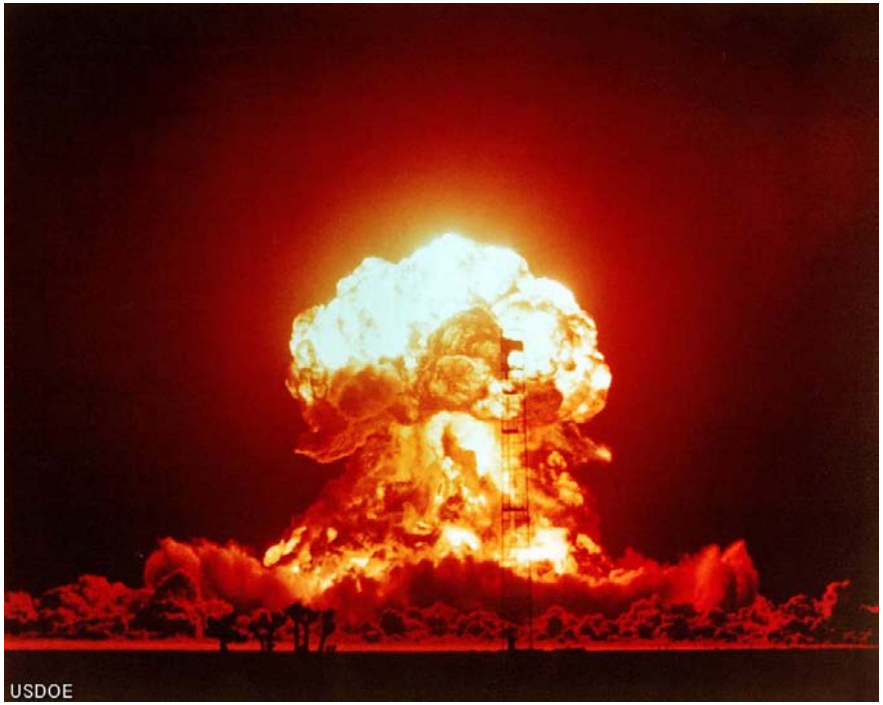
A Baker's Dozen: Fallacies and Pitfalls in Processor Design

Contents crafted by
Chief Scientist Grant Martin &
Technology Evangelist Steve Leibson
Tensilica, Inc.

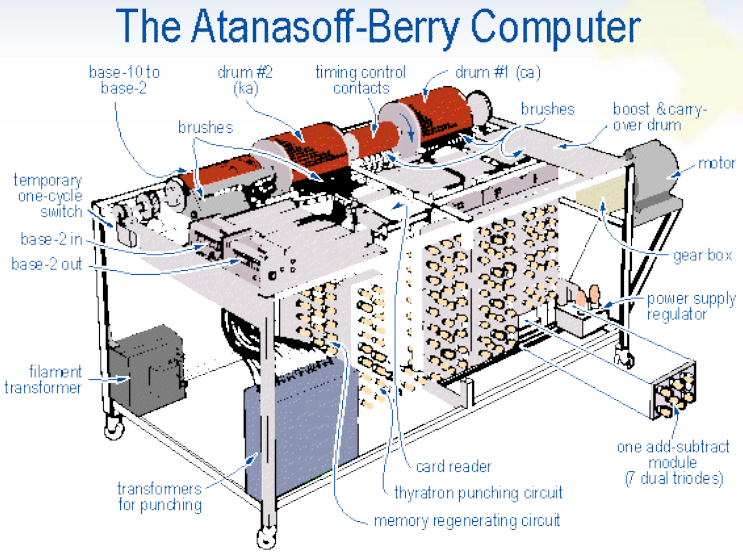
Why Listen to This Presentation?

