

Table of Contents for The Zen of Assembler

Introduction

Part I: The Zen of Assembler

Chapter 1: Zen?

Chapter 2: Assume Nothing

The Zen timer

Starting the Zen timer

Time and the PC

Stopping the Zen timer

Reporting timing results

Notes on the Zen timer

A sample use of the Zen timer

The long-period Zen timer

A sample use of the long-period Zen timer

Further reading

Armed with the Zen timer, onward and upward

Part II: Knowledge

Chapter 3: Context

- The traditional model
- Cycle-eaters
- Code is data
 - Inside the 8088
- Stepchild of the 8086
- Which model to use?

Chapter 4: Things Mother Never Told You: Under the Programming Interface

- Cycle eaters redux
 - The 8-bit bus cycle-eater
 - The impact of the 8-bit bus cycle-eater
 - What to do about the 8-bit bus cycle-eater?
 - The prefetch queue cycle-eater
 - Official execution times are only part of the story
 - There is no such beast as a true instruction execution time
 - Approximating overall execution times
 - What to do about the prefetch queue cycle-eater?
 - Holding up the 8088
- Dynamic RAM refresh: The invisible hand
 - How DRAM refresh works in the PC
 - The impact of DRAM refresh
 - What to do about the DRAM refresh cycle-eater?
 - Wait states
- The display adapter cycle-eater
 - The impact of the display adapter cycle-eater
 - What to do about the display adapter cycle-eater?
 - What does it all mean?

Chapter 5: Night of the Cycle-Eaters

- No, we're not in Kansas anymore
- Cycle-eaters by the battalion
- ...there's still no such beast as a true execution time
- 170 cycles in the life of a PC
- The test set-up
- The results
- Code execution isn't all that exciting

Abrash/Zen: Table of Contents/

The 8088 really does coprocess
When does an instruction execute?
The true nature of instruction execution
Variability
You never know unless you measure (in context!)
The longer the better
Odds and ends
Back to the programming interface

Chapter 6: The 8088

An overview of the 8088

Resources of the 8088

Registers

The 8088's register set

The general-purpose registers

The AX register

The BX register

The CX register

The DX register

The SI register

The DI register

The BP register

The SP register

The segment registers

The CS register

The DS register

The ES register

The SS register

Other registers

The Instruction Pointer

The FLAGS register

The Carry (C) flag

The Parity (P) flag

The Auxiliary Carry (A) flag

The Zero (Z) flag

The Sign (S) flag

The Overflow (O) flag

The Interrupt (I) flag

The Direction (D) flag

The Trap (T) flag

There's more to life than just registers

Chapter 7: Memory Addressing

Efficient stack frames...the odd architecture of 8088 memory access...use nears whenever possible, use <=64K fars if not...organize programs and data so you can set up the segments for long periods at a time, to reduce a far task to a series of near operations...leave ES loaded if possible...loading segments (push/pop vs. mov/mov vs. mov reg,mem...immediate addressing incurs overhead...<<<MORE>>>

Chapter 8: Strange Fruit of the 8080

- The 8080 legacy
 - More than a passing resemblance
- Accumulator-specific instructions
- Accumulator-specific direct-addressing instructions
 - Looks aren't everything
 - How fast are they?
 - When should you use them?
- Accumulator-specific immediate-operand instructions
- An accumulator-specific example
- Other accumulator-specific instructions
- The accumulator-specific version of **test**
- The AX-specific version of **xchg**
- Pushing and popping the 8080 flags
 - lahf** and **sahf**: An example
 - A brief digression on optimization
- Onward to the instruction set

Chapter 9: Around and About the Instruction Set

- Shortcuts for handling zero and constants
- Making zero
- Initializing constants from the registers
- Initializing two bytes with a single **mov**
- More fun with zero
 - inc** and **dec**
 - Using 16-bit **incs** and **decs** for 8-bit operations
 - How **inc** and **add** (and **dec** and **sub**) differ--and why
- Carrying results along in a flag
 - Byte-to word and word-to-doubleword conversion
 - xchg** is handy when registers are tight
 - Destination: Register
 - neg** and **not**
 - Rotates and shifts
- Rotates
- Shifts
- Signed division with **sar**
- ASCII and decimal adjust
 - daa**, **das**, and packed BCD arithmetic
 - aam**, **aad**, and unpacked BCD arithmetic
 - aas**, **aas**, and decimal ASCII arithmetic

