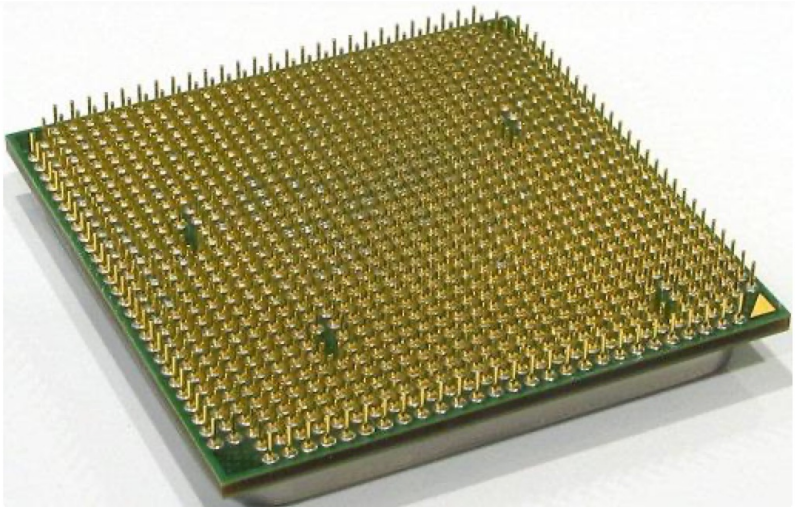
Support chips [edit]

- Intel 8237: direct memory access (DMA) controller
- Intel 8251: universal synchronous/asynchronous receiver/transmitter at 19.2 kbit/s
- Intel 8253: programmable interval timer, 3x 16-bit max 10 MHz
- Intel 8255: programmable peripheral interface, 3x 8-bit I/O pins used for printer connection etc.
- Intel 8259: programmable interrupt controller
- Intel 8279: keyboard/display controller, scans a keyboard matrix and display matrix like 7-seg
- Intel 8282/8283: 8-bit latch
- Intel 8284: clock generator
- Intel 8286/8287: bidirectional 8-bit driver. In 1980 both Intel I8286/I8287 (industrial grade) version
- Intel 8288: bus controller
- Intel 8289: bus arbiter
- NEC μPD765 or Intel 8272A: floppy controller[20]

AMD Packaging

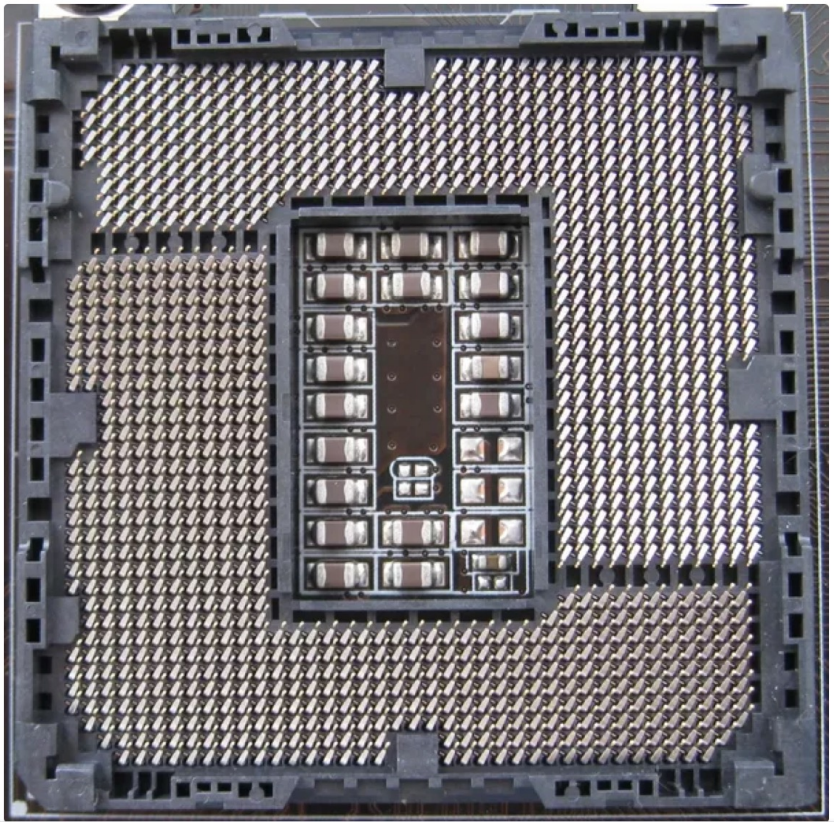
AMD motherboard's don't have pins where the processor sits, instead they have the holes that these pins go into. For example:



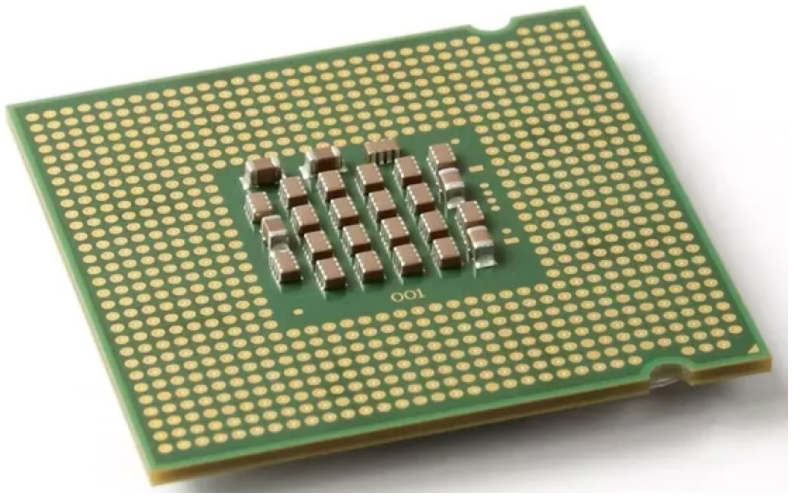


Intel Packaging

Intel processors have the pins **on the motherboard,** like this:



Therefore an **Intel processor** looks like this:



i486 Specs

Joe Zbiciak
Processor architect for ~20 years · 4y

If a 486 CPU was created with the modern chip technology (22nm etc.), how little power would it take?

It would depend on what clock rate you ran it at, and a number of other assumptions.

The original 486 had the following specs:

- 1.2 million transistors
- 1µm process technology
- 5V V_{DD}
- 33MHz top clock rate 33MHz
- 4.7W max power consumption 4.7W

Let's suppose you merely shrunk that first-generation 1µm 486 down to 14nm, lowered the voltage down from 5V to 0.8V, and held clock rate constant at 33MHz.

You'd get an immediate $40\times$ reduction just due to the lower voltage alone.

Computer Architecture

x86 ISA

x86 ISA

Stack instructions [edit]

The x86 architecture has hardware support for an execution stack mechanism. Instructions such as `push`, `pop`, `call` and `ret` are used with the properly set up stack to pass parameters, to allocate space for local data, and to save and restore call-return points. The `ret` *size* instruction is very useful for implementing space efficient (and fast) calling conventions where the callee is responsible for reclaiming stack space occupied by parameters.

When setting up a stack frame to hold local data of a recursive procedure there are several choices; the high level `enter` instruction (introduced with the 80386) takes a *procedure-nesting-depth* argument as well as a *local size* argument, and *may* be faster than more explicit manipulation of the registers (such as `push bp` ; `mov bp, sp` ; `sub sp, size`). Whether it is faster or slower depends on the particular x86-processor implementation as well as the calling convention used by the compiler, programmer or particular program code; most x86 code is intended to run on x86-processors from several manufacturers and on different technological generations of processors, which implies highly varying microarchitectures and microcode solutions as well as varying gate- and transistor-level design choices.

The full range of addressing modes (including *immediate* and *base+offset*) even for instructions such as `push` and `pop`, makes direct usage of the stack for integer, floating point and address data simple, as well as keeping the ABI specifications and mechanisms relatively simple compared to some RISC architectures (require more explicit call stack details).

Integer ALU instructions [edit]

x86 assembly has the standard mathematical operations, `add`, `sub`, `mul`, with `idiv`; the logical operators `and`, `or`, `xor`, `neg`; bitshift arithmetic and logical, `sal` / `sar`, `shl` / `shr`; rotate with and without carry, `rcl` / `rcr`, `rol` / `ror`, a complement of BCD arithmetic instructions, `aaa`, `aad`, `daa` and others.

Flags [edit]

The 8086 has a 16-bit flags register. Nine of these condition code flags are active, and indicate the current state of the processor: Carry flag (CF), Parity flag (PF), Auxiliary carry flag (AF), Zero flag (ZF), Sign flag (SF), Trap flag (TF), Interrupt flag (IF), Direction flag (DF), and Overflow flag (OF). Also referred to as the status word, the layout of the flags register is as follows:[9]

Bit	15-12	11	10	9	8	7	6	5	4	3	2	1	0
Flag		OF	DF	IF	TF	SF	ZF		AF		PF		CF

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP122

x86 Registers

DR JEFF SOFTWARE
INDIE APP DEVELOPER

16-bit [edit]

The original Intel 8086 and 8088 have fourteen 16-bit registers. Four of them (AX, BX, CX, DX) are general-purpose registers (GPRs), although each may have an additional purpose; for example, only CX can be used as a counter with the *loop* instruction. Each can be accessed as two separate bytes (thus BX's high byte can be accessed as BH and low byte as BL). Two pointer registers have special roles: SP (stack pointer) points to the "top" of the stack, and BP (base pointer) is often used to point at some other place in the stack, typically above the local variables (see frame pointer). The registers SI, DI, BX and BP are address registers, and may also be used for array indexing.

Four segment registers (CS, DS, SS and ES) are used to form a memory address. The FLAGS register contains flags such as carry flag, overflow flag and zero flag. Finally, the instruction pointer (IP) points to the next instruction that will be fetched from memory and then executed; this register cannot be directly accessed (read or written) by a program.[20]

The Intel 80186 and 80188 are essentially an upgraded 8086 or 8088 CPU, respectively, with on-chip peripherals added, and they have the same CPU registers as the 8086 and 8088 (in addition to interface registers for the peripherals).

The 8086, 8088, 80186, and 80188 can use an optional floating-point coprocessor, the 8087. The 8087 appears to the programmer as part of the CPU and adds eight 80-bit wide registers, st(0) to st(7), each of which can hold numeric data in one of seven formats: 32-, 64-, or 80-bit floating point, 16-, 32-, or 64-bit (binary) integer, and 80-bit packed decimal integer.[6]:S-6, S-13..S-15 It also has its own 16-bit status register accessible through the `fntsw` instruction, and it is not uncommon to simply use some of its bits for branching by copying it *into* the normal FLAGS.[21]

In the Intel 80286, to support protected mode, three special registers hold descriptor table addresses (GDTR, LDTR, IDTR), and a fourth task register (TR) is used for task switching. The 80287 is the floating-point coprocessor for the 80286 and has the same registers as the 8087 with the same data formats.

x86 Registers

20-bit

Intel 8086 registers

19 18 17 16 15 14 13 12 11 10	9 8 7 6 5 4 3 2 1 0	*(bit position)*
Main registers		
AH	AL	**AX** (primary accumulator)
BH	BL	**BX** (base, accumulator)
CH	CL	**CX** (counter, accumulator)
DH	DL	**DX** (accumulator, extended acc.)
Index registers		
0000	SI	Source Index
0000	DI	Destination Index
0000	BP	Base Pointer
0000	SP	Stack Pointer
Program counter		
0000	IP	Instruction Pointer
Segment registers		
CS	0000	Code Segment
DS	0000	Data Segment
ES	0000	Extra Segment
SS	0000	Stack Segment
Status register		
- - - - O D I T S Z - A - P - C		Flags

x86 Registers

Registers [edit]

Further information: X86 architecture § x86 registers

x86 processors have a collection of registers available to be used as stores for binary data. Collectivel
Each register has a special purpose in addition to what they can all do:

- AX multiply/divide, string load & store
- CX count for string operations & shifts
- DX port address for IN and OUT
- BX index register for MOVE
- SP points to top of the stack
- BP points to base of the stack frame
- SI points to a source in stream operations
- DI points to a destination in stream operations

Along with the general registers there are additionally the:

- IP instruction pointer
- FLAGS
- segment registers (CS, DS, ES, FS, GS, SS) which determine where a 64k segment starts (no FS
- extra extension registers (MMX, 3DNow!, SSE, etc.) (Pentium & later only).

The IP register points to the memory offset of the next instruction in the code segment (it points to the by the programmer directly.

The x86 registers can be used by using the MOV instructions. For example, in Intel syntax:

```
mov ax, 1234h ; copies the value 1234hex (4660d) into register AX
```

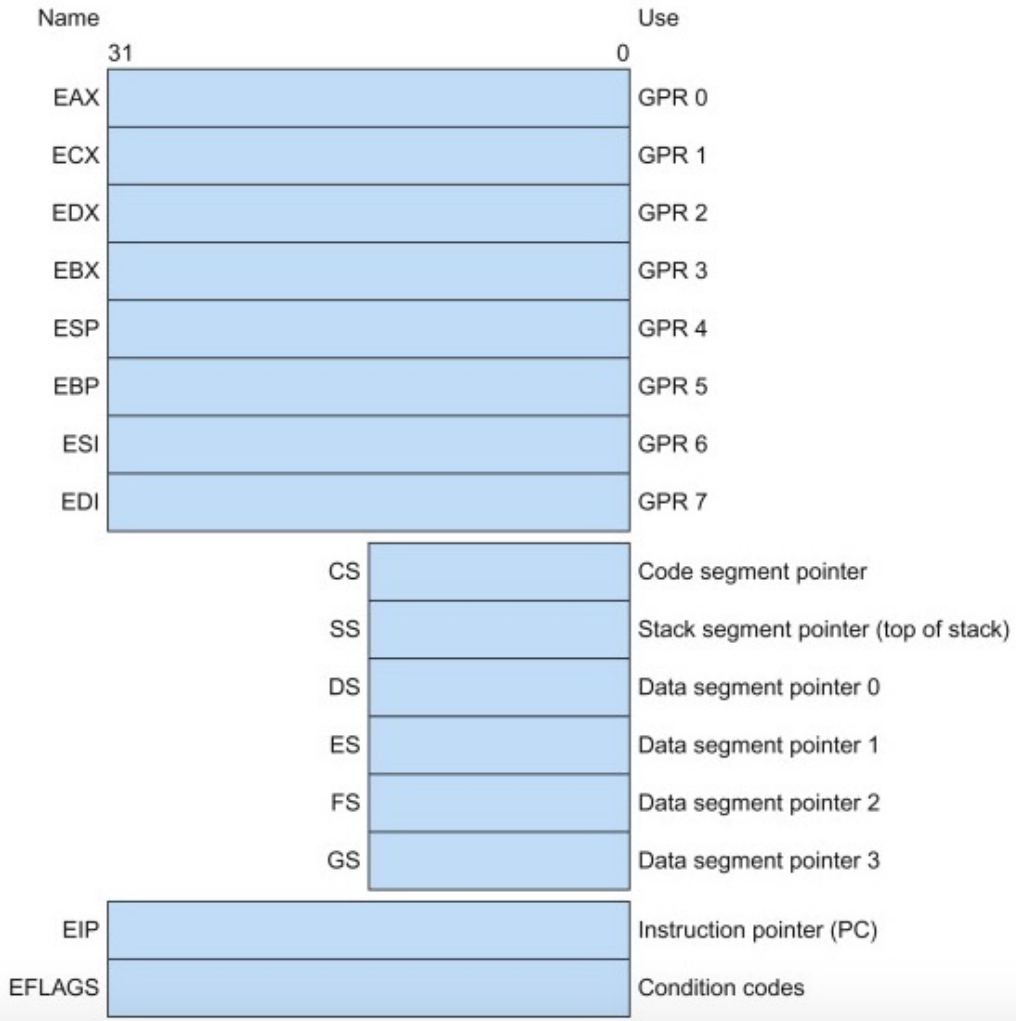
```
mov bx, ax    ; copies the value of the AX register into the BX register
```

x86 Registers

Hennessy & Patterson

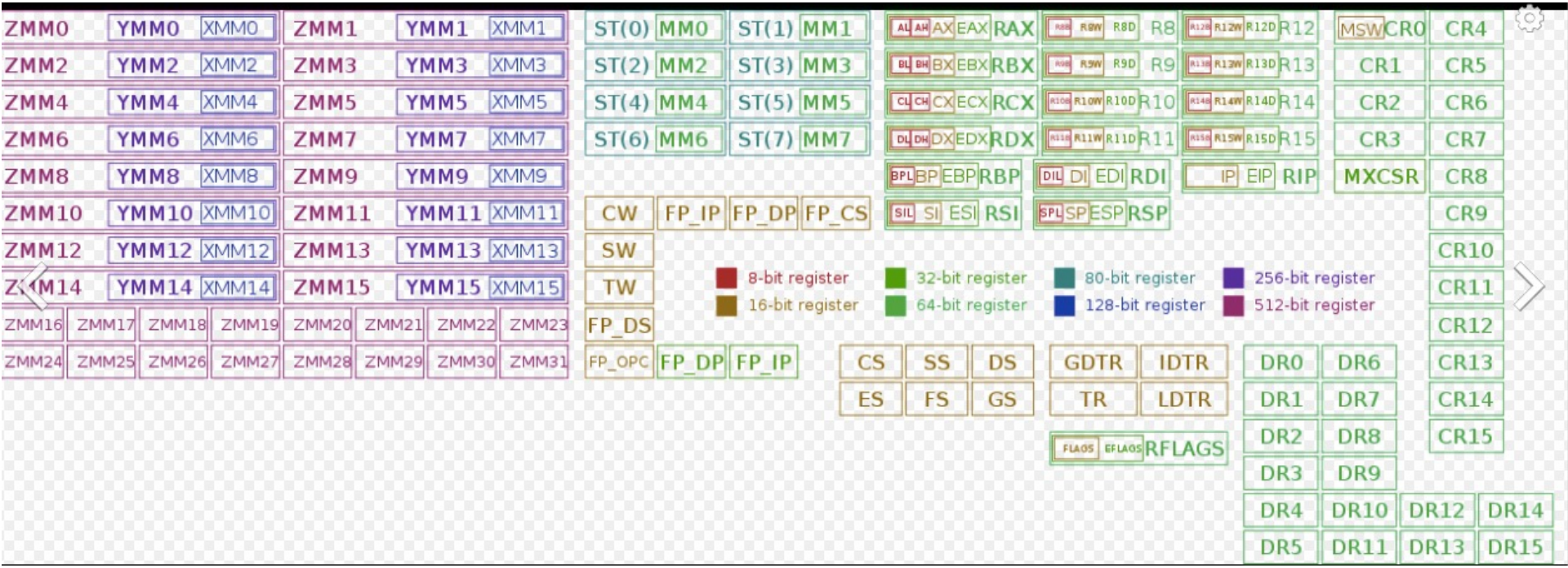
Figure 2.17.1: The 80386 register set (COD Figure 2.36).

Starting with the 80386, the top eight registers were extended to 32 bits and could also be used as general-purpose registers.



CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP122

x86 Register Chart



- 8/16/32/64 bit basic registers
- 128/256/512 bit MMX extended registers

x86 Registers

Registers	
General purpose	• 16-bit: 6 semi-dedicated registers, BP and SP are not general-purpose • 32-bit: 8 GPRs, including EBP and ESP • 64-bit: 16 GPRs, including RBP and RSP
Floating point	• 16-bit: optional separate x87 FPU • 32-bit: optional separate or integrated x87 FPU, integrated SSE2 units in later processors • 64-bit: integrated x87 and SSE2 units, later implementations extended to AVX2 and AVX512

x86 Registers

Structure [edit]

General Purpose Registers (A, B, C and D)

64	56	48	40	32	24	16	8
R?X							
				E?X			
						?X	
						?H	?L

64-bit mode-only General Purpose Registers (R8, R9, R10, R11, R12, R13, R14, R15)

64	56	48	40	32	24	16	8
?							
				?D			
						?W	
							?B

Segment Registers (C, D, S, E, F and G)

16	8
?S	

Pointer Registers (S and B)

64	56	48	40	32	24	16	8
R?P							
				E?P			
						?P	
							?PL

Note: The ?PL registers are only available in 64-bit mode.

x86 Registers

Index Registers (S and D)

64	56	48	40	32	24	16	8
R?I							
				E?I			
						?I	
							?IL

Note: The ?IL registers are only available in 64-bit mode.

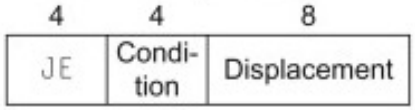
Instruction Pointer Register (I)

64	56	48	40	32	24	16	8
RIP							
				EIP			
						IP	

x86 Instruction Formats

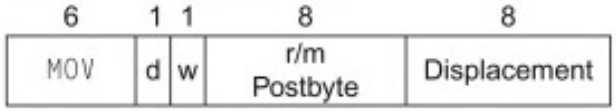
Hennessy & Patterson

a. JE EIP + displacement

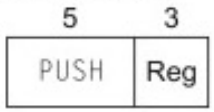


b. CALL

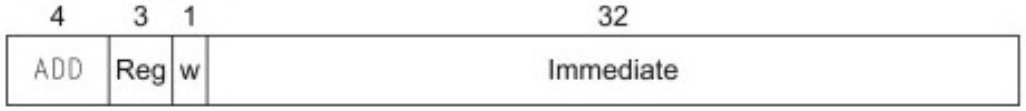
c. MOV EBX, [EDI + 45]



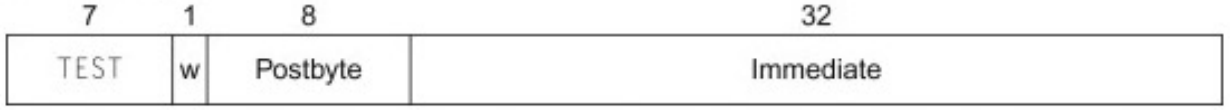
d. PUSH ESI



e. ADD EAX, #6765



f. TEST EDX, #42



x86 Instructions

Hennessy & Patterson

Figure 2.17.4: Some typical x86 instructions and their functions (COD Figure 2.39).

A list of frequent operations appears in the figure below. The CALL saves the EIP of the next instruction on the stack. (EIP is the Intel PC.)

Instruction	Function
je name	if equal(condition code) {EIP=name}; EIP-128 <= name < EIP+128
jmp name	EIP=name
call name	SP=SP-4; M[SP]=EIP+5; EIP=name;
movw EBX,[EDI+45]	EBX=M[EDI+45]
push ESI	SP=SP-4; M[SP]=ESI
pop EDI	EDI=M[SP]; SP=SP+4
add EAX,#6765	EAX= EAX+6765
test EDX,#42	Set condition code (flags) with EDX and 42
movsl	M[EDI]=M[ESI]; EDI=EDI+4; ESI=ESI+4

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP122

x86 Instructions

Hennessy & Patterson

Figure 2.17.5: Some typical operations on the x86 (COD Figure 2.40).

Many operations use register-memory format, where either the source or the destination may be memory and the other may be a register or immediate operand.

Instruction	Meaning
Control	**Conditional and unconditional branches**
`jnz, jz`	Jump if condition to EIP + 8-bit offset; `JNE` (for`JNZ`), `JE` (for `JZ`) are alternative names
`jmp`	Unconditional jump—8-bit or 16-bit offset
`call`	Subroutine call—16-bit offset; return address pushed onto stack
`ret`	Pops return address from stack and jumps to it
`loop`	Loop branch—decrement ECX; jump to EIP + 8-bit displacement if ECX ≠ 0
Data transfer	**Move data between registers or between register and memory**
`move`	Move between two registers or between register and memory
`push, pop`	Push source operand on stack; pop operand from stack top to a register
`les`	Load ES and one of the GPRs from memory
Arithmetic, logical	**Arithmetic and logical operations using the data registers and memory**
`add, sub`	Add source to destination; subtract source from destination; register-memory format
`cmp`	Compare source and destination; register-memory format
`shl, shr, rcr`	Shift left; shift logical right; rotate right with carry condition code as fill
`cbw`	Convert byte in eight rightmost bits of EAX to 16-bit word in right of EAX
`test`	Logical AND of source and destination sets condition codes
`inc, dec`	Increment destination, decrement destination
`or, xor`	Logical OR; exclusive OR; register-memory format
String	**Move between string operands; length given by a repeat prefix**
`movs`	Copies from string source to destination by incrementing ESI and EDI; may be repeated
`lods`	Loads a byte, word, or doubleword of a string into the EAX register

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP122

x86 Operands

Hennessy & Patterson

Figure 2.17.2: Instruction types for the arithmetic, logical, and data transfer instructions (COD Figure 2.37).

The x86 allows the combinations shown. The only restriction is the absence of a memory-memory mode. Immediates may be 8, 16, or 32 bits in length; a register is any one of the 14 major registers in the figure above (not EIP or EFLAGS).