- Learn about the seven decades of design mistakes made by computer and processor designers
- So that you won't make them too
- Except that you will



A Brief History of Processors

- ~70 years of electronic computer development starting in the 1930s with Alan Turing, the Conrad Zuse Z1 in Germany, and the Antanasoff-Berry Computer at Iowa State College
- Many processor species have appeared over this time and many have disappeared
- Some reappear from time to time as technology changes



Conrad Zuse

Definition: ISA

- **An ISA (Instruction Set Architecture) includes:**
  - The processor's state (registers and register files)
  - The processor's instruction set
  - The processor's interfaces
    - Main bus
    - Local memory buses
    - Other I/O ports

What Have We Learned in 70 Years?

Pitfall 1: ISAs that Directly Support High-Level Languages

- **The Idea**
  - Optimize the processor's ISA so to improve execution efficiency for one or more selected High-Level programming Languages (HLLs)
- **Many Notable Attempts**
  - Burroughs E-mode machines (B5000, B6000, B7000 for Algol 60)
  - Burroughs B2000, B3000, B4000 (Cobol)
  - LISP Machines, Symbolics (LISP)
  - Inmos Transputer (Occam)
  - Intel 432 (Ada)
  - Java machines (Sun picoJava, ARM Jazelle, etc.)

The Intel 432 Was Designed to Run Ada

- **Compiler generated spurious instructions**
- **Compiler did not perform common-subexpression elimination**
- **Compiler passed parameters by value/result**
  - Rather than by reference, even for large arrays
- **Compiler used slower intra-module calls even when not necessary**
- **Instructions were bit-aligned**
  - Slow to decode
- **No more than one instruction stream literal allowed**
- **Inefficient procedure calls**
  - > 1000 clock cycles including 282 wait states, compared to 100 clock cycles for non-Ada-specific processors

Intel 432: Consequences

- On benchmarks, the Intel 432 was 10x to 26x slower than a DEC Vax 11/780, the benchmark "1 MIPS" machine of the day
- On benchmarks, the Intel 432 was 2x to 23x slower than an 8-MHz Intel 8086 processor, which was a much simpler processor

- Consequently, the Intel 432 disappeared and is now barely even remembered

HLL-Specific Processors: Lessons Learned

- **It's not "all about the language"**
  - It is all about computation, communications, and algorithms
    - dot products, multiply/accumulate, zero-overhead loops, etc.
  - Processor ISAs must reflect these application needs in their function units
- **HLL tastes and fashions change**
  - Algol, Pascal, and Ada all had their day
  - No way to predict how long a language will stay fashionable or when a new language will eclipse today's in-vogue HLL
  - Processor architectures are not be as ephemeral as HLLs
    - Hardware lasts a long time
    - Code lasts only until the next dot release

Pitfall 2: Using Intermediate ISAs to Emulate More Powerful Machines

- **Usually achieved through microprogramming**
- **Benefits of intermediate ISAs**
  - Provide object-code compatibility with other processors
  - Permit the creation of a processor family with a range of price and performance
  - Permit the creation of compilers that work across a range of machines
  - Permit the processor to dynamically adapt to different HLLs at run time
  - Speed compiler development by holding target ISA constant over time
  - Reduce code size with a suitably complex ISA