Mnemonics that cover multiple instructions
On to the string instructions

Chapter 10: String Instructions: The Magic Elixir

Inherently faster and smaller...repeated string instructions have prefetch benefits as well...as with LOOP, don't assume string instructions are always faster (SCASB versus CMP/JZ)...use word whenever possible...REP doesn't work on 64K items---how to handle...prefixes...don't use multiple prefixes...REP in its various forms...initializing blocks

Chapter 11: Branching

Jumps are slow...know all the jump conditions (JS)...JCXZ/LOOP/LOOPNZ/LOOPZ (no flag effects)...jumping from memory...jumping from a register...constructing a jump as a return on the stack to preserve/save registers...jump tables...INT...jmp/jmp vs call/ret...pop/jmp reg for ret...faking IRET w/flags..faking INT w/far call...On to the 80286 and 80386...fake far call by pushing CS and doing a near call

Chapter 12: 80286/80386 Considerations

Both chips were designed to pretty much eliminate the prefetch queue bottleneck---with zero-wait-state memory, so long as you don't branch...memory and I/O wait states in stock ATs...wait states & memory architecture in 80386 machines...the prefetch queue...8-bit bus emulation...branching...word and doubleword alignment (tale of developing Zen timer)...registers still pay off...much- reduced effective address calculation time...8/16-bit memory wait states (including display adapters!)...buses are slowed down for standard peripherals...can't really plan as well for these, though, except not to expect your code to run as fast as it should...refresh

Chapter 13: System Resources

Interrupts...Timers...DMA controller...BIOS (write dot)...DOS...let them all do for you what they do well...beware of redirection

Chapter 14: Understanding What MASM Can Do

The de facto standard for the 8088 world...this is not a MASM book, but there are some aspects of MASM that are part of the Zen of assembler, and you should be very familiar with it...MASM is a strange assembler--learn to live with it...don't calculate anything at run time that you can calculate at assembly time (tables)...conditional block for debugging and development

Chapter 15: Macros: The Good, the Bad, and the Occasionally Ugly

Let them do the work for you wherever possible (backward jumps, psuedo-instructions)...sometimes of dubious reliability...building a table of addresses...using macros to build tables...macros slow up assembly...it can be costly to rely heavily on high-level macros or subroutines, since by being reusable they can be inefficient: modifying working code often works better...mention TASM

Part III: The Flexible Mind

Chapter 16: Knowledge Matters Only when It's Used

The programmer's integration of knowledge and application is the key to good software...two levels: 1) making the most effective code locally (local optimization); 2) matching that to the application (global optimization)...no sharp line between the two...key is always to know what the PC can do, then match that to the task as efficiently as possible, even when that means using unorthodox techniques...we'll look at an example, then review a number of general and specific principles for "zenning" code (define "zenning")...zen in big ways (program structure, algorithms) & little ways (clever test & jump)

Chapter 17: Executing Zen: A Case History

"Zenning" the simple example from [Turbo Pascal Solutions](#)

Chapter 18: Limit Scope as Needed to Match Available Resources

Use buffers $\leq 64K$ in size, to allow speedy searching and manipulation, paging in data with restartable string instructions if necessary to support this...reduce resolution or color selection if there's not enough memory otherwise...in short, look for ways to pare the program back to the essentials if that's what it takes to run well on a PC...example of redirected file filtering versus internally buffered filtering versus block string filtering

Chapter 19: Be Willing to Break Your Own Rules When Necessary

Don't always preserve all registers...don't stick to parameter-passing conventions when it's not worth it