Source/destination operand type	Second source operand
Register	Register
Register	Immediate
Register	Memory
Memory	Register
Memory	Immediate

x86 Addressing Modes

Addressing modes for 16-bit x86 processors can be summarized by the formula:[16][17]

$$\begin{matrix}\texttt{CS}:\\ \texttt{DS}:\\ \texttt{SS}:\\ \texttt{ES}:\end{matrix}\left(\begin{bmatrix}\texttt{BX}\\ \texttt{BP}\end{bmatrix}+\begin{bmatrix}\texttt{SI}\\ \texttt{DI}\end{bmatrix}\right)+\text{displacement}$$

Addressing modes for 32-bit x86 processors,[18] and for 32-bit code on 64-bit x86 processors, c

$$\begin{matrix}\texttt{CS}:\\ \texttt{DS}:\\ \texttt{SS}:\\ \texttt{ES}:\\ \texttt{FS}:\\ \texttt{GS}:\end{matrix}\begin{bmatrix}\texttt{EAX}\\ \texttt{EBX}\\ \texttt{ECX}\\ \texttt{EDX}\\ \texttt{ESP}\\ \texttt{EBP}\\ \texttt{ESI}\\ \texttt{EDI}\end{bmatrix}+\left(\begin{bmatrix}\texttt{EAX}\\ \texttt{EBX}\\ \texttt{ECX}\\ \texttt{EDX}\\ \texttt{EBP}\\ \texttt{ESI}\\ \texttt{EDI}\end{bmatrix}*\begin{bmatrix}1\\ 2\\ 4\\ 8\end{bmatrix}\right)+\text{displacement}$$

Addressing modes for 64-bit code on 64-bit x86 processors can be summarized by the formula:

$$\left\{\begin{matrix}\texttt{FS}:\\ \texttt{GS}:\end{matrix}\begin{bmatrix}\vdots\\ \texttt{GPR}\\ \vdots\end{bmatrix}+\left(\begin{bmatrix}\vdots\\ \texttt{GPR}\\ \vdots\end{bmatrix}*\begin{bmatrix}1\\ 2\\ 4\\ 8\end{bmatrix}\right)\right\}+\text{displacement}$$

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE

COMP122

x86 Instructions

Hennessy & Patterson

Figure 2.17.3: x86 32-bit addressing modes with register restrictions and the equivalent MIPS code (COD Figure 2.38).

The Base plus Scaled Index addressing mode, not found in ARM or MIPS, is included to avoid the multiplies by 4 (scale factor of 2) to turn an index in a register into a byte address (see COD Figures 2.25 (MIPS assembly code of the procedure swap) and 2.27 (MIPS assembly version of procedure sort)). A scale factor of 1 is used for 16-bit data, and a scale factor of 3 for 64-bit data. A scale factor of 0 means the address is not scaled. If the displacement is longer than 16 bits in the second or fourth modes, then the MIPS equivalent mode would need two more instructions: a `lui` to load the upper 16 bits of the displacement and an **add** to sum the upper address with the base register `$s1`. (Intel gives two different names to what is called Based addressing mode—Based and Indexed—but they are essentially identical and we combine them here.)

Mode	Description	Register restrictions	MIPS equivalent
Register indirect	Address is in a register.	Not ESP or EBP	`lw $s0,0($s1)`
Based mode with 8- or 32-bit displacement	Address is contents of base register plus displacement.	Not ESP	`lw $s0,100($s1) # <= 16-bit # displacement`

Base plus scaled index	The address is Base + (2^{Scale} x Index) where Scale has the value 0, 1, 2, or 3.	Base: any GPR Index: not ESP	`mul $t0,$s2,4`
			`add $t0,$t0,$s1`
			`lw $s0,0($t0)`

Base plus scaled index with 8- or 32-bit displacement	The address is Base + (2^{Scale} x Index) + displacement where Scale has the value 0, 1, 2, or 3.	Base: any GPR Index: not ESP	`mul $t0,$s2,4`
			`add $t0,$t0,$s1`
			`lw $s0,100($t0) #<-16-bit # displacement`

x86 Segmentation

Segmentation [edit]

See also: x86 memory segmentation

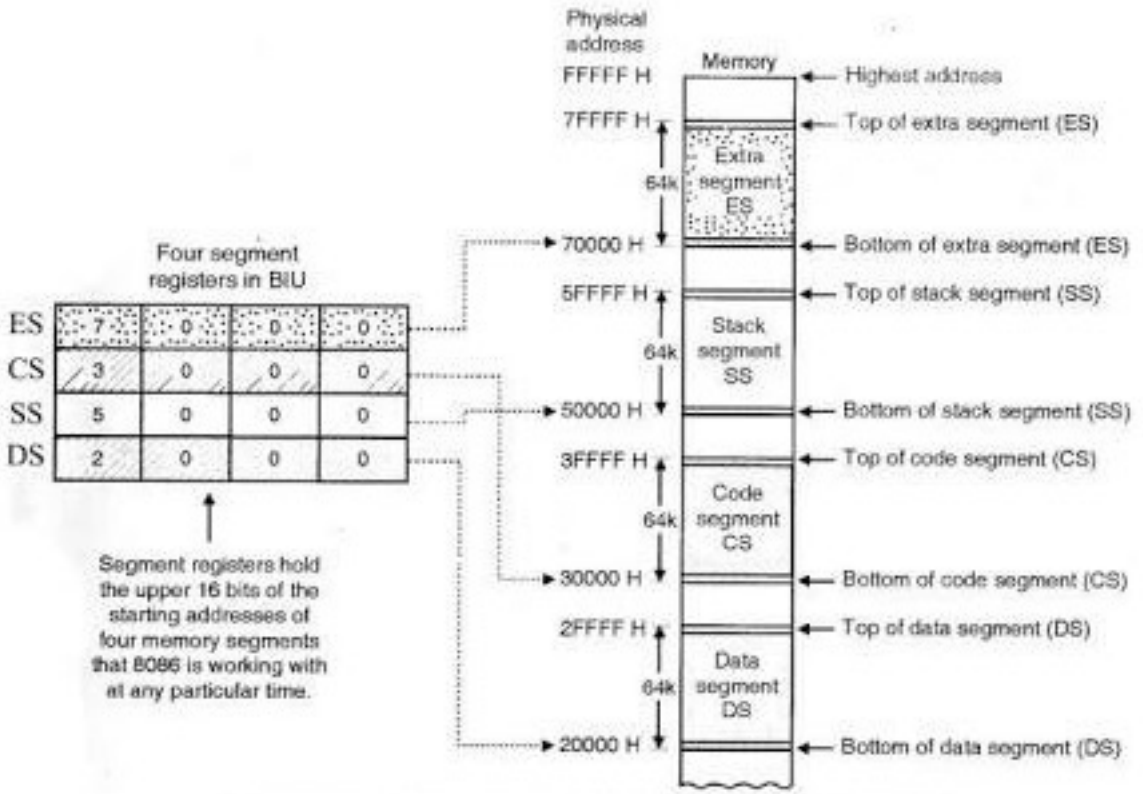
There are also three 16-bit segment registers (see figure) that allow the 8086 CPU to access one megabyte of memory in an unusual way. Rather than concatenating the segment register with the address register, as in most processors whose address space exceeds their register size, the 8086 shifts the 16-bit segment only four bits left before adding it to the 16-bit offset (16×segment + offset), therefore producing a 20-bit external (or effective or physical) address from the 32-bit segment:offset pair. As a result, each external address can be referred to by $2^{12} = 4096$ different segment:offset pairs.

```
  0110 1000 1000 0111 0000 Segment, 16 bits, shifted 4 bits left (or multiplied by 0x10)
+      0011 0100 1010 1001 Offset,    16 bits
-------------------------
  0110 1011 1101 0001 1001 Address, 20 bits
```

Although considered complicated and cumbersome by many programmers, this scheme also has advantages; a small program (less than 64 KB) can be loaded starting at a fixed offset (such as 0000) in its own segment, avoiding the need for relocation, with at most 15 bytes of alignment waste.

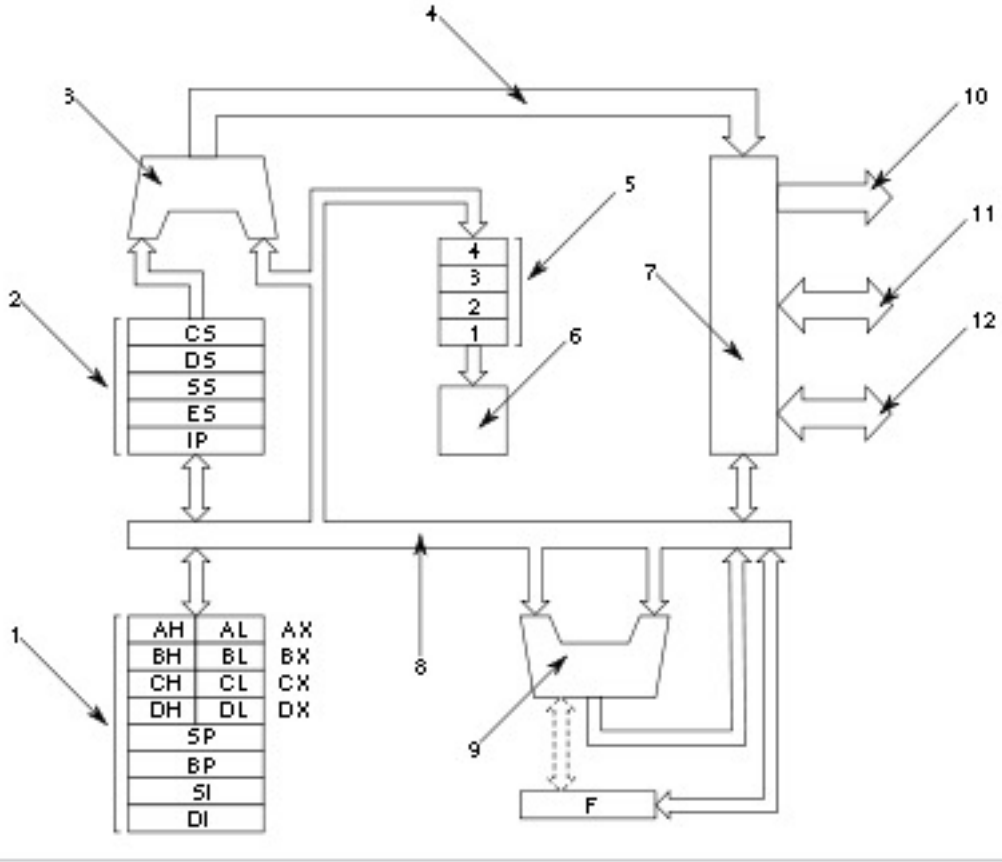
Compilers for the 8086 family commonly support two types of pointer, *near* and *far*. Near pointers are 16-bit offsets implicitly associated with the program's code or data segment and so can be used only within parts of a program small enough to fit in one segment. Far pointers are 32-bit segment:offset pairs resolving to 20-bit external addresses. Some compilers also support *huge* pointers, which are like far pointers except that pointer arithmetic on a huge pointer treats it as a linear 20-bit pointer, while pointer arithmetic on a far pointer wraps around within its 16-bit offset without touching the segment part of the address.

x86 Segmentation



One way of positioning four 64k byte segments within the 1M byte memory space of an 8086

x86 Organization



Simplified block diagram over Intel 8088 (a variant of 8086); 1=main registers; 2=segment registers and IP; 3=address adder; 4=internal address bus; 5=instruction queue; 6=control unit (very simplified!); 7=bus interface; 8=internal databus; 9=ALU; 10/11/12=external address/data/control bus.

x86 Performance

CISC

Execution times for typical instructions (in clock cycles)[12]

instruction	register-register	register immediate	register-memory	memory-register	memory-immediate
mov	2	4	8+EA	9+EA	10+EA
ALU	3	4	9+EA,	16+EA,	17+EA
jump	*register* ≥ 11 ; *label* ≥ 15 ; *condition,label* ≥ 16				
integer multiply	70~160 (depending on operand *data* as well as size) *including* any EA				
integer divide	80~190 (depending on operand *data* as well as size) *including* any EA				

- EA = time to compute effective address, ranging from 5 to 12 cycles.
- Timings are best case, depending on prefetch status, instruction alignment, and other factors.

CSUN
CALIFORNIA
STATE UNIVERSITY
NORTHRIDGE

COMP122

x86 Modes

Execution modes [edit]

Further information: X86 architecture

The x86 processors support five modes of operation for x86 code, **Real Mode**, **Protected Mode**, **Long Mode**, **Virtual 86 Mode**, and **System Management Mode**, in which some instructions are available and others are not. A 16-bit subset of instructions are available on the 16-bit x86 processors, which are the 8086, 8088, 80186, 80188, and 80286. These instructions are available in real mode on all x86 processors, and in 16-bit protected mode (80286 onwards), additional instructions relating to protected mode are available. On the 80386 and later, 32-bit instructions (including later extensions) are also available in all modes, including real mode; on these CPUs, V86 mode and 32-bit protected mode are added, with additional instructions provided in these modes to manage their features. SMM, with some of its own special instructions, is available on some Intel i386SL, i486 and later CPUs. Finally, in long mode (AMD Opteron onwards), 64-bit instructions, and more registers, are also available. The instruction set is similar in each mode but memory addressing and word size vary, requiring different programming strategies.

The modes in which x86 code can be executed in are:

- Real mode (16-bit)
- Protected mode (16-bit and 32-bit)
- Long mode (64-bit)
- Virtual 8086 mode (16-bit)
- System Management Mode (16-bit)

Switching modes [edit]

The processor runs in real mode immediately after power on, so an operating system kernel, or other program, must explicitly switch to another mode if it wishes to run in anything but real mode. Switching modes is accomplished by modifying certain bits of the processor's control registers after some preparation, and some additional setup may be required after the switch.

x86 ISA

Intel, AMD, VIA

4. Instruction tables

Lists of instruction latencies, throughputs and micro-operation breakdowns for Intel, AMD, and VIA CPUs

By Agner Fog. Technical University of Denmark.

Introduction

This is the fourth in a series of five manuals:

1. Optimizing software in C++: An optimization guide for Windows, Linux, and Mac platforms.
2. Optimizing subroutines in assembly language: An optimization guide for x86 platforms.
3. The microarchitecture of Intel, AMD, and VIA CPUs: An optimization guide for assembly programmers and compiler makers.
4. Instruction tables: Lists of instruction latencies, throughputs and micro-operation breakdowns for Intel, AMD, and VIA CPUs.
5. Calling conventions for different C++ compilers and operating systems.

x86 ISA

Instruction sets

Explanation of instruction sets for x86 processors

x86	This is the name of the common instruction set, supported by all processors in this lineage.
80186	This is the first extension to the x86 instruction set. New integer instructions: PUSH i, PUSHA, POPA, IMUL r,r,i, BOUND, ENTER, LEAVE, shifts and rotates by immediate ≠ 1.
80286	System instructions for 16-bit protected mode.
80386	The eight general purpose registers are extended from 16 to 32 bits. 32-bit addressing. 32-bit protected mode. Scaled index addressing. MOVZX, MOVSX, IMUL r,r, SHLD, SHRD, BT, BTR, BTS, BTC, BSF, BSR, SETcc.
80486	BSWAP. Later versions have CPUID.
x87	This is the floating point instruction set. Supported when a 8087 or later coprocessor is present. Some 486 processors and all processors since Pentium/ K5 have built-in support for floating point instructions without the need for a coprocessor.
80287	FSTSW AX
80387	FPREM1, FSIN, FCOS, FSINCOS.

x86 ISA

MMX

Pentium	RDTSC, RDPMC.
PPro	Conditional move (CMOV, FCMOV) and fast floating point compare (FCOMI) instructions introduced in Pentium Pro. These instructions are not supported in Pentium MMX, but are supported in all processors with SSE and later.
MMX	Integer vector instructions with packed 8, 16 and 32-bit integers in the 64-bit MMX registers MM0 - MM7, which are aliased upon the floating point stack registers ST(0) - ST(7).
SSE	Single precision floating point scalar and vector instructions in the new 128-bit XMM registers XMM0 - XMM7. PREFETCH, SFENCE, FXSAVE, FXRSTOR, MOVNTQ, MOVNTPS. The use of XMM registers requires operating system support.
SSE2	Double precision floating point scalar and vector instructions in the 128-bit XMM registers XMM0 - XMM7. 64-bit integer arithmetics in the MMX registers. Integer vector instructions with packed 8, 16, 32 and 64-bit integers in the XMM registers. MOVNTI, MOVNTPD, PAUSE, LFENCE, MFENCE.
SSE3	FISTTP, LDDQU, MOVDDUP, MOVSHDUP, MOVSLDUP, ADDSUBPS, ADDSUPPD, HADDPS, HADDPD, HSUBPS, HSUBPD.
SSSE3	(Supplementary SSE3): PSHUFB, PHADDW, PHADDSW, PHADDD, PMADDUBSW, PHSUBW, PHSUBSW, PHSUBD, PSIGNB, PSIGNW, PSIGND, PMULHRSW, PABSB, PABSW, PABSD, PALIGNR.

x86 ISA

AVX2

AVX2	Integer vector instructions are available in 256-bit versions. Furthermore, the following instructions are added in AVX2: ANDN, BEXTR, BLSI, BLSMSK, BLSR, BZHI, INVPCID, LZCNT, MULX, PEXT, PDEP, RORX, SARX, SHLX, SHRX, TZCNT, VBROADCASTI128, VBROADCASTSS, VBROADCASTSD, VEXTRACTI128, VGATHERDPD, VGATHERQPD, VGATHERDPS, VGATHERQPS, VPGATHERDD, VPGATHERQD, VPGATHERDQ, VPGATHERQQ, VINSERTI128, VPERM2I128, VPERMD, VPERMPD, VPERMPS, VPERMQ, VPMASKMOVD, VPMASKMOVQ, VPSLLVD, VPSLLVQ, VPSRAVD, VPSRLVD, VPSRLVQ.
FMA3	(FMA): Fused multiply and add instructions: VFMADDxxxPD, VFMADDxxxPS, VFMADDxxxSD, VFMADDxxxSS, VFMADDSUBxxxPD, VFMADDSUBxxxPS, VFMSUBADDxxxPD, VFMSUBADDxxxPS, VFMSUBxxxPD, VFMSUBxxxPS, VFMSUBxxxSD, VFMSUBxxxSS, VFNMADDxxxPD, VFNMADDxxPS, VFNMADDxxxSD, VFNMADDxxxSS, VFNMSUBxxxPD, VFNMSUBxxxPS, VFNMSUBxxxSD, VFNMSUBxxxSS.
FMA4	Same as Intel FMA, but with 4 different operands according to a preliminary Intel specification which is now supported only by some AMD processors. Intel's FMA specification has later been changed to FMA3, which is now also supported by AMD.

x86 ISA

x64

64 bit

This instruction set is called x86-64, x64, AMD64 or EM64T. It defines a new 64-bit mode with 64-bit addressing and the following extensions: The general purpose registers are extended to 64 bits, and the number of general purpose registers is extended from eight to sixteen. The number of XMM registers is also extended from eight to sixteen, but the number of MMX and ST registers is still eight. Data can be addressed relative to the instruction pointer. There is no way to get access to these extensions in 32-bit mode

Most instructions that involve segmentation are not available in 64 bit mode. Direct far jumps and calls are not allowed, but indirect far jumps, indirect far calls and far returns are allowed. These are used in system code for switching mode. Segment registers DS, ES, and SS cannot be used. The FS and GS segments and segment prefixes are available in 64 bit mode and are used for addressing thread environment blocks and processor environment blocks

x86 ISA Evolution

Evolution of the Intel x86

ARM and MIPS were the vision of single small groups in 1985; the pieces of these architectures fit nicely together, and the whole architecture can be described succinctly. Such is not the case for the x86; it is the product of several independent groups who evolved the architecture over 35 years, adding new features to the original instruction set as someone might add clothing to a packed bag. Here are important x86 milestones.

- **1978**: The Intel 8086 architecture was announced as an assembly language—compatible extension of the then successful Intel 8080, an 8-bit microprocessor. The 8086 is a 16-bit architecture, with all internal registers 16 bits wide. Unlike MIPS, the registers have dedicated uses, and hence the 8086 is not considered a *general-purpose register* architecture.

General-purpose register (***GPR***): A register that can be used for addresses or for data with virtually any instruction.

- **1980**: The Intel 8087 floating-point coprocessor is announced. This architecture extends the 8086 with about 60 floating-point instructions. Instead of using registers, it relies on a stack (see COD Section 2.21 (Historical perspective and further reading) and COD Section 3.7 (Real Stuff: Streaming SIMD extensions and advanced vector extensions in x86)).
- **1982**: The 80286 extended the 8086 architecture by increasing the address space to 24 bits, by creating an elaborate memory-mapping and protection model (see COD Chapter 5 (Large and Fast: Exploiting Memory Hierarchy)), and by adding a few instructions to round out the instruction set and to manipulate the protection model.
- **1985**: The 80386 extended the 80286 architecture to 32 bits. In addition to a 32-bit architecture with 32-bit registers and a 32-bit address space, the 80386 added new addressing modes and additional operations. The added instructions make the 80386 nearly a general-purpose register machine. The 80386 also added paging support in addition to segmented addressing (see COD Chapter 5 (Large and Fast: Exploiting Memory Hierarchy)). Like the 80286, the 80386 has a mode to execute 8086 programs without change.
- **1989-95**: The subsequent 80486 in 1989, Pentium in 1992, and Pentium Pro in 1995 were aimed at higher performance, with only four instructions added to the user-visible instruction set: three to help with multiprocessing (COD Chapter 6 (Parallel Processor from Client to Cloud)) and a conditional move instruction.

x86 ISA Evolution

1978-2011

Hennessy & Patterson

- **1997**: After the Pentium and Pentium Pro were shipping, Intel announced that it would expand the Pentium and the Pentium Pro architectures with MMX (Multi Media Extensions). This new set of 57 instructions uses the floating-point stack to accelerate multimedia and communication applications. MMX instructions typically operate on multiple short data elements at a time, in the tradition of *single instruction, multiple data* (SIMD) architectures (see COD Chapter 6 (Parallel Processor from Client to Cloud)). Pentium II did not introduce any new instructions.
- **1999**: Intel added another 70 instructions, labeled SSE (*Streaming SIMD Extensions*) as part of Pentium III. The primary changes were to add eight separate registers, double their width to 128 bits, and add a single precision floating-point data type. Hence, four 32-bit floating-point operations can be performed in parallel. To improve memory performance, SSE includes cache prefetch instructions plus streaming store instructions that bypass the caches and write directly to memory.

- **2001**: Intel added yet another 144 instructions, this time labeled SSE2. The new data type is double precision arithmetic, which allows pairs of 64-bit floating-point operations in parallel. Almost all of these 144 instructions are versions of existing MMX and SSE instructions that operate on 64 bits of data in parallel. Not only does this change enable more multimedia operations; it gives the compiler a different target for floating-point operations than the unique stack architecture. Compilers can choose to use the eight SSE registers as floating-point registers like those found in other computers. This change boosted the floating-point performance of the Pentium 4, the first microprocessor to include SSE2 instructions.
- **2003**: A company other than Intel enhanced the x86 architecture this time. AMD announced a set of architectural extensions to increase the address space from 32 to 64 bits. Similar to the transition from a 16- to 32-bit address space in 1985 with the 80386, AMD64 widens all registers to 64 bits. It also increases the number of registers to 16 and increases the number of 128-bit SSE registers to 16. The primary ISA change comes from adding a new mode called *long mode* that redefines the execution of all x86 instructions with 64-bit addresses and data. To address the larger number of registers, it adds a new prefix to instructions. Depending how you count, long mode also adds four to ten new instructions and drops 27 old ones. PC-relative data addressing is another extension. AMD64 still has a mode that is identical to x86 (*legacy mode*) plus a mode that restricts user programs to x86 but allows operating systems to use AMD64 (*compatibility mode*). These modes allow a more graceful transition to 64-bit addressing than the HP/Intel IA-64 architecture.

x86 ISA Evolution

- **2004**: Intel capitulates and embraces AMD64, relabeling it *Extended Memory 64 Technology* (EM64T). The major difference is that Intel added a 128-bit atomic compare and swap instruction, which probably should have been included in AMD64. At the same time, Intel announced another generation of media extensions. SSE3 adds 13 instructions to support complex arithmetic, graphics operations on arrays of structures, video encoding, floating-point conversion, and thread synchronization (see COD Section 2.11 (Parallelism and instructions: Synchronization)). AMD added SSE3 in subsequent chips and the missing atomic swap instruction to AMD64 to maintain binary compatibility with Intel.
- **2006**: Intel announces 54 new instructions as part of the SSE4 instruction set extensions. These extensions perform tweaks like sum of absolute differences, dot products for arrays of structures, sign or zero extension of narrow data to wider sizes, population count, and so on. They also added support for virtual machines (see COD Chapter 5 (Large and Fast: Exploiting Memory Hierarchy)).
- **2007**: AMD announces 170 instructions as part of SSE5, including 46 instructions of the base instruction set that adds three operand instructions like MIPS.
- **2011**: Intel ships the Advanced Vector Extension that expands the SSE register width from 128 to 256 bits, thereby redefining about 250 instructions and adding 128 new instructions.