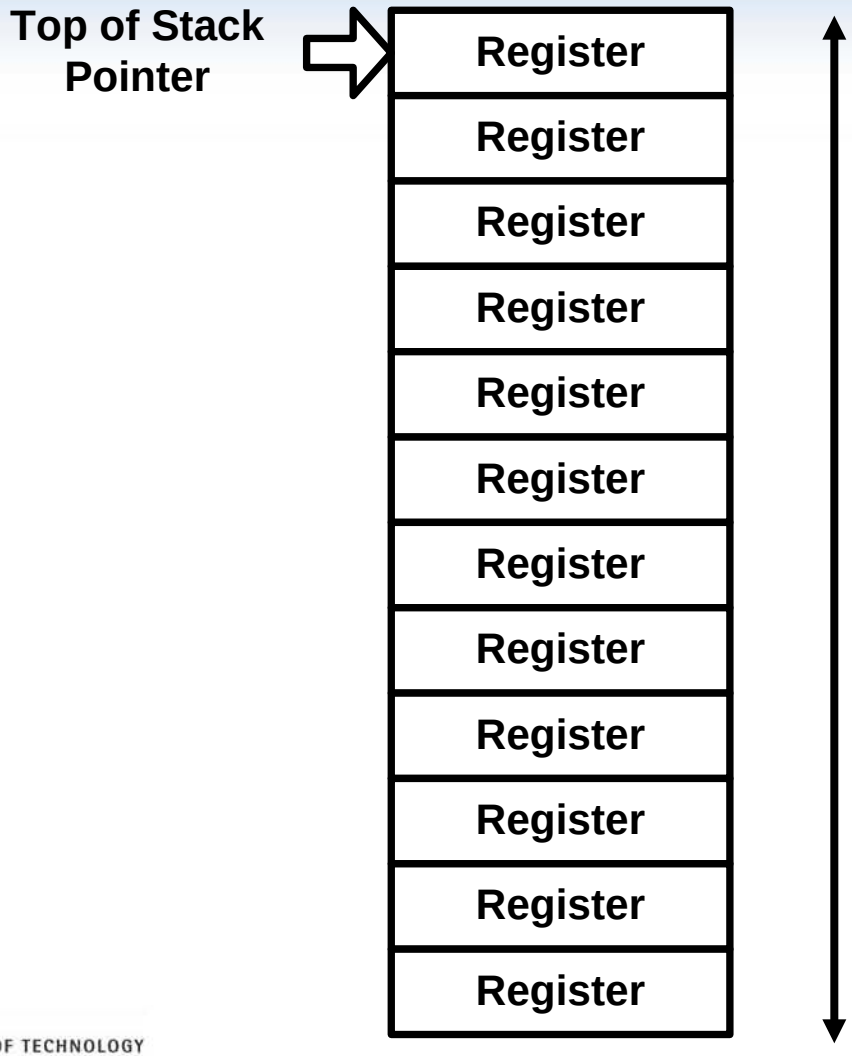
Pitfall 2: Using Intermediate ISAs to Emulate More Powerful Machines

- **Liabilities of intermediate ISAs**
  - ISA layering reduces performance versus the native, “bare-metal” ISA
  - Compilers generating code for an intermediate ISA cannot fully optimize underlying native ISA code
  - Machines optimized to multiple ISAs can contain undue performance compromises
  - Microcode for intermediate ISA can be hard to maintain because it’s infrequently scrutinized and updated

Pitfall 2: Using Intermediate ISAs to Emulate More Powerful Machines

- **Various attempts to create intermediate ISAs for the following languages:**
  - FORTRAN, EULER (An Algol variant), COBOL, RPG
  - One notable success: Java (seems successful, so far)
- **Counterexample: Using a well-established, popular ISA as an intermediate ISA**
  - AMD K6 used x86 ISA as an intermediate ISA to drive an underlying superscalar RISC machine
  - Highly successful idea, now used by all Intel, AMD x86 processors and by Transmeta's VLIW Crusoe processor

Pitfall 3: Stack Machines



Stack machines provide indirect access to a large number of registers

Pitfall 3: Stack Machines

- **Popular idea because:**
  - Stack machines provide access to a large number of fast registers (fast access to localized data)
  - “Stacks are the most basic and natural tool that can be used in processing well structured code” – Koopman
  - Machines with LIFO (last-in, first-out) stacks are required to compile computer languages – Koopman
  - “Any computer with hardware support for stack structures will probably execute applications requiring stacks more efficiently than other machines” – Koopman

http://www.ece.cmu.edu/~koopman/stack_computers/index.html

Pitfall 3: Stack Machines

- **Why are stack machines faster?**
  - Register accesses are more than an order of magnitude faster than memory accesses
  - Compilers can use registers for data storage more efficiently than any other kind of memory
  - Registers provide fast access making variable reuse more efficient