Chapter 20: Think Laterally: Use Your Right Brain

Pick the right algorithm, but match it to the potential of the PC...avoid compileritis like the plague (compilers can out-compile you, but they can't out-lateral you; you know more and can assume more; don't write assembler code built around compiler conventions like stack frames)...example of A XOR B XOR B to transfer values when an intermediate register wasn't available...don't trap yourself in a limited environment (C programmer who cleared the screen a character at a time; using longs, fars, or hugs unnecessarily) (also, don't build in permanent safeguards against yourself--- modularity and security are nice, but speed is better--- during development, insert safeguards so that they can later be pulled by setting a single flag)...follow the trail wherever it leads (my path to understanding the display adapter bottleneck)...understand all the code you use (tale of Joel and his EGA ID code from a book)...know when it's worth the effort (inside loops, but not necessarily when setting up for loops) (searching examples)...know when to be elegant (searching/sorting examples)...each solution is a unique work of art...example of animation during vertical non-display: I was so sure no more objects could be animated, and then John pointed out that page flipping allowed any number---a different perspective on system resources...example of non-blue underlined text on the EGA and VGA...don't be afraid to dive in and apply Zen to already-working code---in important code, just working is not enough

Chapter 21: Live in the Registers

Registers avoid effective address calculation...fewer instruction bytes...in a way, prefetch queue bottleneck is worse (overdemands on BIU), but fewer bytes per function...register-specific instructions (INC word, XCHG with AX)...using registers to hold constants...use all the registers (Dan's use of SP with interrupts on---but it would have been all right with interrupts off)...remember that BP can address off any segment, and if SS and DS are the same (as in COM files), BP is by-and-large as useful as BX for memory addressing...using half-registers...PUSH/RET to vector if registers are in short supply...memory variables should be in [] brackets--they are not like having lots of registers!...funnel multiple cases to clean-up code, with values in registers, so there's only one set of memory-accessing instructions...avoid immediate operands (keep cmp & add, etc.,

Abrash/Zen: Table of Contents/

values in registers if possible--extension of zero/constant handling in chapter 9)

Chapter 22: Don't jump!

Strange title, when decision-making is key, but 8088 is slow at branching, so minimize it (decision-making and repeating differ)...what the prefetch queue means when branching...ADC DI,0 versus JNC/INC DI...preload default value & jump only one way...lead into in-line code 2 chapters away through next chapter

Chapter 23: Memory Is Cheap (You Could Look It Up)

Throwing memory at problems can compensate for limited processor power...tables are a good way of precalculating results...jump tables...put them in CS if you're not sure what DS will be---the cost is small...multiplying by 80...bit doubling

Chapter 24: (Sometimes) LOOP Is a Crock: In-line Alternatives

Just because there's an instruction for looping doesn't mean it's particularly fast...in-line code can do the same thing without the branching penalty...mix the two for a large fraction of both the speed benefits of in-line code and the size benefits of LOOP...looping high to low instead of low to high...more about in-line code in general

Chapter 25: Flexible Data & Mini-Interpreters

Assembler is by far the best language for flexible data specification...mini-interpreters are compact and reliable, and can be driven by flexible data strings containing addresses of tables and routines, as well as data of any type...mini-interpreters allow use of programming models unique to assembler (could even embed control strings in CS and returning to the instruction immediately following the string---courtesy of a DDJ article)...don't be afraid to put data in CS, which can help with staying in the near model

Chapter 26: Display Adapter Programming

CGA, MDA, Hercules, EGA, and VGA programming considerations...using string instructions & related approaches, to minimize memory accesses...prerotate images...predefine control strings...byte align, don't mask/clip within a byte...don't xor/and/or if possible, since full wait on second, but preferable to two accesses...single instructions to read/modify/write

Chapter 27: Odds and Ends

Returning results and statuses...self-modifying code...move work

Abrash/Zen: Table of Contents/

outside loops...parameter passing...be clever with high/low bit testing
(rotate, shift, sign test)...boolean logic & binary arithmetic...and
bx,xxx to both convert to word and mask off

Part IV: Animation: The Zen of Assembler in Action

Chapter 28: Animation Fundamentals

How animation is generated...a personal journey through animation driver code and techniques...what various approaches do best

Chapter 29: A Discourse on VGA Graphics

Basic adapter architecture and resources

Chapter 30: Evolution of an Animation Application

The germ of the program...growing the program concept in the framework of the VGA

Chapter 31: Key Pieces of the Animation Program

Animation drivers...panning

Chapter 32: An Overview of the Animation Program Code

A quick scan through the code, looking at overall logic

Appendixes

Appendix A: The 8088 Instruction Set

Includes sizes & timings...286/386 instructions & timings would be useful as well

Appendix B: Listing of the Animation Program