This history illustrates the impact of the "golden handcuffs" of compatibility on the x86, as the existing software base at each step was too important to jeopardize with significant architectural changes.

Golden Handcuffs of x86 compatibility

x86 Instruction Sizes

Quora

Andrey Gazibarov, System programmer and moderate SW fan since 1984

The reason fot this is the existence of instruction prefixes: special opcodes that are not instructions but modify the execution of the next instruction. I8086 has two types of prefixes and no limit on their number and sequence.

Otherwise, the longest meaningfull instruction is 8 bytes, e.g.

```
1  F0 26 81 80 1234 5678
2  LOCK ADD [ES:BX+SI+1234],5678
```

Without anything new in the encoding scheme, I80286 imposed an 11-byte limit on instructions. Such length may be reached only if redundant prefixes are present. If exceeded, a general protection exception is raised.

x86 Instruction Sizes

Quora

Andrey Gazibarov, System programmer and moderate SW fan since 1984

I80386 introduced two additional prefixes, 32-bit offsets and immediates and a second address byte. Thus the theoretical maximum length of an instruction became 15 bytes.

```
1  .16p
2  F0 67 26 66 81 84 1E 12345678 9ABCDEF0
3  LOCK ADD [ES:EBX+ESI+12345678],9ABCDEF0
```

The imposed limit is 15 bytes as well.

The REX, VEX and EVEX encoding schemes introduced not only new sets of prefixes, but also limits on presence and ordering of other prefixes. Incorrect usage raises the undefined instruction exception.

The limit of 15 bytes still stays.

x86 Micro Architecture

The microarchitecture of Intel, AMD, and VIA CPUs

An optimization guide for assembly programmers and compiler makers

By Agner Fog. Technical University of Denmark.

x86 Micro Architecture

3 Branch prediction (all processors)

The pipeline in a modern microprocessor contains many stages, including instruction fetch, decoding, register allocation and renaming, µop reordering, execution, and retirement. Handling instructions in a pipelined manner allows the microprocessor to do many things at the same time. While one instruction is being executed, the next instructions are being fetched and decoded. The biggest problem with pipelining is branches in the code. For example, a conditional jump allows the instruction flow to go in any of two directions. If there is only one pipeline, then the microprocessor does not know which of the two branches to feed into the pipeline until the branch instruction has been executed. The longer the pipeline, the more time does the microprocessor waste if it does not know which branch to feed into the pipeline.

The microarchitecture tries to overcome this problem by feeding the most probable branch into the pipeline and execute it *speculatively*. Speculative execution means that the instructions are decoded and executed, but the results are not *retired* into the permanent register file, and memory writes are pending until the branch instruction is finally resolved. If it turns out that the guess was wrong and the wrong branch was executed speculatively, then the pipeline is flushed, the results of the speculative execution are discarded and the other branch is fed into the pipeline. This is called a branch misprediction, and the result is that several clock cycles are wasted. The number of wasted clock cycles is approximately equal to the length of the pipeline.

The designers are inventing more and more sophisticated mechanisms for predicting which way a branch will go, in order to minimize the frequency of branch mispredictions. The

x86 Micro Architecture

AMD

Processor name	Microarchitecture Code name	Family number (hex)	Model number (hex)
AMD K7 Athlon		6	6
AMD K8 Opteron		F	5
AMD K10 Opteron		10	2
AMD Bulldozer	Bulldozer, Zambezi	15	1
AMD Piledriver	Piledriver	15	2
AMD Steamroller	Steamroller, Kaveri	15	30
AMD Excavator	Bristol Ridge	15	65
AMD Ryzen	Zen 1	17	1
AMD Ryzen 3700	Zen 2	17	71
AMD Ryzen 5000	Zen 3	19	21
AMD Bobcat	Bobcat	14	1
AMD Kabini	Jaguar	16	0

x86 Micro Architecture

Intel

Processor name	Microarchitecture Code name	Family number (hex)	Model number (hex)
Intel Pentium	P5	5	2
Intel Pentium MMX	P5	5	4
Intel Pentium II	P6	6	6
Intel Pentium III	P6	6	7
Intel Pentium 4	Netburst	F	2
Intel Pentium 4 EM64T	Netburst, Prescott	F	4
Intel Pentium M	Dothan	6	D
Intel Core Duo	Yonah	6	E
Intel Core 2 (65 nm)	Merom	6	F
Intel Core 2 (45 nm)	Wolfdale	6	17
Intel Core i7	Nehalem	6	1A

x86 Micro Architecture

Intel

Processor name	Microarchitecture Code name	Family number (hex)	Model number (hex)
Intel Pentium	P5	5	2
Intel Pentium MMX	P5	5	4
Intel Pentium II	P6	6	6
Intel Pentium III	P6	6	7
Intel Pentium 4	Netburst	F	2
Intel Pentium 4 EM64T	Netburst, Prescott	F	4
Intel Pentium M	Dothan	6	D
Intel Core Duo	Yonah	6	E
Intel Core 2 (65 nm)	Merom	6	F
Intel Core 2 (45 nm)	Wolfdale	6	17
Intel Core i7	Nehalem	6	1A
Intel 2nd gen. Core	Sandy Bridge	6	2A
Intel 3rd gen. Core	Ivy Bridge	6	3A
Intel 4th gen. Core	Haswell	6	3C
Intel 5th gen. Core	Broadwell	6	56
Intel 6th gen. Core	Skylake	6	5E
Intel 7th gen. Core	Skylake-X	6	55
Intel 9th gen. Core	Coffee Lake	6	9E
Intel 11th gen. Core	Tiger Lake	6	8C

Interrupts on x86

ZDNet

Torvalds, however, doesn't see it that way. He wrote:

> The AMD version is essentially "Fix known bugs in the exception handling definition."

> The Intel version is basically "Yeah, the protected mode 80286 exception handling was bad, then 386 made it odder with the 32-bit extensions, and then syscall/sysenter made everything worse, and then the x86-64 extensions introduced even more problems. So let's add a mode bit where all the crap goes away."
>
> In contrast, the AMD one is basically a minimal effort to fix actual fundamental problems with all that legacy-induced crap that is nasty to work around and that has caused issues.

Interrupts on x86

ZDNet

So, what are these problems? They are hidden with the x86's architecture's Interrupt Descriptor Table (IDT). This is a data structure that implements an interrupt vector table. It comes, unfortunately, with numerous exception problems. **These, according to Torvalds, include:**

- IDT itself is a horrible nasty format and you shouldn't have to parse memory in odd ways to handle exceptions. It was fundamentally bad from the 80286 beginnings, it got a tiny bit harder to parse for 32-bit, and it arguably got much worse in x86-64.
- The %rsp general-purpose register is not being restored properly by return-to-user mode.
- Delayed debug traps into supervisor mode.
- Several bad exception nesting problems: Non-Maskable Interrupts (NMI), machine checks, and STI-shadow handling at the very least).
- Various atomicity problems with gsbase (swapgs) and stack pointer switching
- Several different exception stack layouts, and literally hundreds of different entry points for exceptions, interrupts and system calls (and that's not even

Section

Mult/Div

x86 Mult/Div

How many machine cycles does it take for a modern 64-bit processor to divide 64- bit numbers?

Joe Zbiciak, Developed practical algorithms actually used in production.

Answered 2h ago

It varies widely, and it could be data dependent.

Fog

If you're truly curious, you can look up much more detailed answers for x86 in Agner Fog's optimization resources. You'll find very different numbers for different generations of Intel, AMD, Via, and other processors.

For ARM Neoverse N1, ARM offers an optimization guide with similar information. On N1, `SDIV` and `UDIV` take from 5 to 20 cycles with 64-bit arguments. ARM notes that N1's divide can early-exit depending on its arguments.

Note that I only looked up integer divide. Floating point divide has different characteristics than integer divide. Some microarchitectures use a converging-divide algorithm. For example, AMD uses Goldschmidt division, in many of its microarchitectures. Others use iterative algorithms similar to integer divide.

Multiply & Divide

MULTIPLY

- *Unsigned* only
- First convert negative numbers (2sC) – NEG op
- Compute result sign: 0 if both signs same, 1 else (not=)
- Complement result if sign is negative – NEG op
- Other MPUs use *signed* multiply (2sC) via "Booth's Algorithm"

DIVIDE

- No hardware, no instruction
- Create subroutine (may find ones in asm library)
- Compute
  - Long division
  - Non-restoring division
  - Iterative subtraction (very slow)
- Use tricks
  - Divide by **2** or any $\mathbf{2^n}$: right SHIFT by n
  - Divide by **10**: convert to BCD, then right SHIFT by 4 (reconvert to binary)
  - Divide by **5**: divide by 10, then multiply by 2 (by shifting after conv. Bin)

x86 Mult/Div

How many machine cycles does it take for a modern 64-bit processor to divide 64- bit numbers?

Integer instructions

Instruction	Operands	Ops	Latency
MUL, IMUL	r16/m16	3	3
MUL, IMUL	r32/m32	3	4
IMUL	r16,r16/m16	2	3
IMUL	r32,r32/m32	2	4
IMUL	r16,(r16),i	2	4
IMUL	r32,(r32),i	2	5
IMUL	r16,m16,i	3	
IMUL	r32,m32,i	3	
DIV	r8/m8	32	24

x86 Mult/Div

How many machine cycles does it take for a modern 64-bit processor to divide 64- bit numbers?

Integer instructions

Instruction	Operands	Ops	Latency
			AMD K7
DIV	r16/m16	47	24
DIV	r32/m32	79	40
IDIV	r8	41	17
IDIV	r16	56	25
IDIV	r32	88	41
IDIV	m8	42	17
IDIV	m16	57	25
IDIV	m32	89	41

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP122

Division Algorithms

Lauri's blog Posts

Floating Point | Can be imprecise

Goldschmidt division algorithm

05. Dec '13

Introduction

Goldschmidt division is iterative division algorithm deployed in many processors. Higher precision can be achieved by adding fraction bits for intermediate calculations or by having more iterations.

Correct result:

$$Q = \frac{N_{-1}}{D_{-1}} = 12.285714285714285714285714285714285714285714285714$$

Initial reciprocal is the inverse of divisor which is calculated by shifting bits around the fraction point:

Integer | Non-Restoring

Computer Architecture

x86 AVX (SIMD)

AVX

Quora

Yowan Rajcoomar, Computer Technician (2008-present)
Answered 18h ago

There is no current use cases which requires general purposes registers larger than 64-bits.

The current trend is the extension of **vector and SIMD capabilities** and the addition of **domain-specific accelerators** which will help demanding applications such as CAD, rendering, scientific workloads, machine/deep learning and AI. There, a lot of data needs to be crunched at the same time so companies such as Intel have introduced the following:

- AVX-512F which bumps the number of FP/SIMD registers from 16 to 32 while providing additional features (at the cost of die space and thermals)
- AVX-512 VNNI (Vector Neural Instructions) for handing 8 and 16-bit values in convolutional neural networks. This was made available in Cascade and Ice Lake-based chips.
- AVX-512 BF16 which provides a speedup when dealing with dot products on bfloat16 pairs.
- x86 AMX or Advanced Matrix Extension. This is literally an accelerator built into the x86 core with its own register file and instructions. This will debut sometime later this year with Intel's next HEDT lineup.

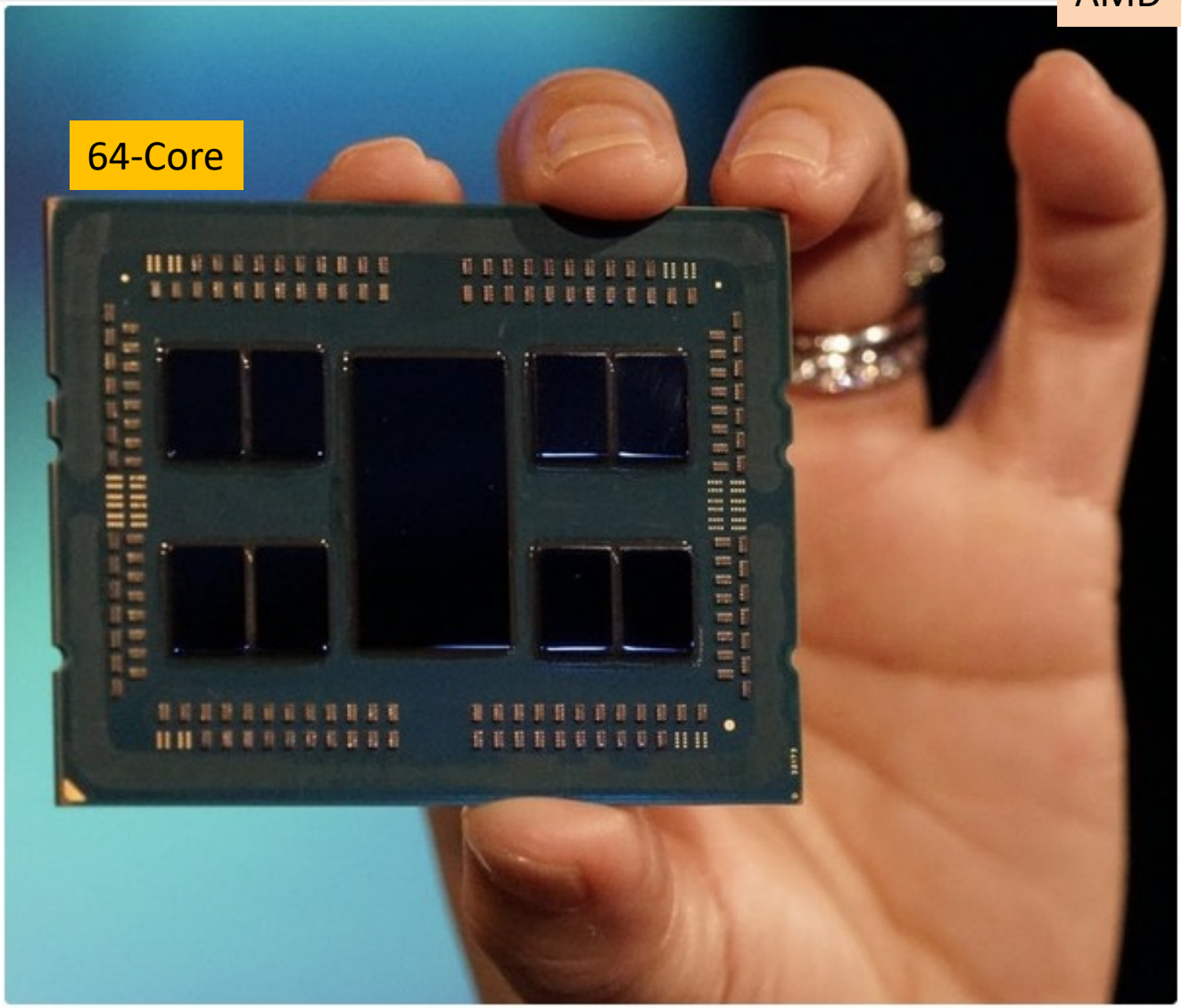
ARM also went through similar changes with the addition of Scalable Vector Extensions (SVE and SVE2) which is flexible in terms of width, ranging from 128 to 2048-bits. The RISC-V spec also provisions a 128-bit mode but no actual hardware implements it because it would be a waste of resources, die space and would add

Computer Architecture

x86 Multi-core

Multi-Cores + L2/L3

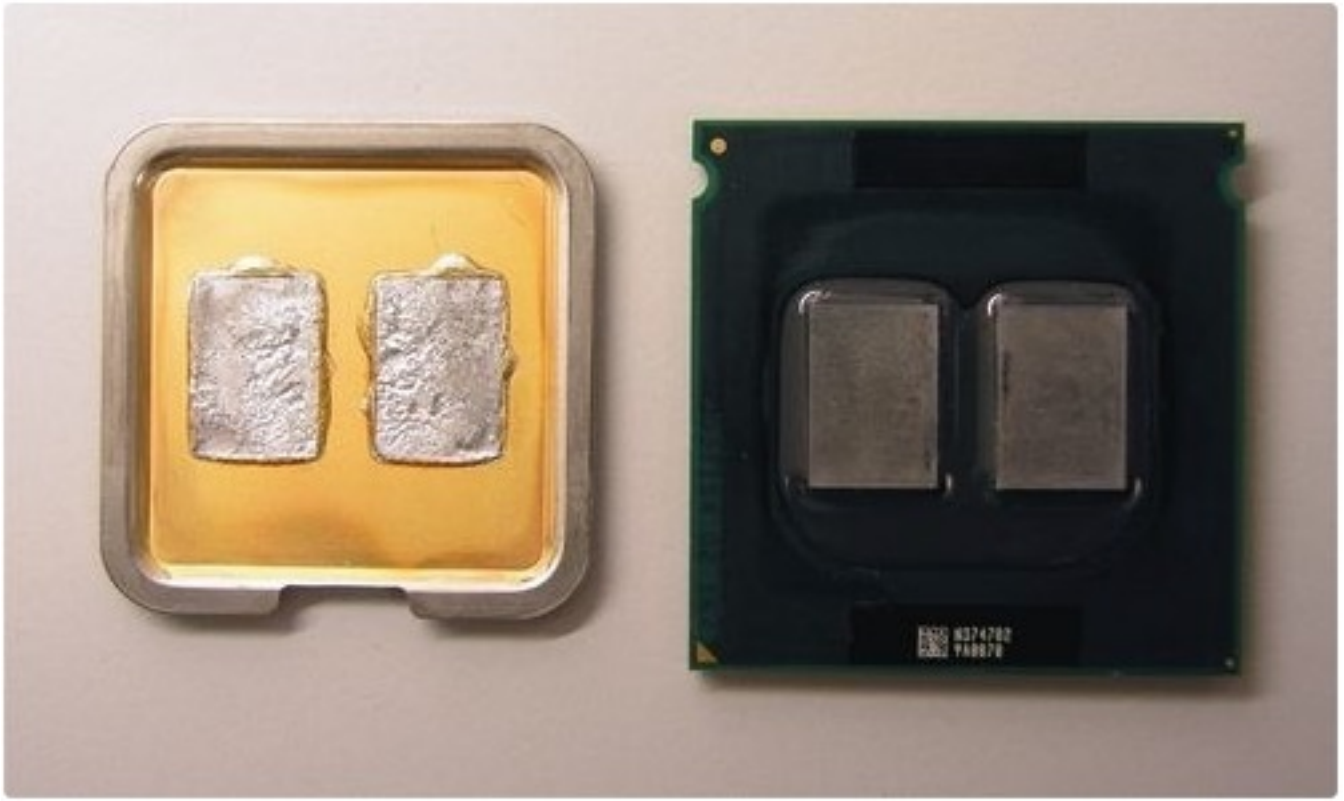
AMD



Lisa Su proudly shows off her 64-core EPYC monster at CES.

Multi-Cores + L2/L3

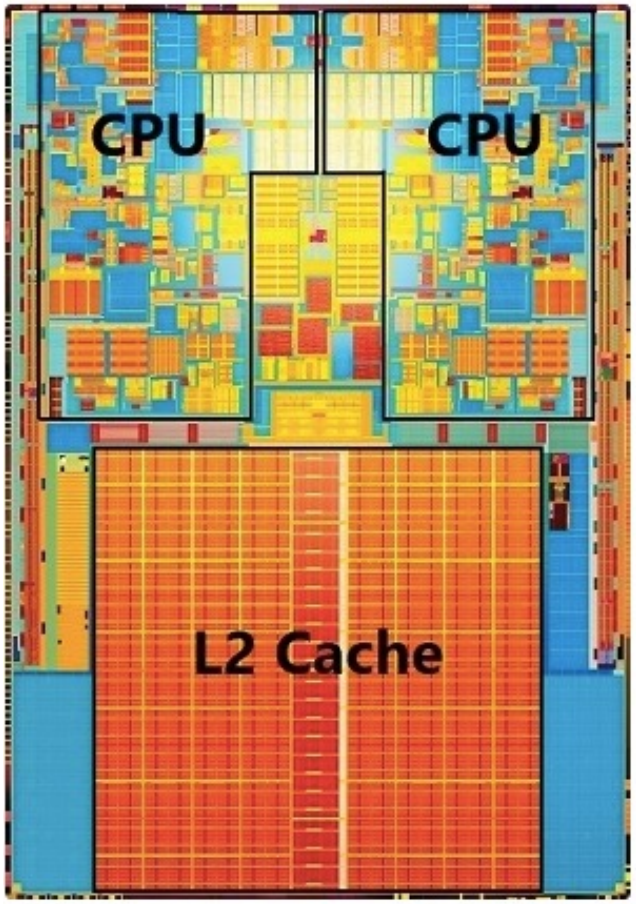
Intel

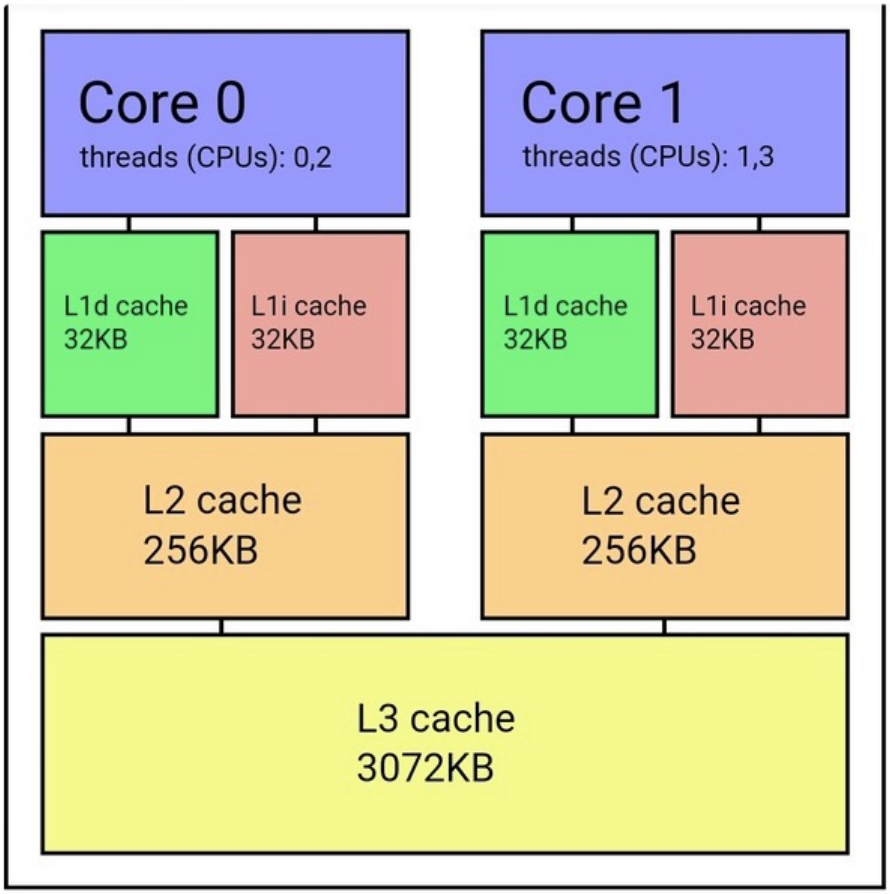


This is a delided Core 2 Quad with the two Core 2 Duo dies.