Pitfall 3: Stack Machines

- **What happened to stack machines?**
  - Like stack machines, RISC processors now have large general-purpose register files that provide fast access to large numbers of registers
  - They also offer simultaneous access to multiple registers through multiple read ports, which stack machines lack
  - Register-allocation improvements in HLL compilers exploit the ability to access multiple registers simultaneously in large RISC register files
  - RISC processors have superseded stack machines, for now

Pitfall 4: Extreme CISC and Extreme RISC

- **Extremism in ISA design comes from monotonically increasing feature sets (creeping featurism)**
  - Once a feature is designed into an ISA, it's darn hard to delete that feature because someone, somewhere has written code that depends on that feature

Pitfall 4: Extreme CISC and Extreme RISC

- **Some reasons for CISC ISA expansion**
  - Attempt to compensate for poor optimizing compilers (this attempt failed, poor compilers still optimized poorly)
    - In development of an integrated VAX, engineers found that 20% of the VAX instructions consumed 80% of the microcode and represented 0.2% of the executed instructions
  - Attempt to alleviate memory costs by reducing code size
    - With the advent and evolution of semiconductor RAM, memory cost has become a non-problem
  - Processor performance theoretically can be increased if one instruction does the work of many
    - Increasingly complex, microcoded CISC instructions consume multiple cycles, nullifying the benefits of the complex instruction
  - Complex instructions can more closely match the needs of HLLs
    - See Pitfall #1

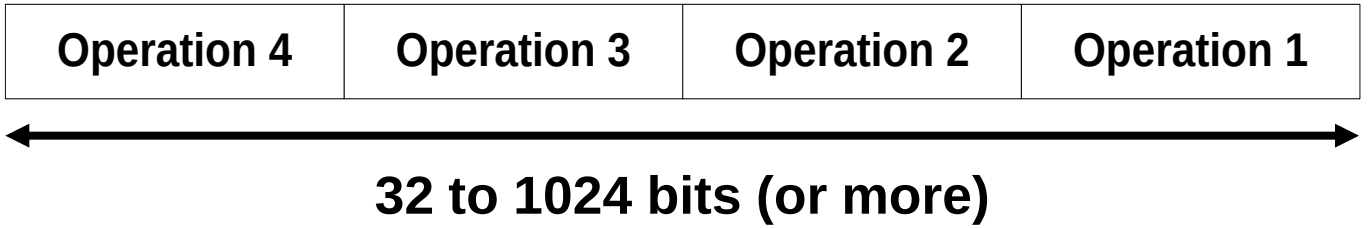
Pitfall 4: Extreme CISC and Extreme RISC

- **Some reasons for RISC ISA expansion**
  - Early RISC processors lacked floating-point instructions
    - Add floating-point unit and instructions
  - Add multiple hardware function units to exploit inherent code parallelism
    - Requires superscalar hardware, hardware schedulers, and register scoreboarding
    - 4-way superscalar designs average 1.5 instructions/clock and 6-way superscalar designs average 2.3 instructions/clock – diminishing returns
  - Branch prediction
  - Speculative instruction execution
  - Speculative code traces
  - Speculative code and data traces (superspeculation)
    - Lots of hardware (essentially multiple machines), lots of wasted calculations, lots of wasted energy

Pitfall 5: Very Long Instruction Word (VLIW) Machines

- **Use compilers to extract parallelism and simplify the hardware**
  - Each operation slot controls one or more function units
  - The compiler schedules multiple operations within each instruction
  - The Multiflow VLIW computer in the 1980s could issue 28 operations per instruction

VLIW Instruction Word with Four Operation Slots



The Problem with VLIW Machines

- **A lack of intrinsic parallelism in the code causes VLIW code bloat**
  - The compiler inserts NOPs when there's no operation that can be performed by the operation slot's corresponding function unit
  - Imagine the problem of filling 28 operation slots in each instruction