Multi-Cores + L2/L3

Intel

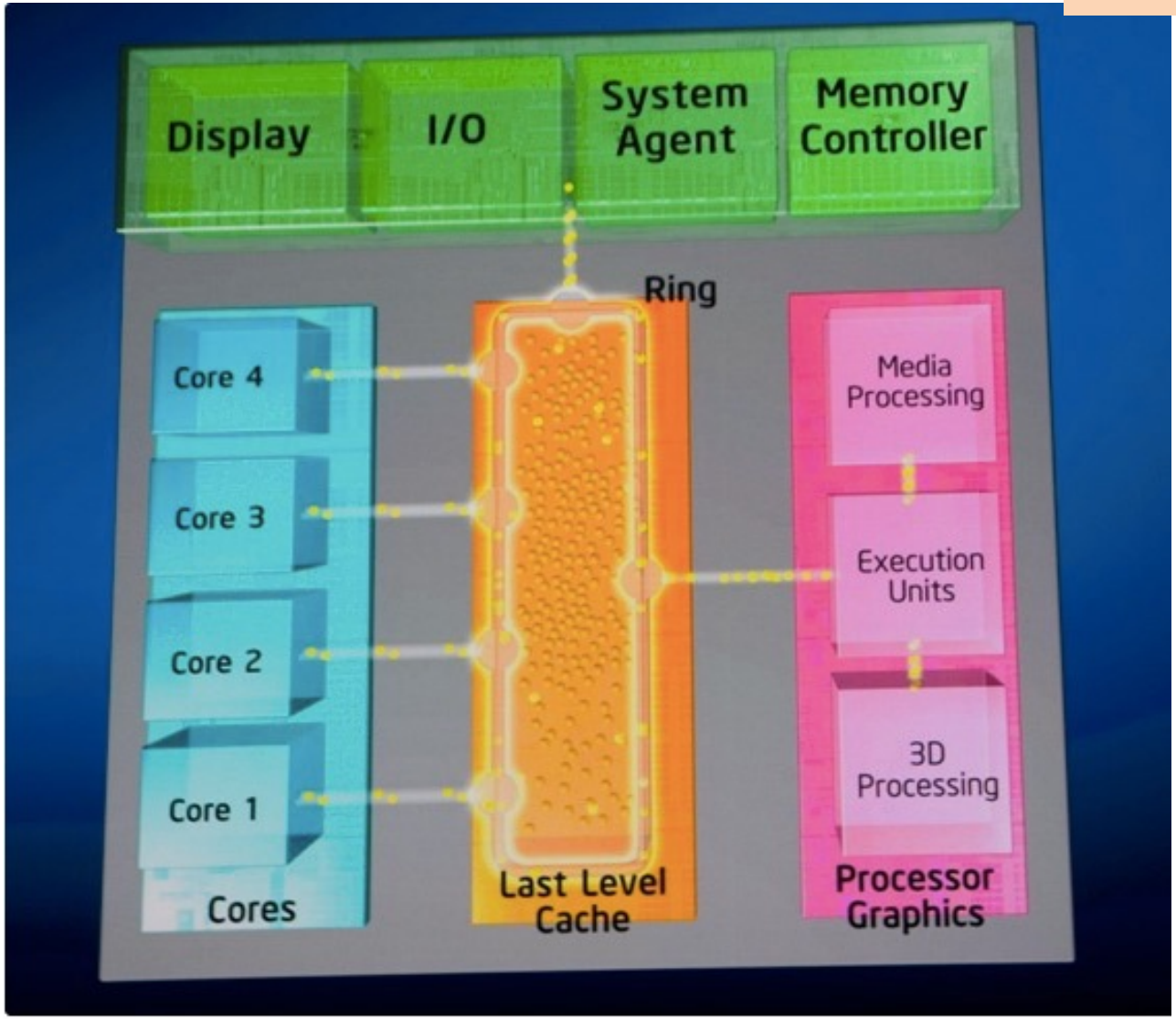




This is a late Core 2 Duo (Wolfdale), note how massive the uncore L2 is compared to the actual CPU cores. (uncore = outside the CPU)

Multi-Cores + L2/L3

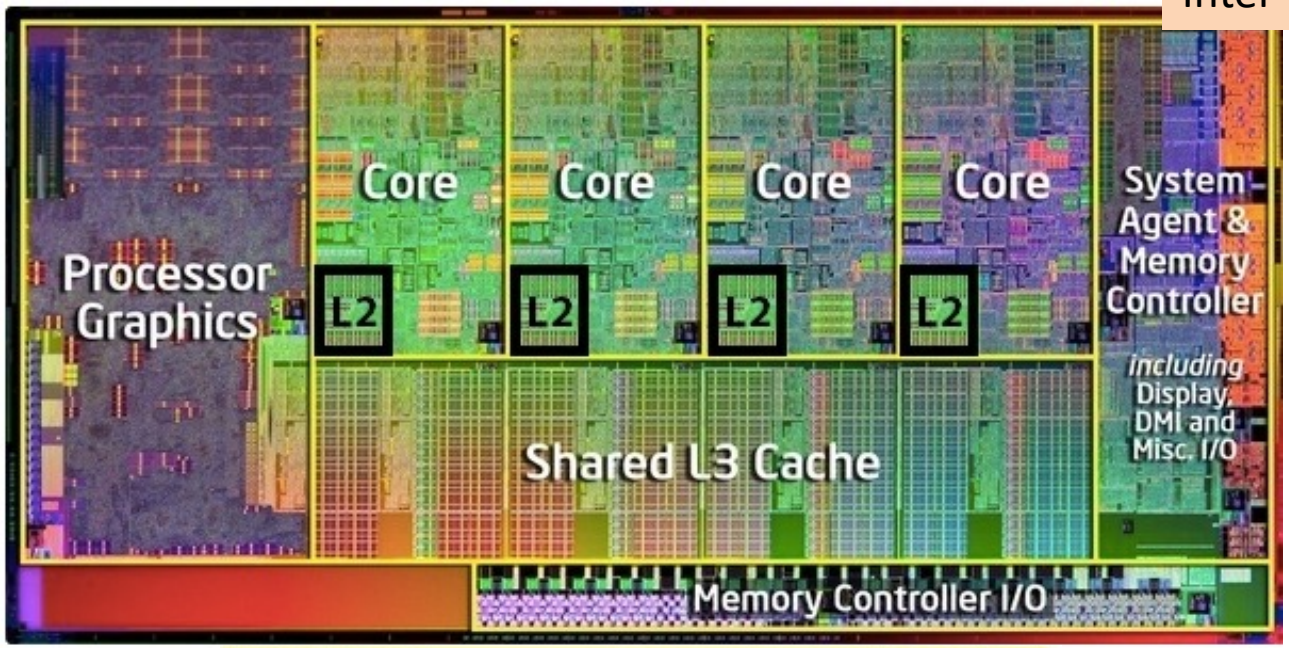
Intel

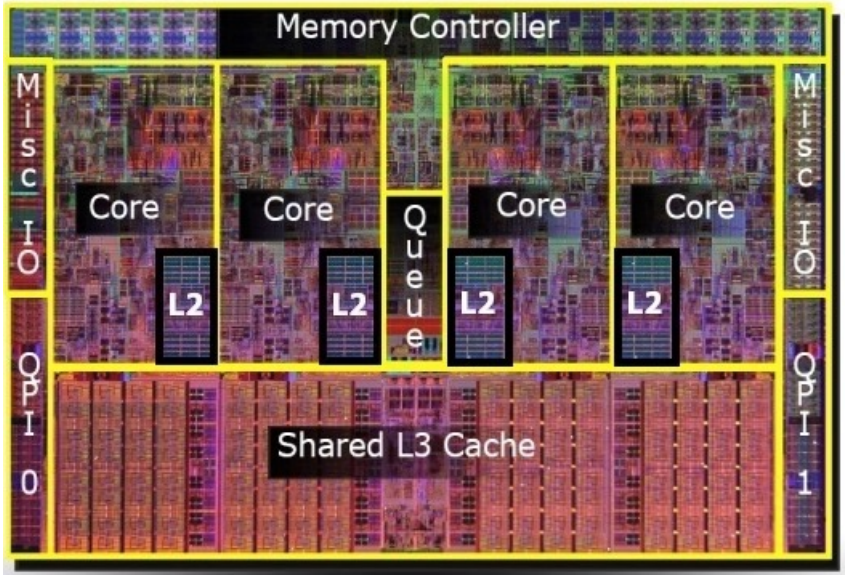


This special 'glue' remains in use today in all non-HEDT Intel CPUs.

Multi-Cores + L2/L3

Intel | Core 4

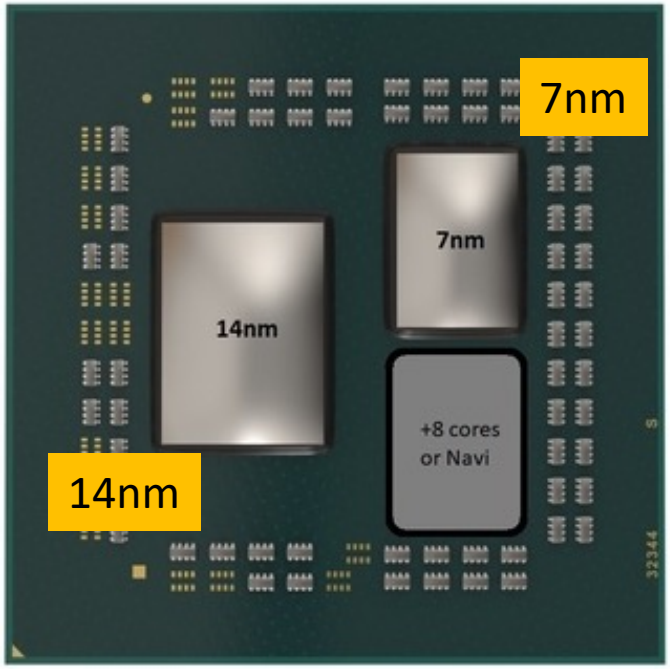




Multi-Cores + L2/L3

AMD Zen 2

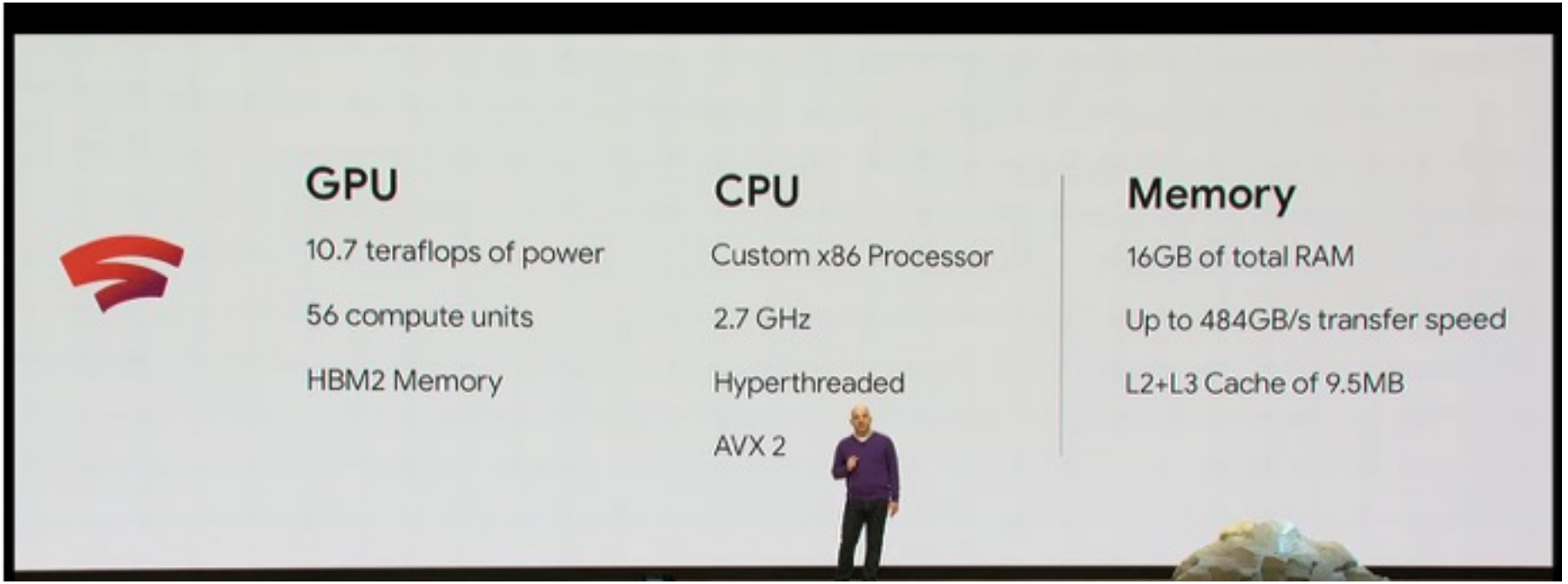
AMD is taking a creative approach with its "7nm" production by keeping the most problematic part of the CPU at 14nm and putting the scalable "core" segments on 7nm octa-core "chiplets" that can be used alone or doubled up with another octa-core chiplet or (shhh...) a Navi GPU.



AMD kills two birds with one stone thinking outside the box with this creative Zen 2 layout: silicon yield at 7nm is vastly improved by making smaller chips, and the final product can be reconfigured to produce either workhorse 16-core CPU's or excellent SOC chips with a massive Navi GPU.

Multi-Cores + L2/L3

AMD

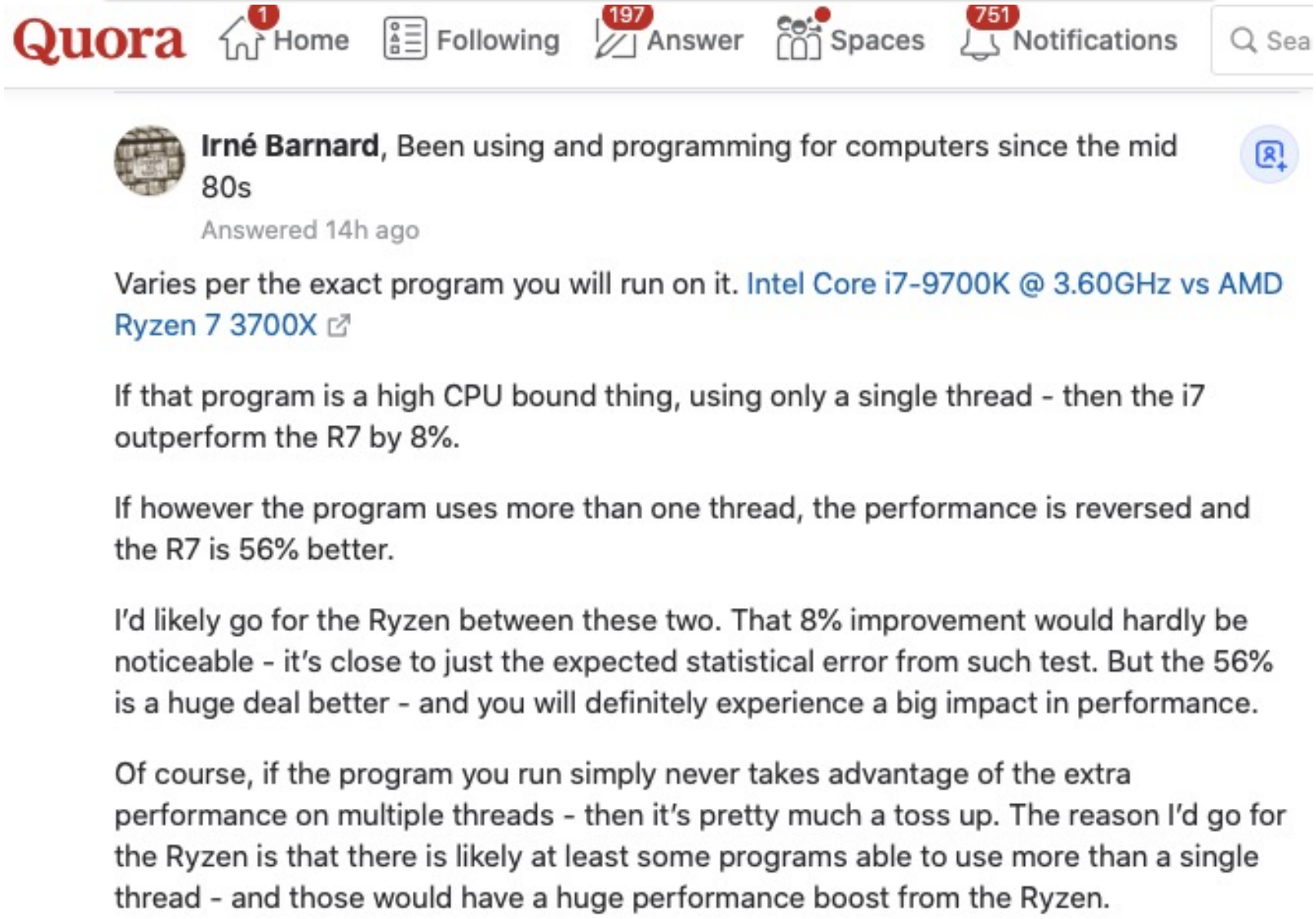


Computer Architecture

x86

Intel vs AMD

Intel vs. AMD



Quora 1 Home Following 197 Answer Spaces 751 Notifications Sea

Irné Barnard, Been using and programming for computers since the mid 80s

Answered 14h ago

Varies per the exact program you will run on it. Intel Core i7-9700K @ 3.60GHz vs AMD Ryzen 7 3700X

If that program is a high CPU bound thing, using only a single thread - then the i7 outperform the R7 by 8%.

If however the program uses more than one thread, the performance is reversed and the R7 is 56% better.

I'd likely go for the Ryzen between these two. That 8% improvement would hardly be noticeable - it's close to just the expected statistical error from such test. But the 56% is a huge deal better - and you will definitely experience a big impact in performance.

Of course, if the program you run simply never takes advantage of the extra performance on multiple threads - then it's pretty much a toss up. The reason I'd go for the Ryzen is that there is likely at least some programs able to use more than a single thread - and those would have a huge performance boost from the Ryzen.

Intel vs. AMD

Irné Barnard,

You'd have to start looking at the newer and higher class Intels to beat the Ryzen 7–3700X. At which point they're more expensive: Intel Core i7-9700K @ 3.60GHz vs AMD Ryzen 7 3700X vs Intel Core i7-10700K @ 3.80GHz vs Intel Core i9-10900K @ 3.70GHz

- i7–10700K - 25% more cost, 15% improved single thread, -16% lower multi thread.
- i9–10900K - 80% extra cost, 18% better single thread, 6% better multi thread.

Both those are more expensive to a larger degree than their performance increases over the Ryzen would suggest.

And of course, once you allow a newer Intel, you should also allow a newer Ryzen. The 5000 series is supposed to have a 19% improvement on single threads over the 3000 series. Which would suggest they'd then beat those 10th series intel CPUs in all measures. Depending on the prices they sell for in the next month this may mean Intel is really the underdog.

If you can find one, even a Ryzen 5–5600X is going to outperform all those Intel CPUs at single thread: Intel Core i7-9700K @ 3.60GHz vs AMD Ryzen 7 3700X vs Intel Core i7-10700K @ 3.80GHz vs Intel Core i9-10900K @ 3.70GHz vs AMD Ryzen 5 5600X

Around 9% better than the i9–10900K's single thread performance. While it is 19% better than the old i7–9700K (at the same cost).

AMD vs Intel i9

So it's best to say that some of the Intel i9 processors (the high end) compete with some of the AMD Threadripper processors (the low-to-mid range). Today's top i9 delivers 10–18% better performance than the Threadripper core, but an 18 core i9 just looks sad compared to a 64 core Threadripper on well threaded code.

Now, it should go without saying, but I'll say it anyway: if you're buying a Threadripper system, you're optimizing for massively multithreaded applications. Top single-threaded performance of any CPU series tends to go down as the number of cores go up. So if you have largely single/small-threaded work, a Threadripper or an i9 is a waste of money.

AMD vs Intel i9

So you're looking for an Intel processor for consumers/workstations that's more or less similar to Threadripper. Intel's answer to EPYC is Xeon, so are there any mainstream i-series processors that correspond closely? The latest high-end Xeons and Phi processors use Intel's LGA 3647 socket. This socket supports six channels of DDR4 memory, but there is no consumer version of an LGA3647 processor.

So the Intel answer to complete with Threadripper is the LGA2066 socket, also called Socket R4. There are lower-end Xeons that also use this socket. This supports DDR4 up to 256GiB on four channels, 48 PCI Express 3.0 lanes (with an additional 24 PCI Express 3.0 links in the X299 chipset). Current LGA2066 chips offer up to 18 CPU cores.

AMD vs Intel

Sockets

The current Threadripper sTRX4 socket is an LGA design with 4094 pins.. they're kind of mindboggling to look at. Modern EPYC processors use a mechanically identical but electrically tweaked Socket SP3r3 socket. Older chips use TR4 and SP3r2 sockets, respectively. The EPYC series and Threadripper Pro actually support up to 2TiB DDR4 DRAM on eight channels and 128 PCI Express 4.0 links. Today's standard Threadrippers support 128 GiB or 256GiB DDR4 DRAM on four channels, with up to 88 PCI Express 4.0 links, and of course, up to 64 CPU cores on all three platforms.

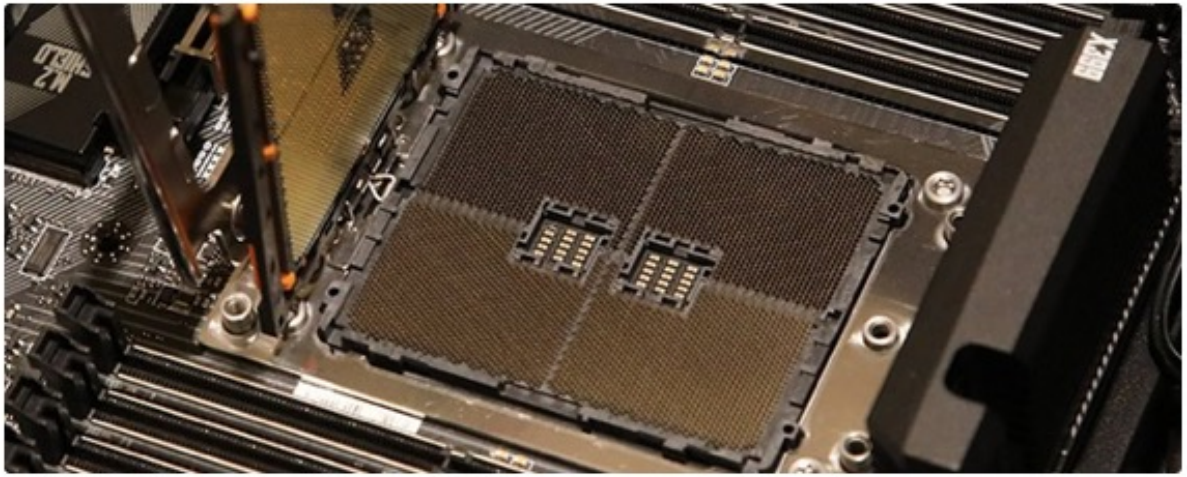
So you're looking for an Intel processor for consumers/workstations that's more or less similar to Threadripper. Intel's answer to EPYC is Xeon, so are there any mainstream i-series processors that correspond closely? The latest high-end

AMD vs Intel

Sockets

AMD Threadrippers are essentially consumer versions of the EPYC line of server processors. There are differences, but the basic idea is the same: four DDR4 memory channels, high core count, etc. My aging Threadripper system has "only" sixteen processor cores and the usual four 64-bit DDR4 memory channels.

Like all Ryzen family processors, Threadrippers are made of multiple "chiplets" connected by ultra high speed Infinity Fabric links. Each chiplet so far contains up to eight processor cores. The central chip in generation 2 and later Threadrippers is a I/O chip, supporting PCI Express links, that sort of thing.



The current Threadripper sTRX4 socket is an LGA design with 4094 pins.. they're kind of mindboggling to look at. Modern EPYC processors use a mechanically identical but electrically tweaked Socket SP3r3 socket. Older chips use TR4 and SP3r2 sockets, respectively. The EPYC series and Threadripper Pro actually support up to 2TiB DDR4 DRAM on eight channels and 128 PCI Express 4.0 links. Today's standard Threadrippers support 128 GiB or 256GiB DDR4 DRAM on four channels, with up to 88 PCI Express 4.0 links, and of course, up to 64 CPU cores

AMD vs Intel

Processor Pricing by Family	AMD	Intel
Threadripper - Cascade Lake-X	$900- $3,750	$800 - $1,000 ($2,999)
AMD Ryzen 9 - Intel Core i9	$434 - $739	$459 - $505
AMD Ryzen 7 - Intel Core i7	$294 -$339	$300 - $370
AMD Ryzen 5 - Intel Core i5	$149 - $249	$125 - $200
AMD Ryzen 3 - Intel Core i3	$95 - $120	$78 - $173

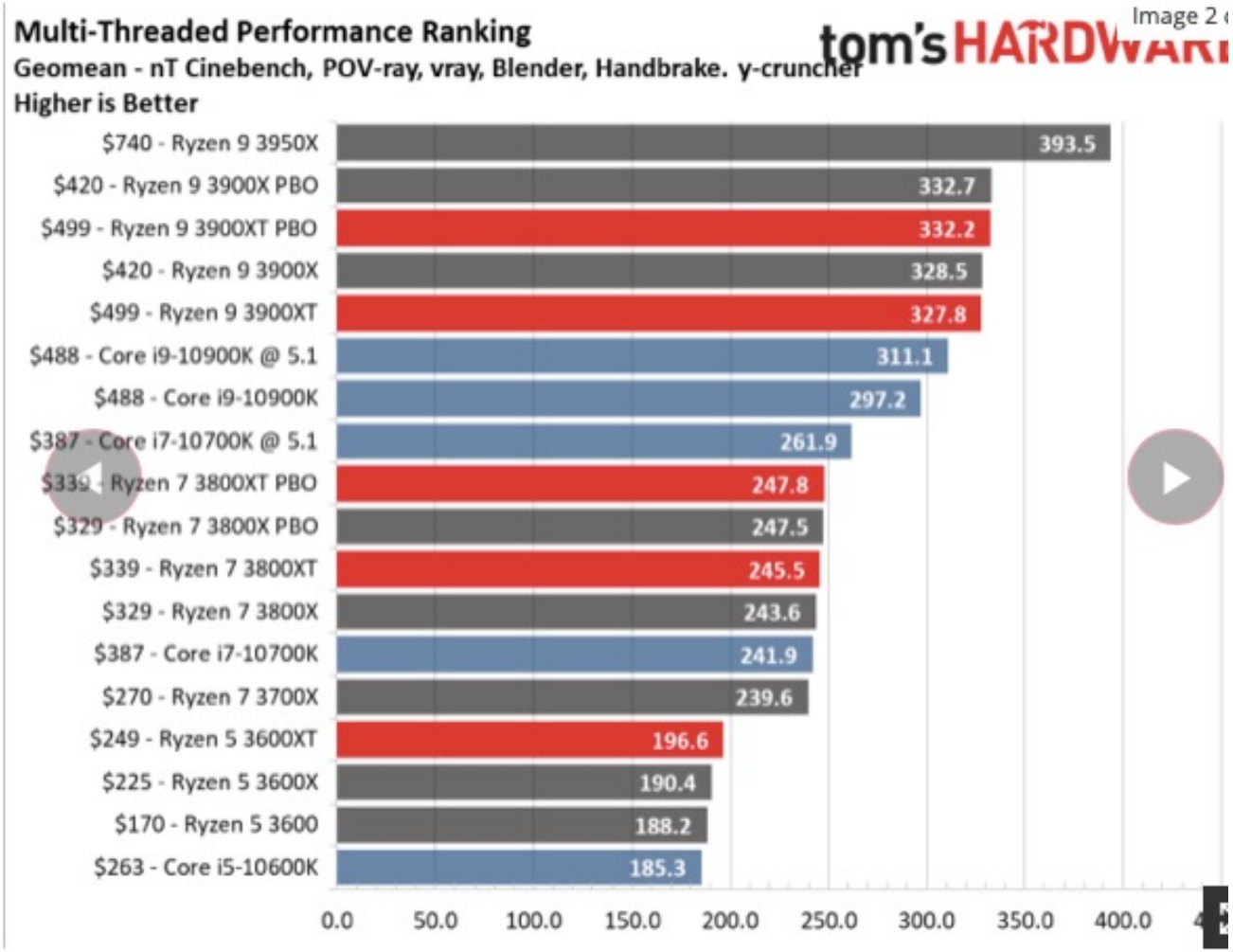
AMD vs Intel

Intel and AMD CPU Multi-Threaded Performance

	Multi-Threaded App Score	Architecture	Cores/Threads	Base/Boost	TDP
Threadripper 3990X	100.0%	Zen 2	64/128	2.9 / 4.3 GHz	280W
Threadripper 3970X	83.76%	Zen 2	32/64	3.7 / 4.5 GHz	280W
Threadripper 3960X	72.04%	Zen 2	24/48	3.8 / 4.5 GHz	280W
Xeon W-3175X	69.92%	Skylake	28/56	3.1 / 4.3 GHz	225W
Ryzen 9 3950X	53.48%	Zen 2	16/32	3.5 / 4.7 GHz	105W
Core i9-10980XE	52.75%	Cascade Lake-X	18/36	3.0 / 4.8 GHz	165W
Core i9-9980XE	52.14%	Skylake	18/36	4.4 / 4.5 GHz	165W
Threadripper 2990WX	48.00%	Zen+	32/64	3.0 / 4.2 GHz	250W
Ryzen 9 3900X	44.64%	Zen 2	12/24	3.8 / 4.6 GHz	105W
Ryzen 9 3900XT	44.55%	Zen 2	12/24	3.8 / 4.7 GHz	105W
Threadripper 2970WX	44.26%	Zen +	24/48	3.0 / 4.2 GHz	250W

AMD vs Intel

AMD vs Intel Productivity and Content Creation Performance



(Image credit: Tom's Hardware)

AMD vs Intel: Mid

AMD vs Intel CPUs Mid-Range and Budget Specs and Pricing

Mainstream	MSRP/Retail	Cores / Threads	Base / Boost GHz	$-Per-Core/Thread (MSRP)	L3 Cach
Core i5-10600K / KF	$262 (K) / $237 (KF)	6 / 12	4.1 / 4.8	~$44 / ~$22	12
Ryzen 5 3600XT	$249	6 / 12	3.8 / 4.5	$41.5 / $41.5	32
Ryzen 5 3600X	$249 / $205	6 / 12	3.8 / 4.4	~$41.5 / ~$21	32
Core i5-10600	$213	6 / 12	3.3 / 4.8	~$36 / ~$23	12
Ryzen 5 3600	$199 / $175	6 / 12	3.6 / 4.2	~$33 / ~$17	32
Core i5-10500	$192	6 / 12	3.1 / 4.5	~$32 / ~$16	12
Core i5-10400 / F	$182 / $157 (F)	6 / 12	2.9 / 4.3	~$26 / ~$13	12

AMD vs Intel: High-end

HEDT

AMD vs Intel CPUs High End Specs and Pricing

High End Mainstream	MSRP/Retail	Cores / Threads	Base / Boost GHz	$-Per-Core (MSRP)	L3 Cache	TDP
Ryzen 9 3950X	$749 / $739	16 / 32	3.5 / 4.7	$46	64	105W
Ryzen 9 3900XT	$499	12 / 24	3.8 / 4.7	$42	64	105W
Ryzen 9 3900X	$499 / $434	12 / 24	3.8 / 4.6	$42	64	105W
Core i9-10900K / KF	$488 (K) / $472 (KF)	10 / 20	3.7 / 5.3	~$49	20	125W
Core i9-10900 / F	$439 / $422 (F)	10 / 20	3.7 / 5.2	~$44	20	65W
Core i7-10700K / KF	$374 (K) / $349 (KF)	8 / 16	3.8 / 5.1	~$47 / ~$24	16	125W
Ryzen 7 3800XT	$399	8 / 16	3.9 / 4.7	$50 / $50	32	105W

AMD vs Intel: High-end

HEDT

AMD vs Intel CPUs HEDT Specs and Pricing

High End Desktop (HEDT)	MSRP / Retail	Cores / Threads	Base / Boost GHz	L3 Cache	TDP	PCIe	Me
Threadripper 3990X	**$3,990 / $3,750**	**64 / 128**	**2.9 / 4.3**	**256**	**280W**	**72 Usable Gen4**	**Qu DI 32**
Intel W-3175X	$2,999 / N/A	28 / 56	3.1 / 4.8	38.5	255W	48 Gen3	Six Ch DI 26
Threadripper 3970X	**$1,999 / $1,899**	**32 / 64**	**3.7 / 4.5**	***128**	**280W**	**72 Usable Gen4**	Qu DI 32
Threadripper 3960X	**$1,399 / $1,399**	**24 / 48**	**3.8 / 4.5**	***128**	**280W**	**72 Usable Gen4**	Qu DI 32
Xeon W-3265	$3,349 / N/A	24 / 48	2.7 / 4.6	33	205W	64 Gen3	Six Ch DI 29
Core i9-10980XE	$979 / $1,099	18 / 36	3.0 / 4.8	24.75	165W	48 Gen3	Qu DI 29

The high end desktop (HEDT) is the land of creative prosumers with fire-breathing multi-core monsters for just about every need. Intel has long enjoyed the uncontested lead in this segment, but while AMD's first-gen Threadripper lineup disrupted the status quo, the Threadripper 3000 lineup destroyed it.

AMD vs Intel

TORONTO

Acer Swift 3 - INTEL vs AMD!

AMD vs Intel

TORONTO

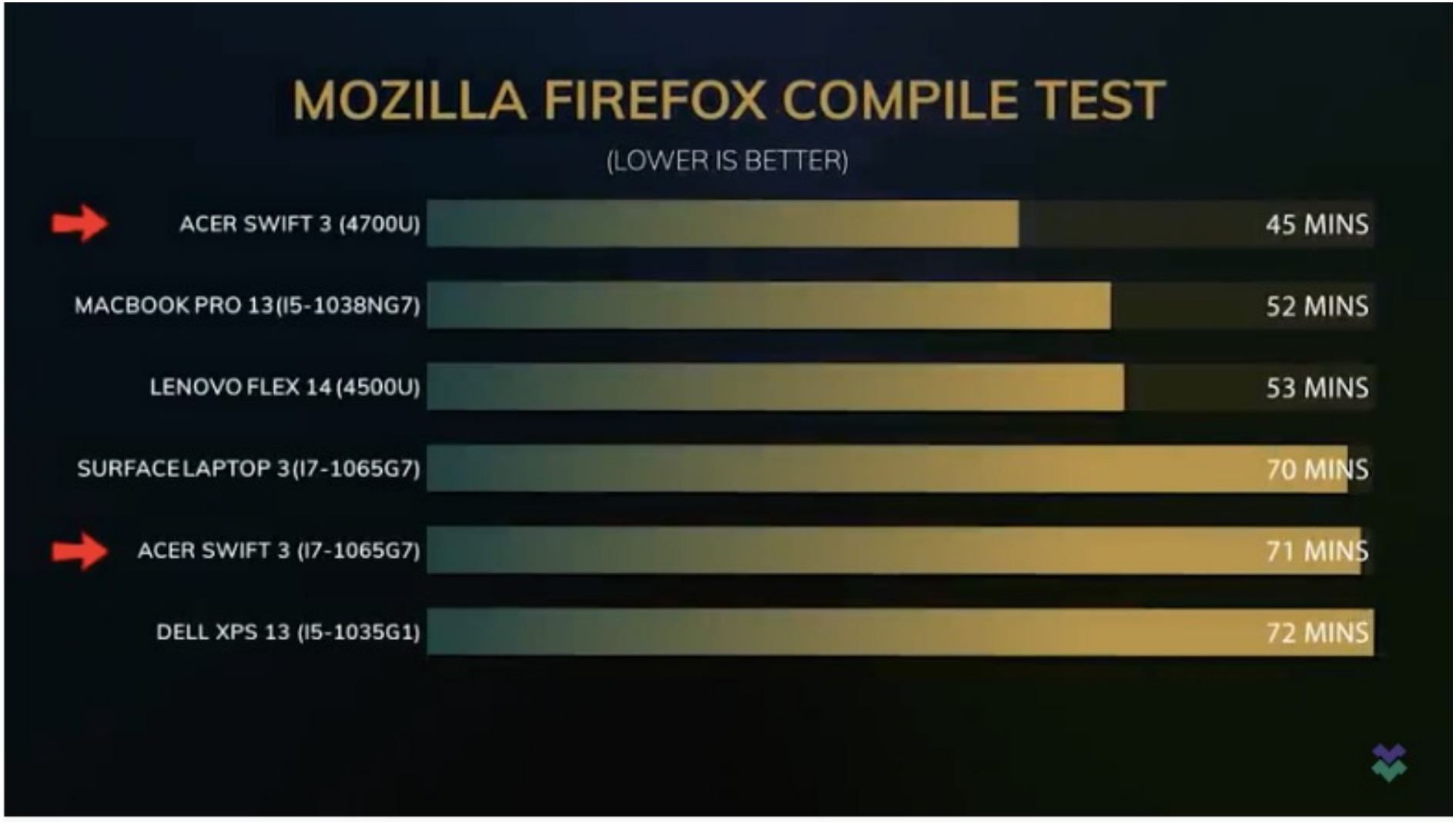
Acer Swift 3 - INTEL vs AMD!

AMD vs Intel

TORONTO

Acer Swift 3 - INTEL vs AMD!

AMD vs Intel



TORONTO

Acer Swift 3 - INTEL vs AMD!

AMD vs Intel: Gaming

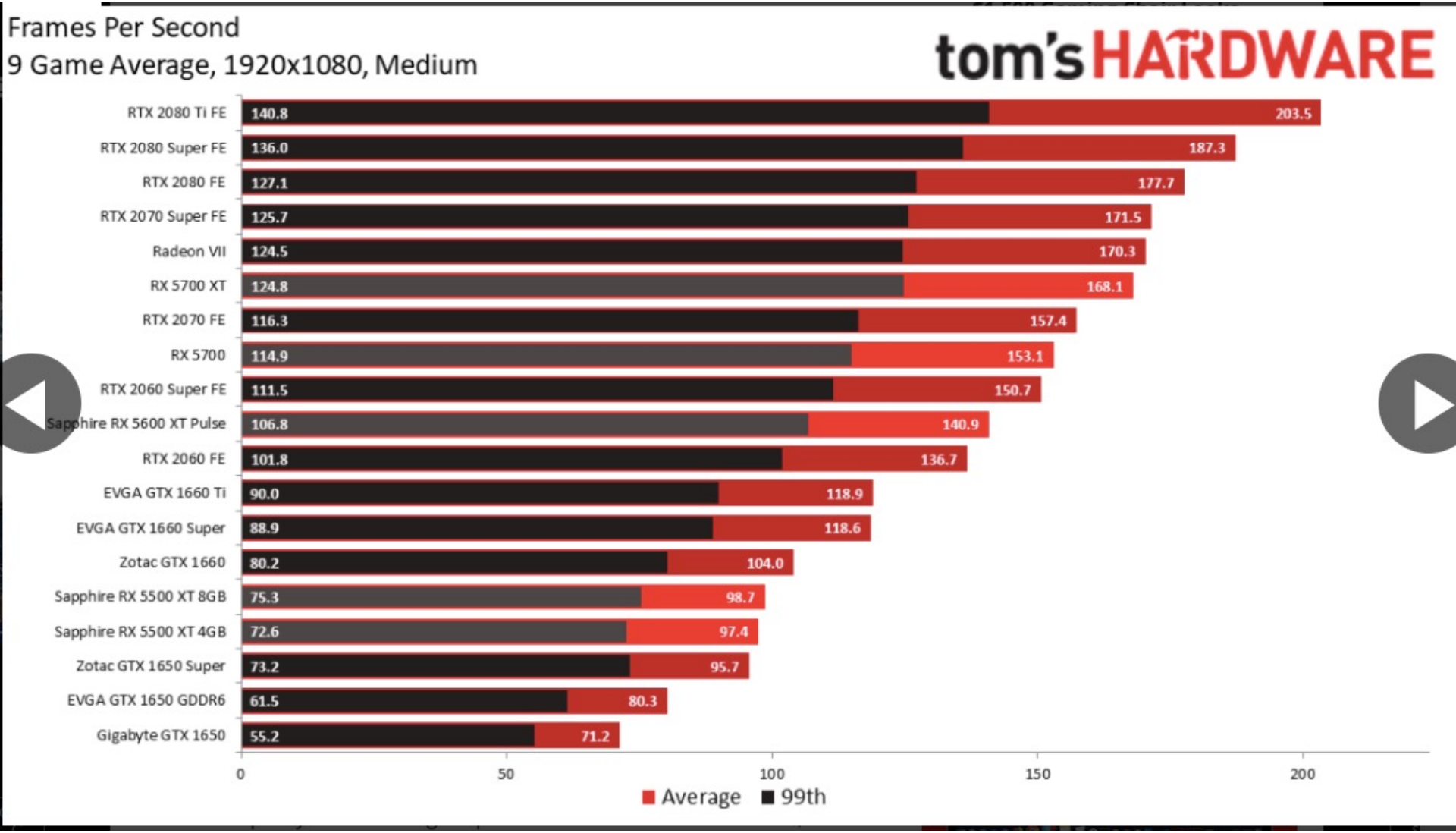
Intel and AMD CPU Gaming Hierarchy

	Gaming Score	CPU	Cores/Threads	Base/Boost	TDP	Buy
Intel Core i9-10900K	100	Comet Lake	10/20	3.7/5.3	125W	Intel Core i9-9900K
Intel Core i9-9900KS	99.83%	Coffee Lake-R	8/16	4.0 / 5.0 GHz	127W	Intel Core i9-9900KS
Intel Core i9-10980XE	98.92%	Cascade Lake-X	18/36	3.0 / 4.8 GHz	165W	Intel Core i9-10980XE
Intel Core i7-10700K	97.58%	Comet Lake	8/16	3.8 / 5.1 GHz	125W	Intel Core i7-10700K
Intel Core i7-9700K	97.18%	Coffee Lake-R	8/8	3.6 / 4.9 GHz	95W	Intel Core i7-9700K
Intel Xeon W-3175X	96.82%	Skylake	28/56	3.1 / 4.3 GHz	225W	Intel Xeon W-3175X
AMD Threadripper 3970X	96.59%	Zen 2	32/64	3.7 / 4.5 GHz	280W	AMD Threadripper 3970X
AMD Threadripper 3960X	96.53%	Zen 2	24/48	3.8 / 4.5 GHz	280W	AMD Ryzen Threadripper 3960X
Intel Core i9-9900K / F	96.25%	Coffee Lake-R	8/16	3.6 / 5.0 GHz	95W	Intel Core i9-9900K
AMD Threadripper 3990X	96.16%	Zen 2	64/128	2.9 / 4.3 GHz	280W	AMD Ryzen Threadripper 3990X
AMD Ryzen 9 3900XT	95.01%	Zen 2	12/24	3.8 / 4.7 GHz	105W	AMD Ryzen 9 3900XT

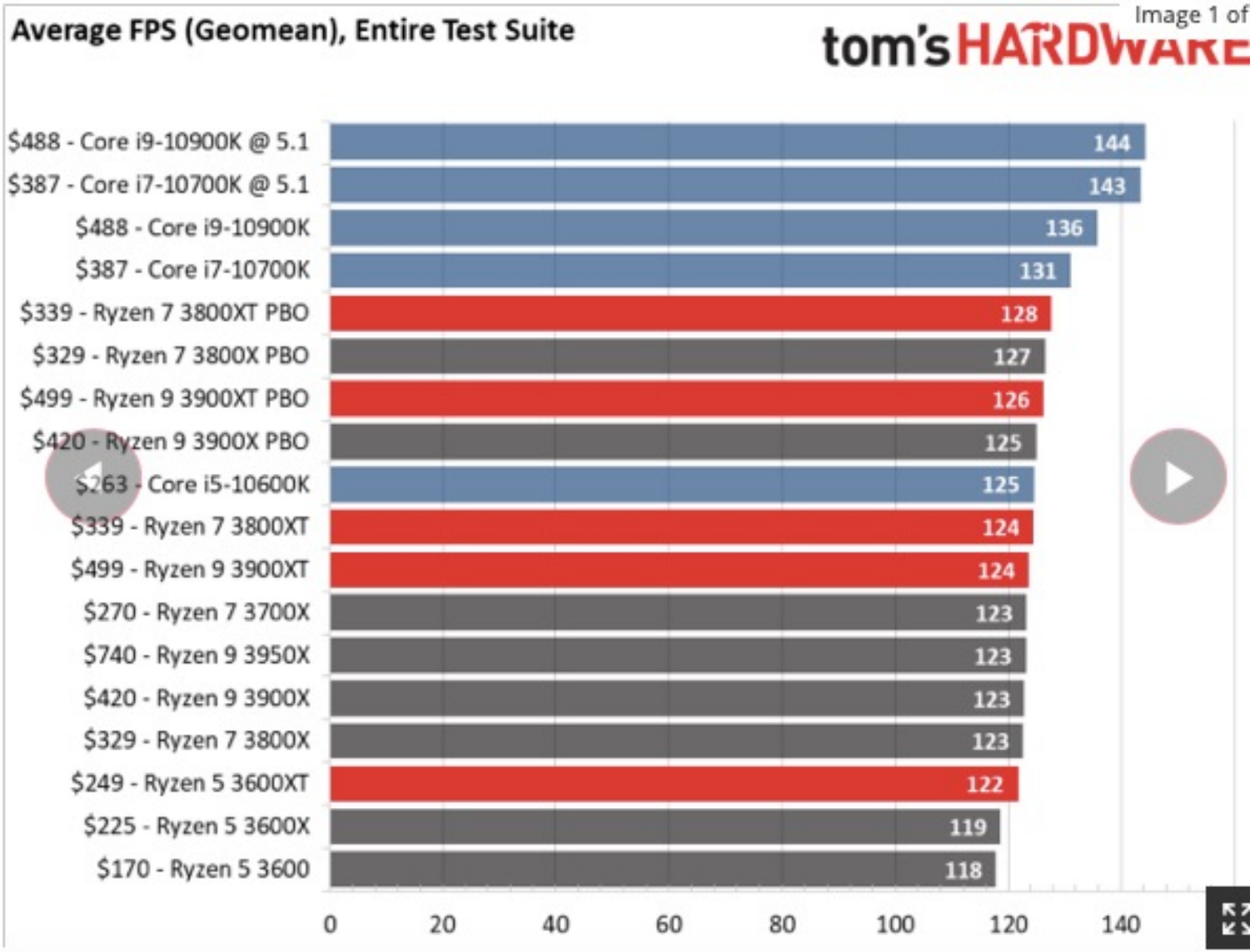
CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP122

AMD vs Intel: Gaming

DR JEFF SOFTWARE
INDIE APP DEVELOPER

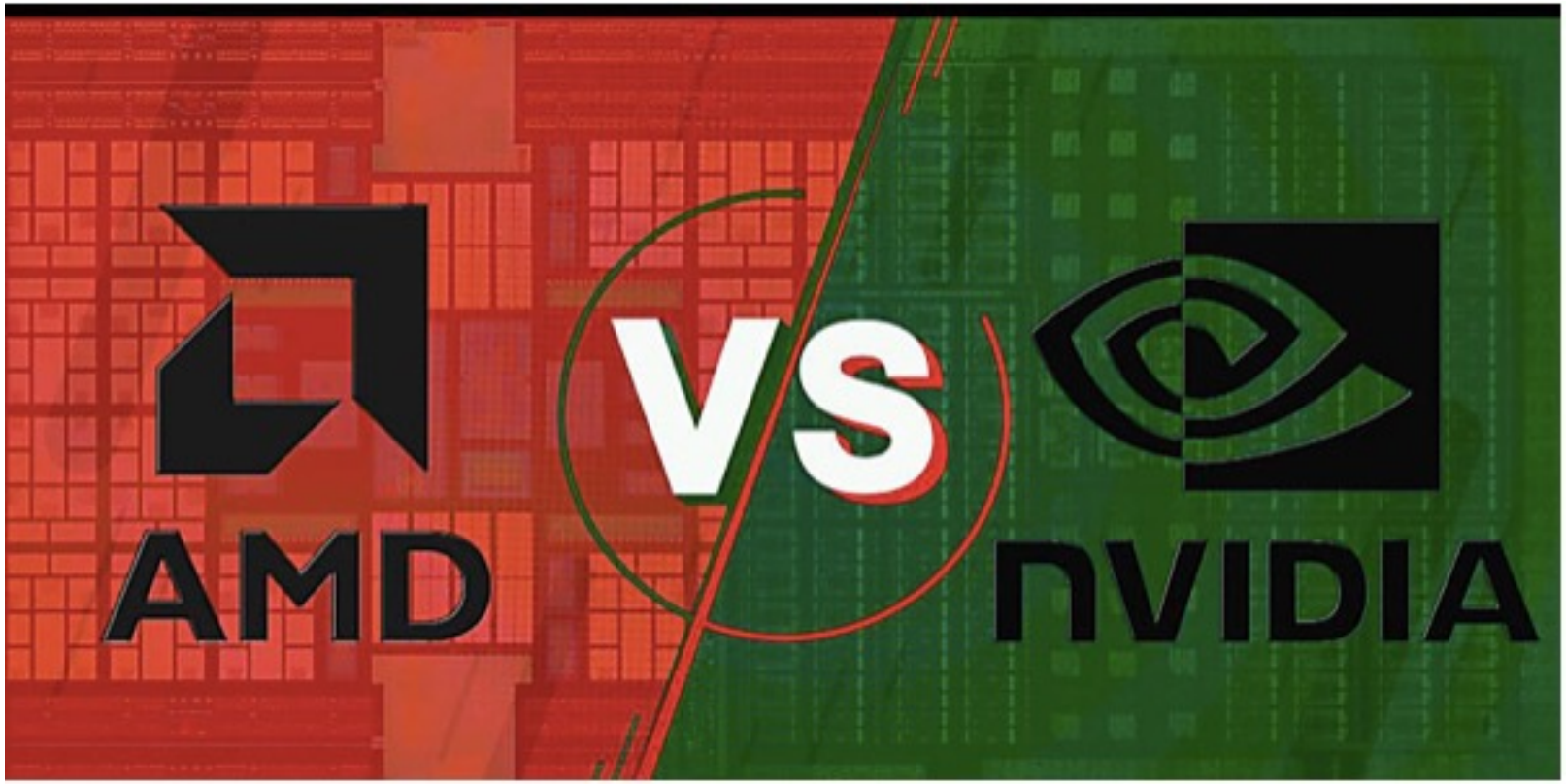


AMD vs Intel: Gaming



(Image credit: Tom's Hardware)

AMD vs Nvidia GPU's



AMD vs Nvidia: Who Makes the Best GPUs?

In the AMD vs Nvidia competition to make the fastest and most efficient GPUs possible, there can be only one winner. We look at performance, features, drivers, …

Computer Architecture

x86 Assembly Language

- See separate slide set "MCS-8 Assembly"

i8086 Code

John Stephenson, Analyst programmer

magnetic tape reels.

Answered 9h ago

Several methods:

Set low byte A = all 1's or all 0's

```
1          MOV   AL,FF
2
3
4          OR    AL,FF
5
6
7          XOR   AL,AL
8          NOT   AL
```

MOV

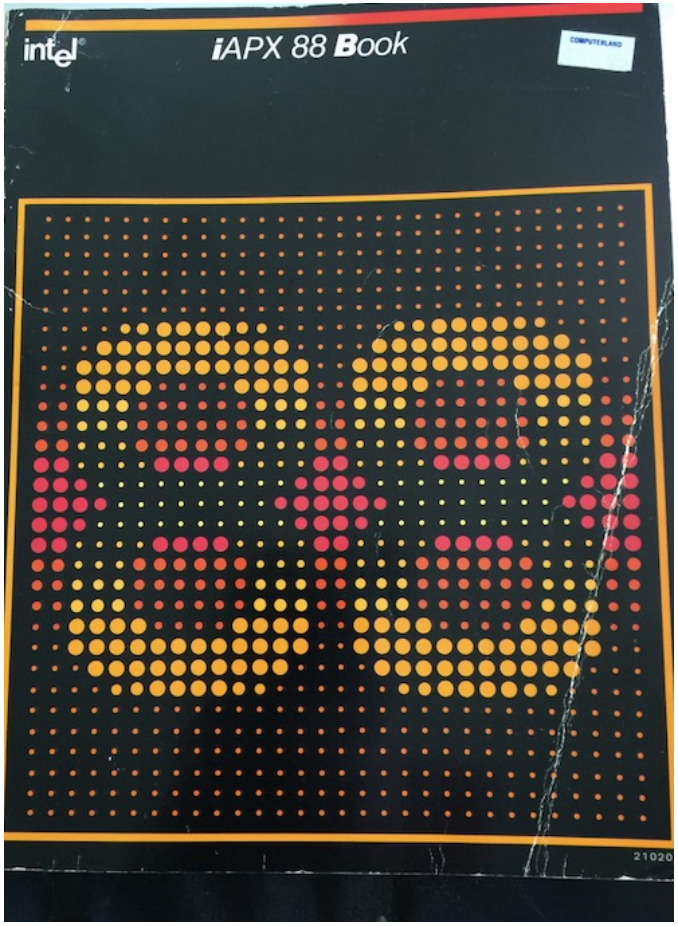
AH	AL

AX

UPPER CASE!

ISA

Intel x86



Intel x86: i8088

Bus Interface

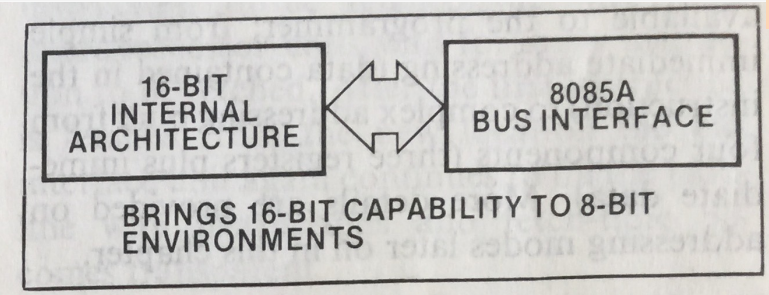


Figure 1-2. 8088 CPU

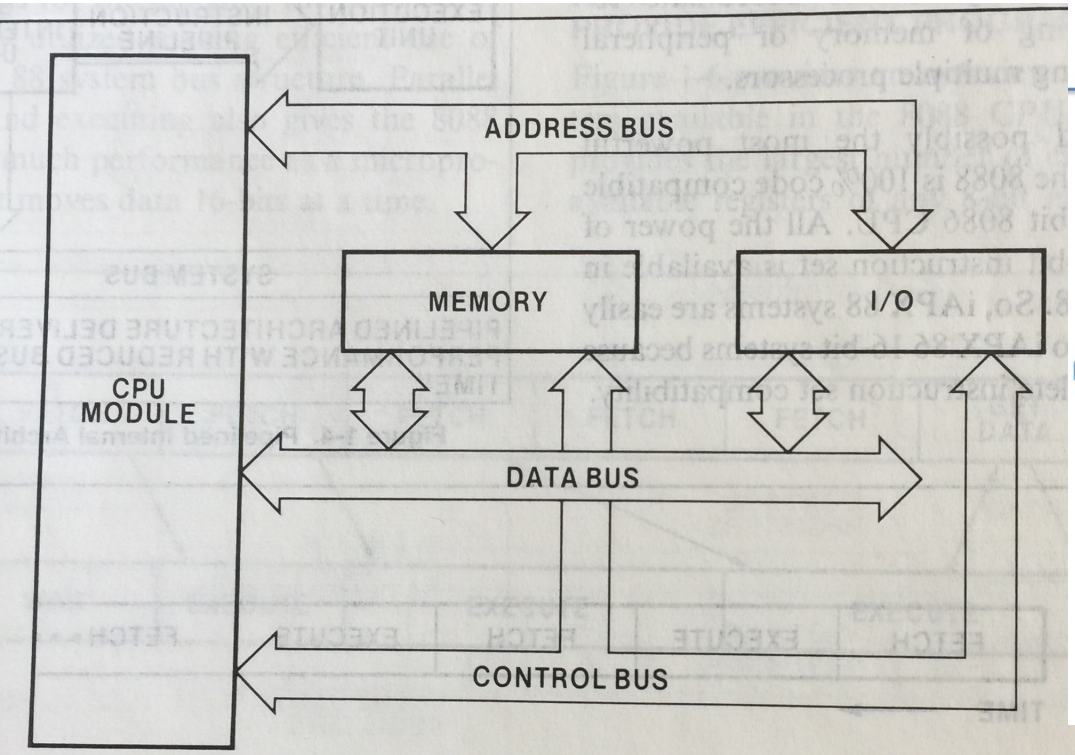
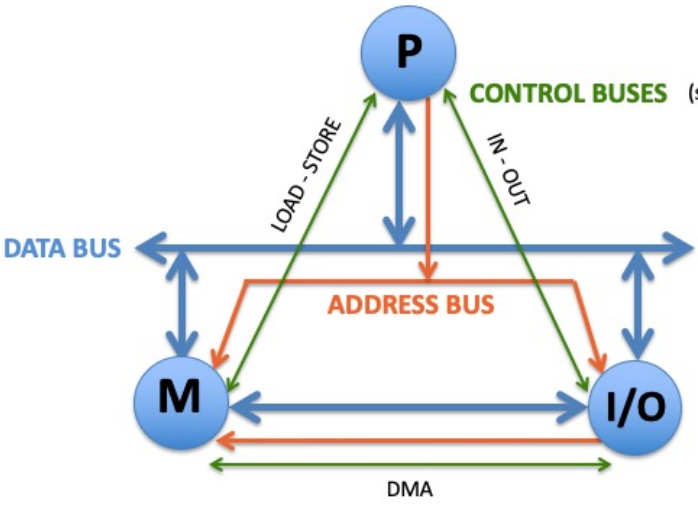


Figure 1-1. Microcomputer Block Diagram

System Buses

NON-Multiplexed **A+D** Buses



NON-MULTIPLEXED BUSES

Intel x86: i8088

Bus Interface

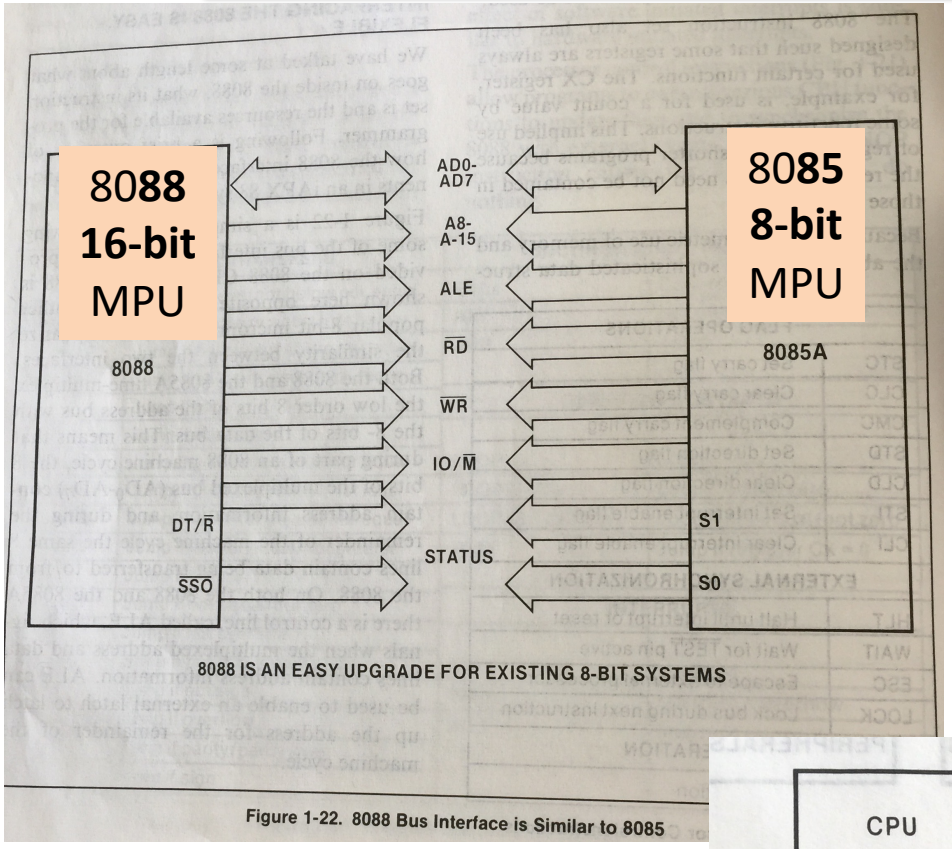


Figure 1-22. 8088 Bus Interface is Similar to 8085

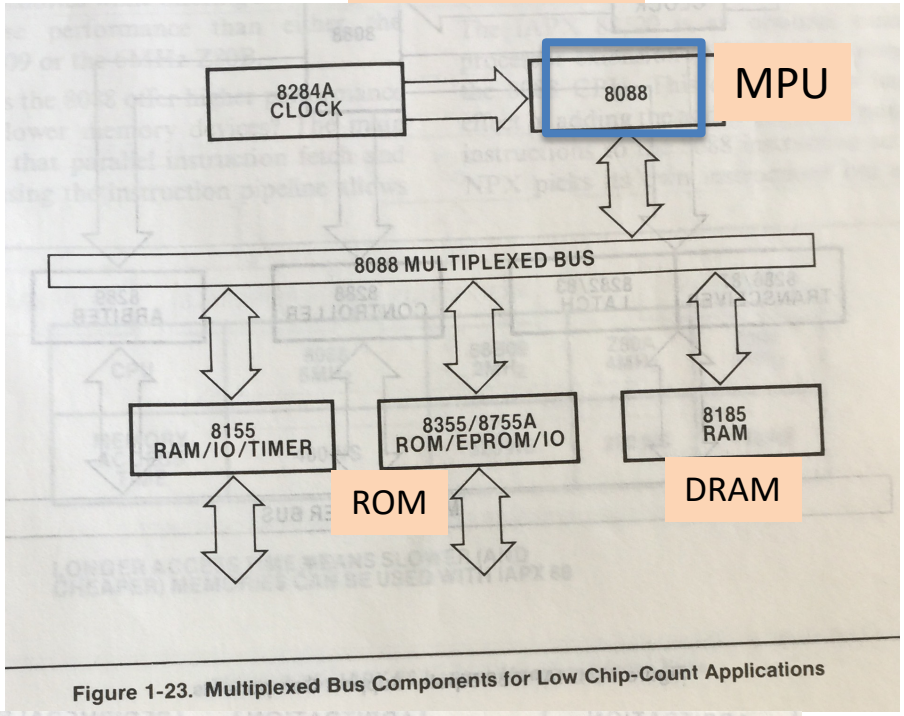


Figure 1-23. Multiplexed Bus Components for Low Chip-Count Applications

CPU	8088 5MHz	68B09 2MHz	Z80A 4MHz	Z80B 6MHz
MEMORY ACCESS TIME	460 NS	320 NS	250 NS	140 NS

LONGER ACCESS TIME MEANS SLOWER (AND CHEAPER) MEMORIES CAN BE USED WITH iAPX 88

Figure 1-25. iAPX 88 Longer Memory Access Time

Intel x86: i8088

Bus Interface

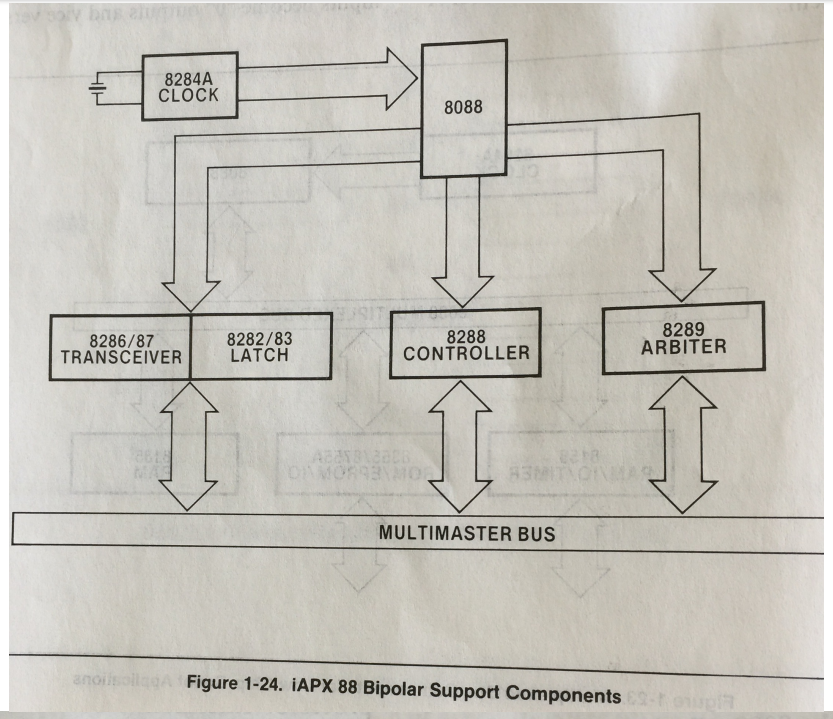
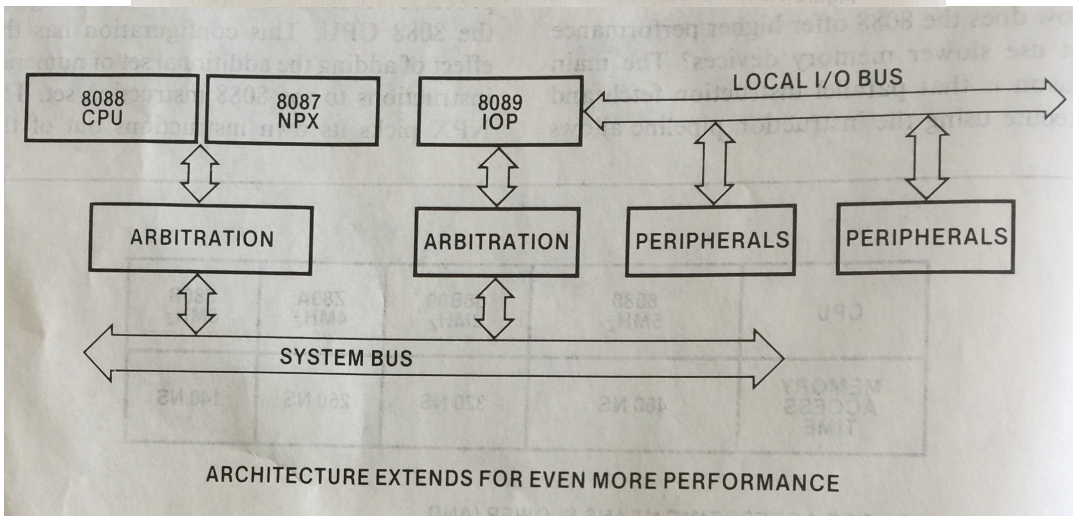


Figure 1-24. iAPX 88 Bipolar Support Components



ARCHITECTURE EXTENDS FOR EVEN MORE PERFORMANCE

Intel x86: i8088

Pipeline

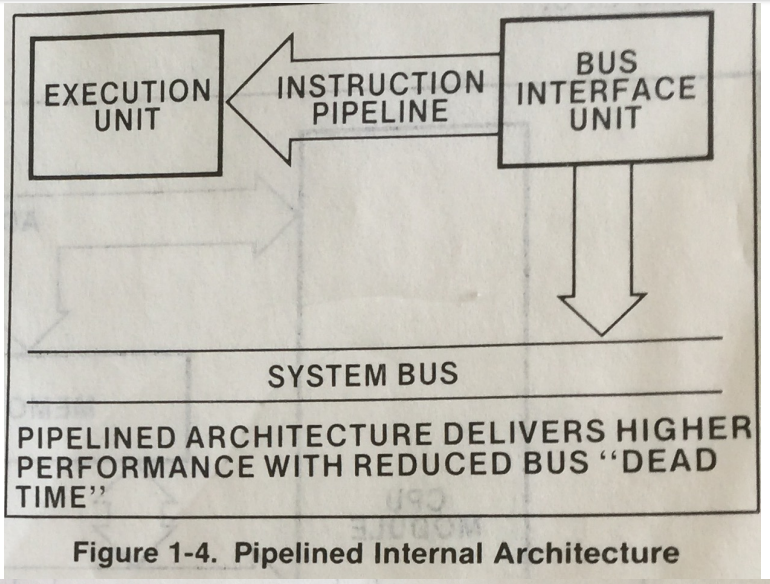


Figure 1-4. Pipelined Internal Architecture

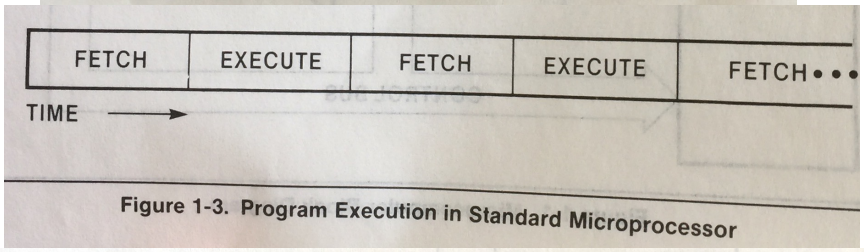


Figure 1-3. Program Execution in Standard Microprocessor

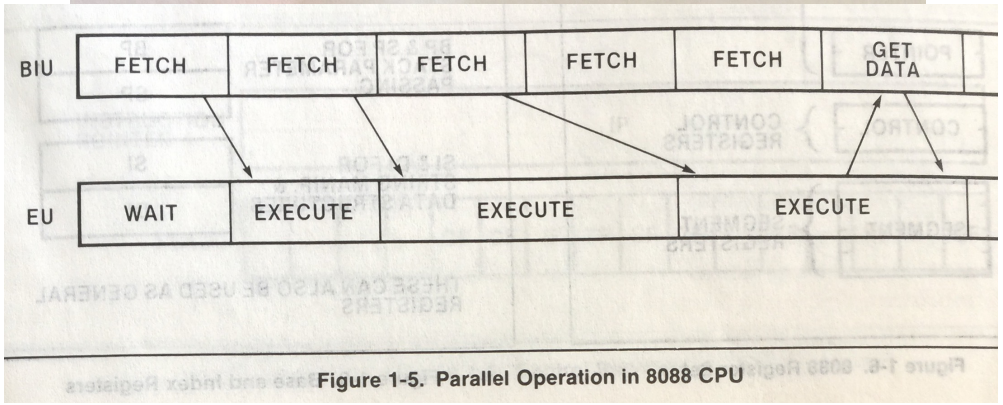


Figure 1-5. Parallel Operation in 8088 CPU

Intel x86: i8088

Registers

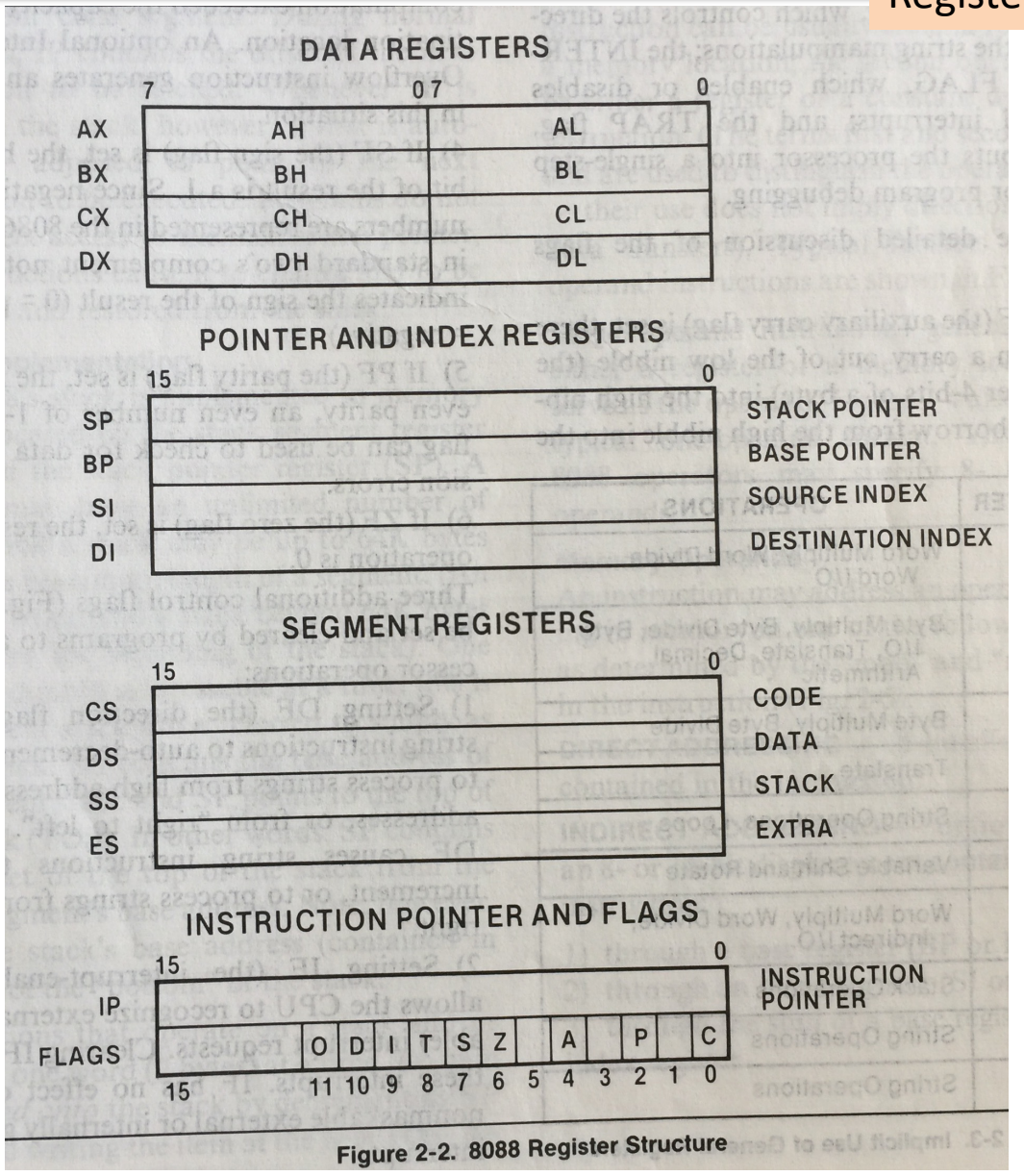


Figure 2-2. 8088 Register Structure

Intel x86: i8088

Registers

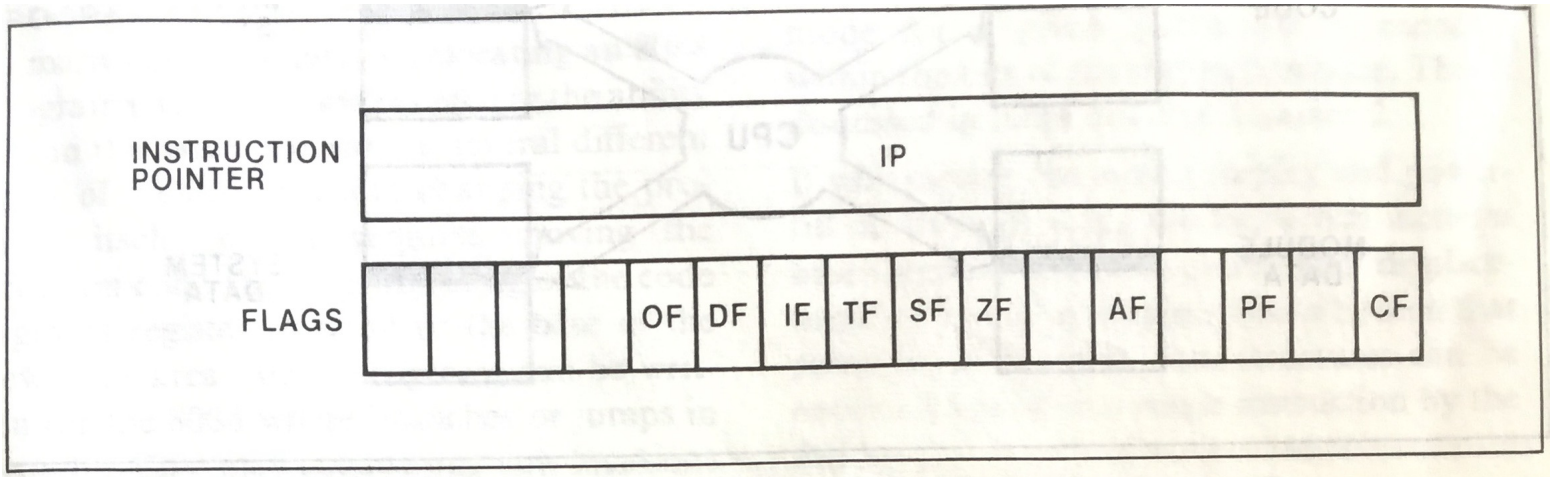


Figure 1-9. Control Registers

Intel x86: i8088

Registers

REGISTER	OPERATIONS
AX	Word Multiply, Word Divide, Word I/O
AL	Byte Multiply, Byte Divide, Byte I/O, Translate, Decimal Arithmetic
AH	Byte Multiply, Byte Divide
BX	Translate
CX	String Operations, Loops
CL	Variable Shift and Rotate
DX	Word Multiply, Word Divide, Indirect I/O
SP	Stack Operations
SI	String Operations
DI	String Operations

Figure 2-3. Implicit Use of General Registers

Intel x86: i8088

Memory

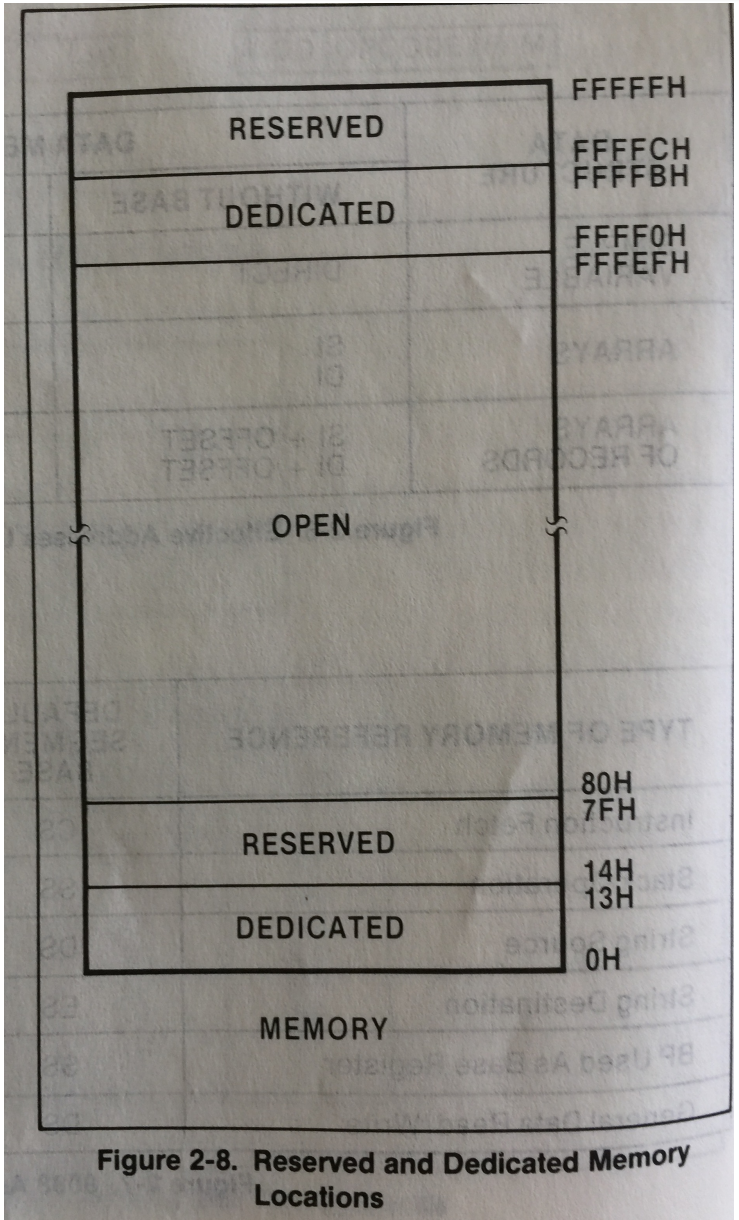


Figure 2-8. Reserved and Dedicated Memory Locations

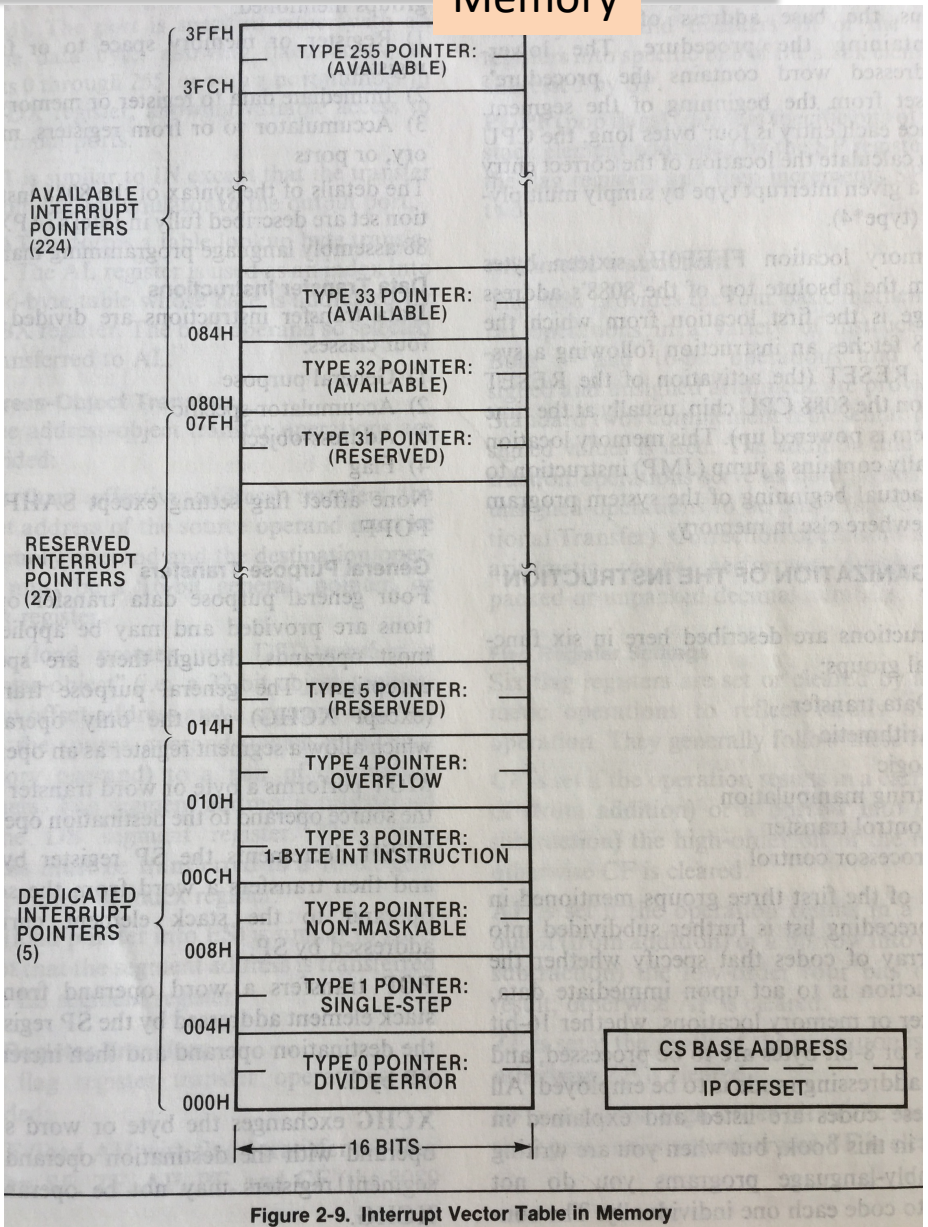


Figure 2-9. Interrupt Vector Table in Memory

Intel x86: i8088

Addressing

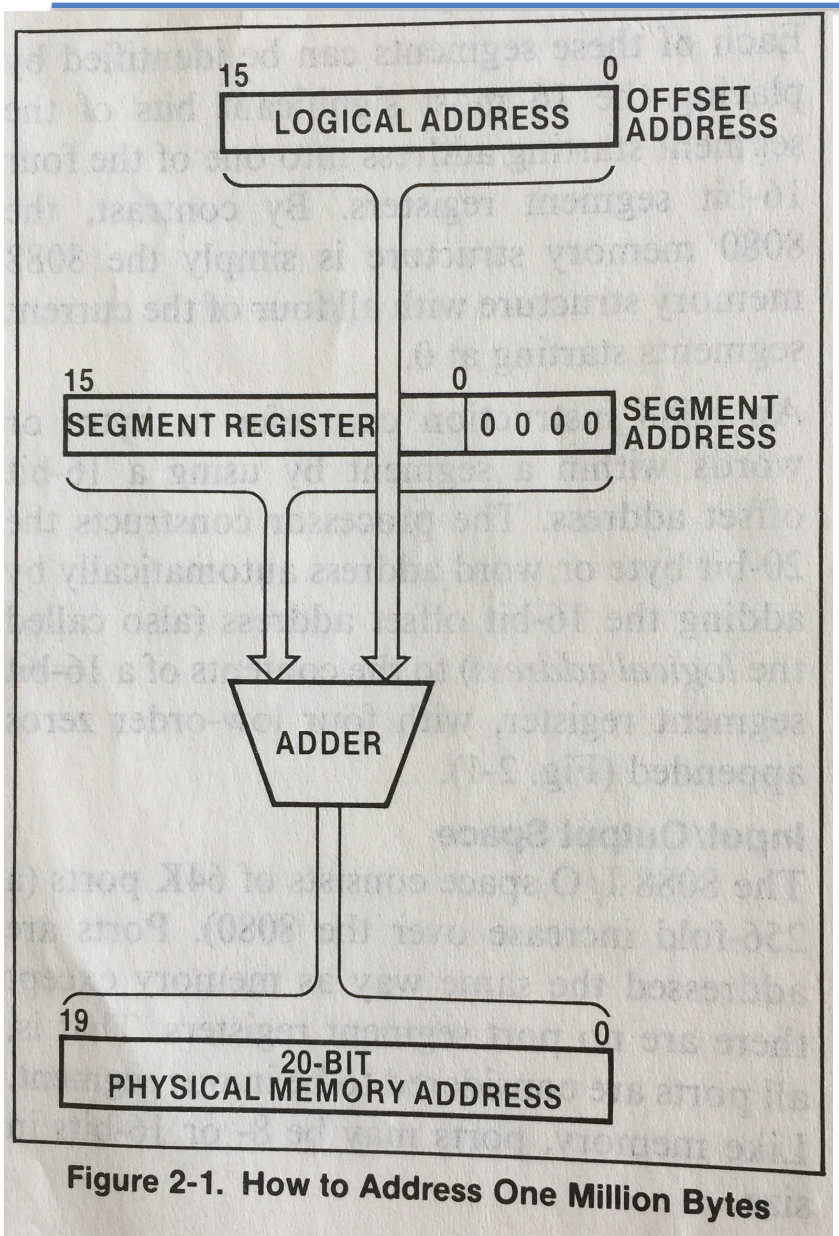


Figure 2-1. How to Address One Million Bytes

EA COMPONENTS		CLOCKS*
Displacement Only		6
Base or Index Only	(BX,BP,SI,DI)	5
Displacement + Base or Index	(BX,BP,SI,DI)	9
Base + Index	BP + DI, BX + SI	7
	BP + SI, BX + DI	8
Displacement + Base + Index	BP + DI + DISP BX + SI + DISP	11
	BP + SI + DISP BX + DI + DISP	12

*Add 2 clocks for segment override

Figure 2-10. Effective Address Calculation Time

Intel x86: i8088

Addressing

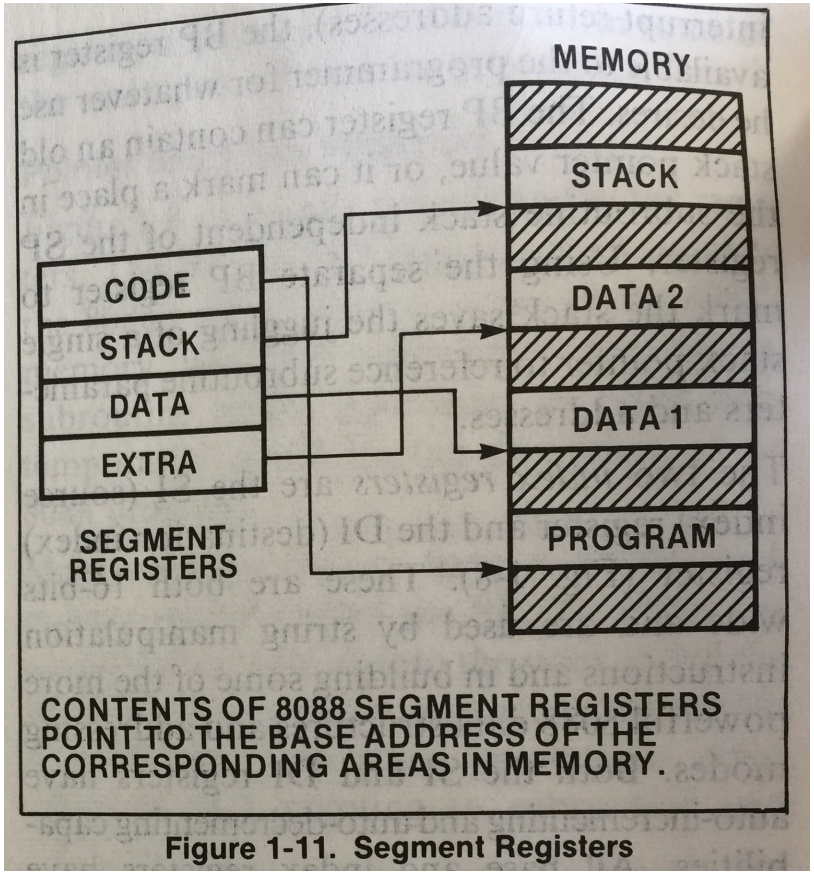


Figure 1-11. Segment Registers

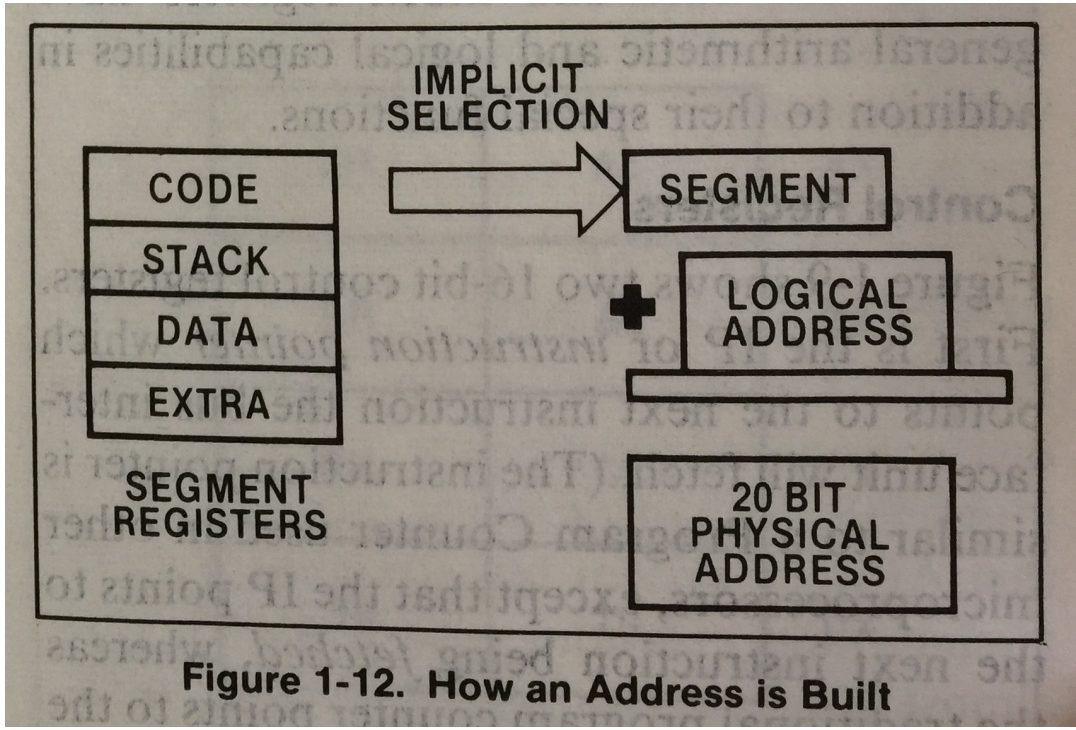


Figure 1-12. How an Address is Built

Intel x86: i8088

Addressing

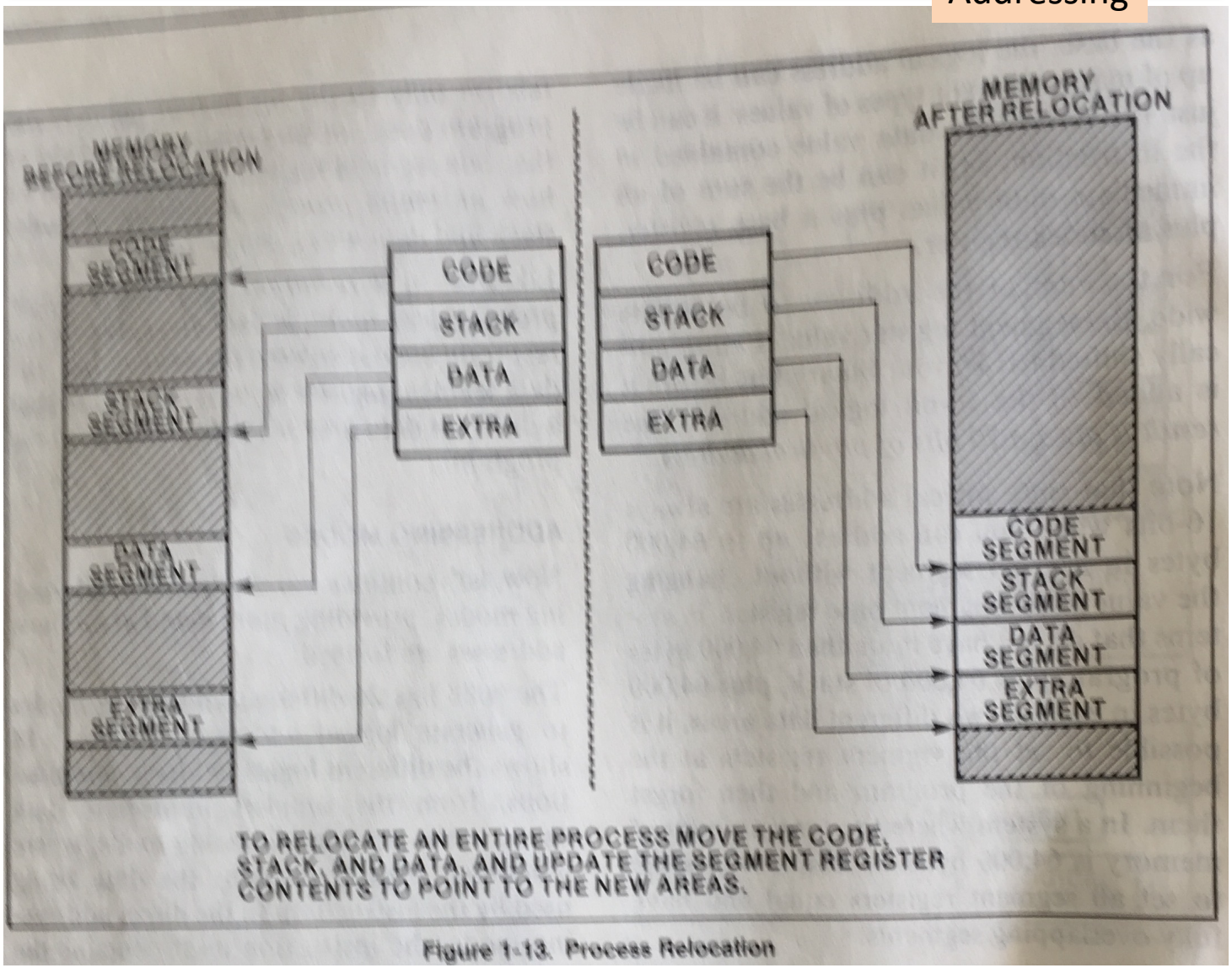


Figure 1-13. Process Relocation

Intel x86: i8088

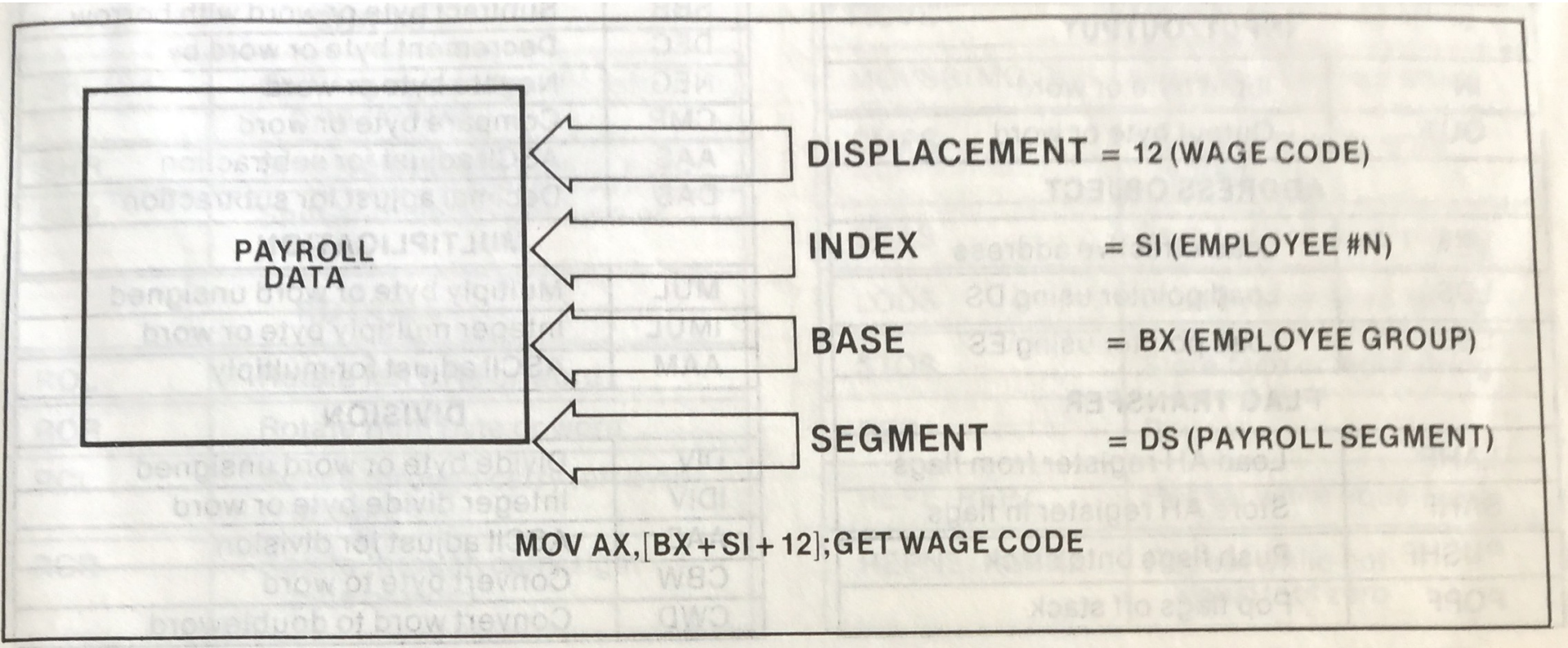


Figure 1-15. Four-Component Addressing Example

Intel x86: i8088

Figure 1-13. Process Rel...

MODE	LOCATION OF DATA
IMMEDIATE	WITHIN INSTRUCTION
REGISTER	IN REGISTER
DIRECT	AT MEMORY LOCATION POINTED TO BY ADDRESS CONTAINED IN INSTRUCTION.
REGISTER INDIRECT	AT MEMORY LOCATION POINTED TO BY ADDRESS CONTAINED IN REGISTER.
INDEXED OR BASED	AT MEMORY LOCATION POINTED TO BY SUM OF INDEX REGISTER OR BASE REGISTER CONTENTS AND IMMEDIATE DATA CONTAINED IN INSTRUCTION.
BASED AND INDEXED WITH DISPLACEMENT	MEMORY ADDRESS IS SUM OF BASE REGISTER CONTENTS AND INDEX REGISTER CONTENTS AND IMMEDIATE DATA.

THE LOCATION OF DATA IS REALLY THE LOGICAL ADDRESS, WHICH IS ADDED TO THE SEGMENT REGISTER VALUE TO FORM THE PHYSICAL MEMORY ADDRESS.

Figure 1-14. iAPX 88 Addressing Modes

Intel x86: i8088

DUAL FUNCTION KEYWORD/SYMBOLS

AND	NOT	OR	SHL	SHR	XOR
AAA	ES	FLD1	FSUBRP	JNGE	PUSH
AAD	ESC	FLDCW	FTST	JNL	PUSH
AAM	F2XM1	FLDENV	FWAIT	JNLE	F
AAS	FABS	FLDL2E	FXAM	JNO	RCL
ADC	FAC	FLDL2T	FXCH	JNP	RCR
ADD	FADD	FLDLN2	FXTRACT	JNS	REP
AH	FADDP	FLDLG2	FYL2X	JNZ	REPE
AL	FALC	FLDPI	FYL2XPI	JO	REPN
ARPL	FBLD	FLDZ	HLT	JP	E
AX	FBSTP	FMUL	IDIV	JPE	REPNZ
BH	FCHS	FMULP	IMUL	JPO	REPZ
BL	FCLEX	FNCLEX	IN	JS	RET
BOUND	FCOM	FNDISI	INC	JZ	ROL
BP	FCOMP	FNENI	INT	LAHF	ROR
BX	FCOMPP	FNINIT	INTO	LDS	SAHF
CALL	FDECSTP	FNOP	IRET	LEA	SAL
CBW	FDISI	FNSAVE	JA	LES	SAR
CH	FDIV	FNSTCW	JAE	LOCK	SBB
CL	FDIVP	FNSTENV	JB	LODS	SCAS
CLC	FDIVR	FNSTSW	JBCZ	LODSB	SCAS
CLD	FDIVRP	FPATAN	JBE	LODSW	B
CLI	FENI	FPREM	JC	LOOP	SCAS
CLTS	FFREE	FPTAN	JCXE	LOOPE	W
CMC	FIADD	FRNDINT	JE	LOOPNE	SI
CMP	FICOM	FRSTOR	JG	LOOPNZ	SP
CMPS	FICOMP	FSAVE	JGE	LOOPZ	SS
CMPSB	FIDIV	FSCALE	JL	MOV	ST
CMPSW	FIDIVR	FSQRT	JLE	MOVS	STC
CS	FILD	FST	JMP	MOVSB	STD
CWD	FMUL	FSTCW	JNA	MOVSW	STI
CX	FINCSTP	FSTENV	JNAE	MUL	STOS
DAA	FINIT	FSTP	JNB	NEG	STOSB
DAS	FIST	FSTSW	JNBE	NIL	STOS
DEC	FISTP	FSUB	JNC	OUT	W
DH	FISUB	FSUBP	JNE	POP	SUB
DI	FISUBR	FSUBR	JNG	POPF	TEST
DIV	FLD				WAIT
DL					XCHG
DS					XLAT
DX					XLATB
					??SEG

Figure 2-14. ASM-86 Reserved Words

Intel x86: i8088

ALU

ADDITION	
ADD	Add byte or word
ADC	Add byte or word with carry
INC	Increment byte or word by 1
AAA	ASCII adjust for addition
DAA	Decimal adjust for addition
SUBTRACTION	
SUB	Subtract byte or word
SBB	Subtract byte or word with borrow
DEC	Decrement byte or word by 1
NEG	Negate byte or word
CMP	Compare byte or word
AAS	ASCII adjust for subtraction
DAS	Decimal adjust for subtraction
MULTIPLICATION	
MUL	Multiply byte or word unsigned
IMUL	Integer multiply byte or word
AAM	ASCII adjust for multiply
DIVISION	
DIV	Divide byte or word unsigned
IDIV	Integer divide byte or word
AAD	ASCII adjust for division
CBW	Convert byte to word
CWD	Convert word to doubleword

Figure 1-17. Arithmetic Instructions

Instructions

GENERAL PURPOSE	
MOV	Move byte or word
PUSH	Push word onto stack
POP	Pop word off stack
XCHG	Exchange byte or word
XLAT	Translate byte
INPUT/OUTPUT	
IN	Input byte or word
OUT	Output byte or word
ADDRESS OBJECT	
LEA	Load effective address
LDS	Load pointer using DS
LES	Load pointer using ES
FLAG TRANSFER	
LAHF	Load AH register from flags
SAHF	Store AH register in flags
PUSHF	Push flags onto stack
POPF	Pop flags off stack

Figure 1-16. Data Transfer Instructions

Intel x86: i8088

Multiplication

Three multiplication operations are provided:

MUL performs an unsigned multiplication of the accumulator (AL or AX) and the source operand, returning a double length result to the accumulator and its extension (AL and AH for 8-bit operation, AX and DX for 16-bit operation). CF and OF are set if the top half of the result is non-zero.

IMUL (integer multiply) is similar to MUL except that it performs a signed multiplication. CF and OF are set if the top half of the result is not the sign-extension of the low half of the result.

AAM (unpacked BCD [ASCII] adjust for multiply) performs a correction of the result in AX of multiplying two unpacked decimal operands, yielding an unpacked decimal product.

Division

Three division operations are provided and two sign-extension operations to support signed division:

DIV performs an unsigned division of the accumulator and its extension (AL and AH for 8-bit operation, AX and DX for 16-bit operation) by the source operand and returns the single length quotient to the accumulator (AL or AX), and returns the single length remainder to the accumulator extension (AH

Intel x86: i8088

Instructions

ALU

Shift

LOGICALS	
NOT	"Not" byte or word
AND	"And" byte or word
OR	"Inclusive or" byte or word
XOR	"Exclusive or" byte or word
TEST	"Test" byte or word
SHIFTS	
SHL/SAL	Shift logical/arithmetic left byte or word
SHR	Shift logical right byte or word
SAR	Shift arithmetic right byte or word
ROTATES	
ROL	Rotate left byte or word
ROR	Rotate right byte or word
RCL	Rotate through carry left byte or word
RCR	Rotate through carry right byte or word

Figure 1-18. Bit Manipulation Instructions

String

MOVS	Move byte or word string
MOVSB/MOVSW	Move byte or word string
CMPS	Compare byte or word string
SCAS	Scan byte or word string
LODS	Load byte or word string
STOS	Store byte or word string
REP	Repeat
REPE/REPZ	Repeat while equal/zero
REPNE/REPNZ	Repeat while not equal/not zero

Figure 1-19. String Instructions

Intel x86: i8088

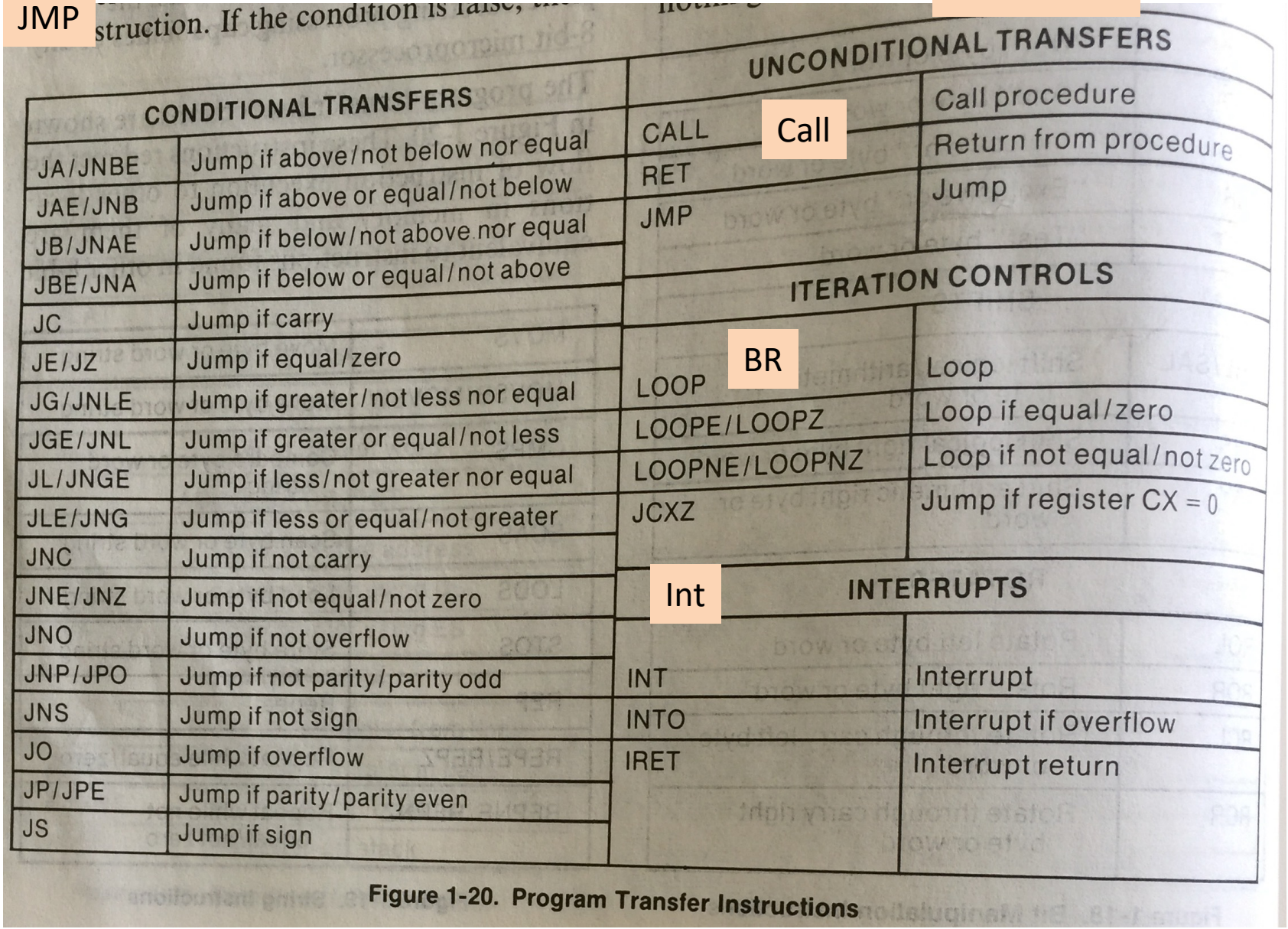
Instructions

JMP

Call

BR

Int



struction. If the condition is false,

CONDITIONAL TRANSFERS	
JA/JNBE	Jump if above/not below nor equal
JAE/JNB	Jump if above or equal/not below
JB/JNAE	Jump if below/not above nor equal
JBE/JNA	Jump if below or equal/not above
JC	Jump if carry
JE/JZ	Jump if equal/zero
JG/JNLE	Jump if greater/not less nor equal
JGE/JNL	Jump if greater or equal/not less
JL/JNGE	Jump if less/not greater nor equal
JLE/JNG	Jump if less or equal/not greater
JNC	Jump if not carry
JNE/JNZ	Jump if not equal/not zero
JNO	Jump if not overflow
JNP/JPO	Jump if not parity/parity odd
JNS	Jump if not sign
JO	Jump if overflow
JP/JPE	Jump if parity/parity even
JS	Jump if sign

UNCONDITIONAL TRANSFERS	
CALL	Call procedure
RET	Return from procedure
JMP	Jump

ITERATION CONTROLS	
LOOP	Loop
LOOPE/LOOPZ	Loop if equal/zero
LOOPNE/LOOPNZ	Loop if not equal/not zero
JCXZ	Jump if register CX = 0

INTERRUPTS	
INT	Interrupt
INTO	Interrupt if overflow
IRET	Interrupt return

Figure 1-20. Program Transfer Instructions

Intel x86: i8088

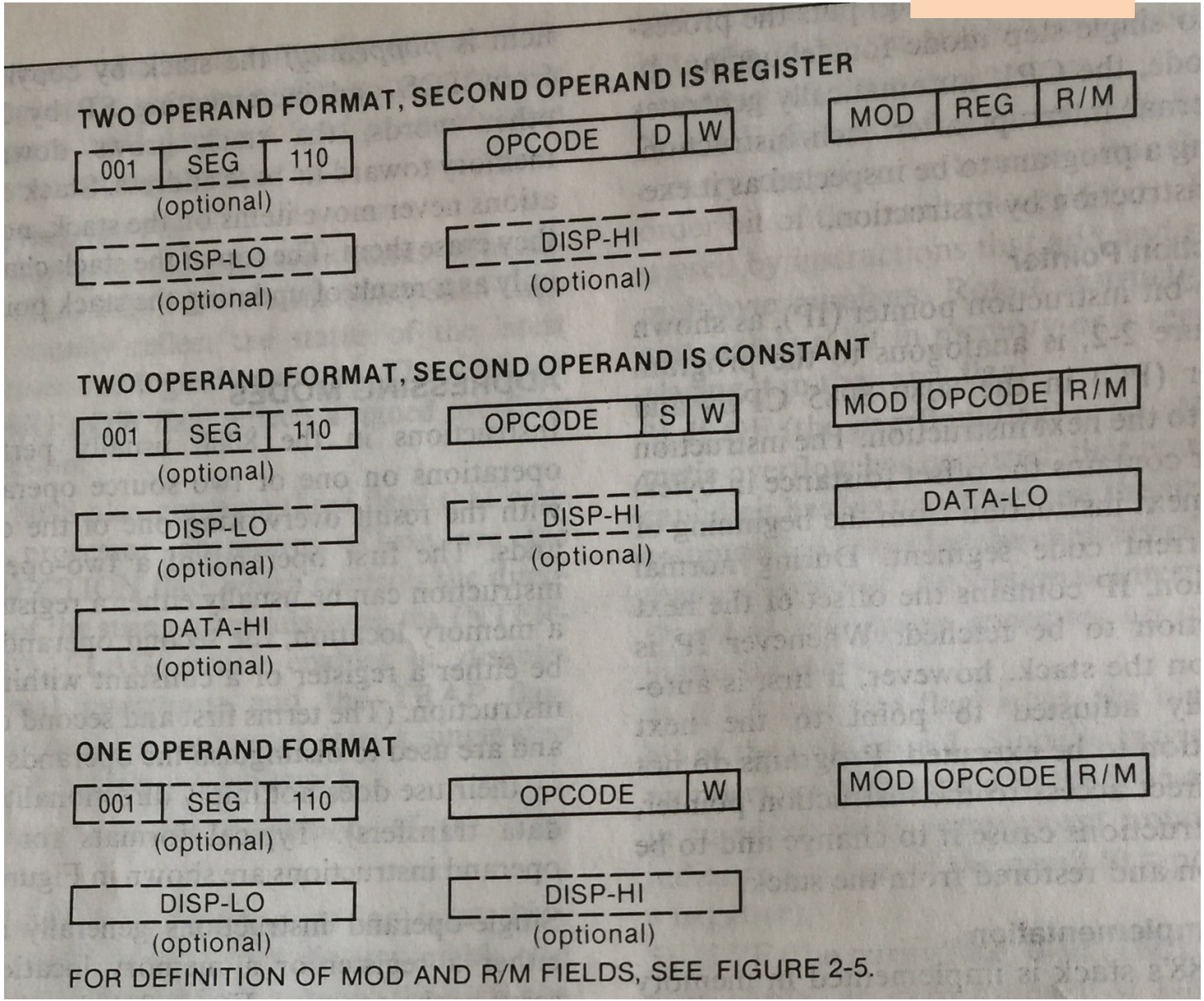
Instructions

FLAG OPERATIONS	
STC	Set carry flag
CLC	Clear carry flag
CMC	Complement carry flag
STD	Set direction flag
CLD	Clear direction flag
STI	Set interrupt enable flag
CLI	Clear interrupt enable flag
EXTERNAL SYNCHRONIZATION	
HLT	Halt until interrupt or reset
WAIT	Wait for $\overline{\text{TEST}}$ pin active
ESC	Escape to external processor
LOCK	Lock bus during next instruction
NO OPERATION	
NOP	No operation

NOP

Figure 1-21. Processor Control Instructions

Intel x86: i8088



Intel x86: i8088

FIRST OPERAND CHOICE DEPENDS ON ADDRESSING MODE:

FIRST OPERAND IN MEMORY			FIRST OPERAND IN REGISTER		
INDIRECT ADDRESSING		DIRECT ADDRESSING	MOD = 11		
MOD = 00* : DISP = 0 MOD = 01 : DISP = DISP-LO SIGN EXTENDED 10 : DISP = DISP-HI, DISP-LO		MOD = 00 AND R/M = 110 OPERAND EFFECTIVE ADDRESS = DISP-HI, DISP-LO	R/M:	REGISTER	
R/M:	OPERAND EFFECTIVE ADDRESS			8-BIT (W = 0)	16-BIT (W = 1)
000	(BX) + (SI) + DISP		000	AL	AX
001	(BX) + (DI) + DISP		001	CL	CX
010	(BP) + (SI) + DISP		010	DL	DX
011	(BP) + (DI) + DISP		011	BL	BX
100	(SI) + DISP		100	AH	SP
101	(DI) + DISP		101	CH	BP
110	(BP) + DISP		110	DH	SI
111	(BX) + DISP		111	BH	DI

Where ( ) means "contents of"
*Exception—direct addressing mode

Figure 2-5. Determining First Operand

DATA STRUCTURE	DATA MEMORY		STACK
	WITHOUT BASE	WITH BASE	
SIMPLE VARIABLE	DIRECT	BX + OFFSET	BP + OFFSET
ARRAYS	SI DI	BX + SI BX + DI	BP + SI BP + DI
ARRAYS OF RECORDS	SI + OFFSET DI + OFFSET	BX + SI + OFFSET BX + DI + OFFSET	BP + SI + OFFSET BP + DI + OFFSET

Figure 2-6. Effective Addresses Used with Different Data Structures

TYPE OF MEMORY REFERENCE	DEFAULT SEGMENT BASE	ALTERNATE SEGMENT BASE	LOGICAL ADDRESS
Instruction Fetch	CS	NONE	IP
Stack Operation	SS	NONE	SP
String Source	DS	CS,ES,SS	SI
String Destination	ES	NONE	DI
BP Used As Base Register	SS	CS,DS,ES	Effective Address
General Data Read/Write	DS	CS,ES,SS	Effective Address

Figure 2-7. 8088 Address Components

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP122

Intel x86: i8088

Assembly Code

labels | Ops | operands | comments

```
1.  IN_AND_OUT  SEGMENT                 ;start of segment
2.              ASSUME  CS: IN_AND_OUT  ;that's what's in CS
3.  CYCLE:      IN      AX,5
4.              INC     AX
5.              OUT     2,AX
6.              JMP     CYCLE
7.  IN_AND_OUT  ENDS                    ;end of segment
8.              END     CYCLE           ;end of assembly
```

Macro Constants (EQU)

```
BOILING_POINT  EQU  212
BUFFER_SIZE    EQU  32
NEW_PORT       EQU  PORT_VAL+1
COUNT          EQU  CX
```

Declare variables

```
THING          DB  ?   ;defines a byte
BIGGER_THING   DW  ?   ;defines a word (2 bytes)
BIGGEST_THING  DD  ?   ;defines a doubleword (4 bytes)
```

Intel x86: i8088

Assembly Code

```
1.  MY_DATA     SEGMENT                          ;data segment
2.  SUM         DB          ?                    ;reserve a byte for SUM
3.  MY_DATA     ENDS
4.  MY_CODE     SEGMENT                          ;code segment
5.              ASSUME      CS:MY_CODE, DS:MY_DATA
                                                 ;contents of CS and DS
6.  PORT_VAL    EQU         3                    ;symbolic name for port number
7.  GO:         MOV         AX,MY_DATA           ;initialize DS to MY_DATA
8.              MOV         DS,AX
9.              MOV         SUM,0                ;clear sum
10. CYCLE:      CMP         SUM,100              ;if SUM exceeds 100
11.             JNA         NOT_DONE
12.             MOV         AL,SUM               ;...then output SUM to port 3
13.             OUT         PORT_VAL,AL
14.             HLT                              ;...and stop execution
15. NOT_DONE:   IN          AL,PORT_VAL          ;otherwise add next input
16.             ADD         SUM,AL
17.             JMP         CYCLE                ;and repeat the test
18. MY_CODE     ENDS
19.             END         GO                   ;this is the end of the assembly
```

Intel x86: i8088

Assembly Code

Stack example

```
MY_DATA     SEGMENT
X           DB      ?
Y           DW      ?
Z           DD      ?
MY_DATA     ENDS
MY_EXTRA    SEGMENT
ALPHA       DB      ?
BETA        DW      ?
GAMMA       DD      ?
MY_EXTRA    ENDS
MY_STACK    SEGMENT
            DW      100 DUP (?)          ;this is the stack
TOP         EQU     THIS WORD
MY_STACK    ENDS
MY_CODE     SEGMENT
            ASSUME  CS:MY_CODE,DX:MY_DATA
            ASSUME  ES:MY_EXTRA,SS:MY_STACK
START:      MOV     AX,MY_DATA           ;initializes DX
            MOV     DS,AX
            MOV     AX,MY_EXTRA          ;initializes ES
            MOV     ES,AX
            MOV     AX,MY_STACK          ;initializes SS
            MOV     SS,AX
            MOV     SP,OFFSET TOP        ;initializes SP
            .
            .
MY_CODE     ENDS
            END     START
```

Intel x86: i8088

Assembly Code

Details of ASM-86

Sample One:

Translate the values from input port 1 into a Gray code and send result to output port 1.

I/O + *Gray code* example

```
MY_DATA   SEGMENT
GRAY      DB        18H,34H,05H,06H,09H,0AH,0CH,11H,12H,14H
MY_DATA   ENDS

MY_CODE   SEGMENT
          ASSUME    CS:MY_CODE, DS:MY_DATA
GO:       MOV       AX,MY_DATA          ;establish data segment
          MOV       DS,AX
          MOV       BX,OFFSET GRAY      ;translation table into BX
CYCLE:    IN        AL,1                ;read in next value
          XLAT      GRAY                ;translate it
          OUT       1,AL                ;output it
          JMP       CYCLE               ;and repeat
MY_CODE   ENDS
          END       GO
```

Intel x86: i8088

Assembly Code

Sample Three:

Decimal multiplication algorithm.

Decimal Mult example

```
MY_DATA   SEGMENT
A         DB      '3','7','5','4','9'
B         DB      '6'
C         DB      LENGTH (A) DUP (?)
MY_DATA   ENDS

MY_CODE   SEGMENT
          ASSUME  CS:MY_CODE,DS:MY_DATA
GO:       MOV     AX,MY_DATA          ;establish data segment
MOV       DS,AX
          CLD                         ;forward strings
          MOV     SI,OFFSET A         ;establish pointers
          MOV     DI,OFFSET C
          MOV     CX,LENGTH A         ;establish count
          AND     B,0FH               ;clear upper half of b
          MOV     BYTE PTR [SI],0     ;clear c[I]
CYCLE:    LODS    A                   ;get a[i]
          AND     AL,0FH              ;clear its high-order bits
          MUL     AL,B                ;multiply by b
          AAM                         ;correct for ASCII
          ADD     [DI]                ;add to c[i]
          AAA                         ;adjust for ASCII
          STOS    C                   ;store in c[i]
          MOV     [DI],AH             ;...and c[I]
          JCXZ    CYCLE               ;repeat for entire string
          HLT
MY_CODE   ENDS
          END     GO
```