Operation 4	**NOP**	**Operation 2**	**Operation 1**
Operation 4	**Operation 3**	**Operation 2**	**Operation 1**
Operation 4	**NOP**	**NOP**	**Operation 1**
Operation 4	**Operation 3**	**Operation 2**	**NOP**

Pitfall 6: Overly Aggressive Pipelining

- **Pipelining restricts the amount of logic exercised in each stage to boost overall processor clock rate**
- **Pipelining originated with the IBM 7030 Stretch computer in 1961**
  - A supercomputer designed to be 100x faster than the fastest available computer
  - Stretch was only 70x faster, so IBM refunded 30% to the US government
  - John Cocke, developer of the IBM 801 RISC processor, was on the stretch team

Pitfall 6: Overly Aggressive Pipelining

- **A Typical 5-stage processor pipeline has the following stages:**
  - IF – instruction fetch
  - RD – read source operands from register file
  - ALU – perform the specified operation
  - MEM – read memory (for a load) or write to memory (for a store)
  - WB – write result of operation to the register file
- **RISC processors adopted pipelining and made it popular**

Pitfall 6: Overly Aggressive Pipelining

- **Pipelining is a good way to win the "clock-rate war" when clock rate is the only important figure of merit**
  - **Intel's original Pentium (P5 architecture)**
    - 5-stage pipeline
    - ~ 100 MHz max clock rate
  - **Intel's Pentium Pro (P6 architecture)**
    - 12-stage pipeline
    - 133 MHz to 1.4 GHz clock rate
  - **Intel's Pentium 4 (P7 architecture)**
    - 28-stage pipeline (including an 8-stage CISC-to-RISC converter)
    - 1.4-3.8 GHz (115W, 119A @3.8 GHz)
    - 4 GHz processor operational, but consumed too much power (>150 W) to meet PC cooling constraints

Pitfall 6: Overly Aggressive Pipelining

- **Intel's Pentium 4 (P7 architecture)**
  - 20-stage RISC pipeline
    (plus 8-stage CISC-to-RISC instruction converter)
  - Two pipeline stages are "drive" stages to accommodate signal delay
    - Therefore, "nothing" happens in 10% of the pipeline
  - Conditional branches cause large pipeline bubbles
    - The delaying effect of bubbles far more prominent in long pipelines
  - The P7 architecture is far less efficient than the P6

  - Clock rate is not necessarily equal to performance

Pitfall 7: Unbalanced Processor Design

- **Processor pipelines get a disproportionate share of attention from designers**
- **Instruction rates have increased much faster than memory bandwidth increases**
  - Instruction rate CAGR ~30%/year, 1985-2005
- **Result: attempt to compensate for the imbalance**
  - Wider buses to main memory (64-256 bits)
  - Larger, more efficient caches

Pitfall 7: Unbalanced Processor Design

- **Components of processing time**
  - Processor time
    - Actual computation time
  - Latency time
    - Memory wait time that cannot be improved with more bandwidth between memory hierarchies
  - Bandwidth time
    - Time lost to memory contention (in MP systems) and lost to inadequate memory bandwidth within memory hierarchy

Pitfall 7: Unbalanced Processor Design

- **How modern design hurts processor performance**
  - Speculation
    - Moves more instructions and data into cache (improves speed)
    - Places speculative traffic on main-memory bus (hurts speed with transactions that are ultimately evicted or discarded)
  - Multithreading
    - Reduce stalls through thread switching (improves speed)
    - Causes cache and TLB thrashing (hurts speed)
  - Clock-rate Wars
    - See Pitfall 6

Pitfall 7: Unbalanced Processor Design

- **Executing single-threaded code using simple RISC instructions drives the need for more processor speed**
- **Law of diminishing returns**
  - On-chip interconnect cannot keep up with clock-rate increases
  - On-chip power becoming unmanageable at GHz clock rates
- **Solution: drive clock rate down through parallelism**
  - More complex instructions
  - Multiple-processor (MP) system design

Pitfall 8: Omitting Pipeline Interlocks

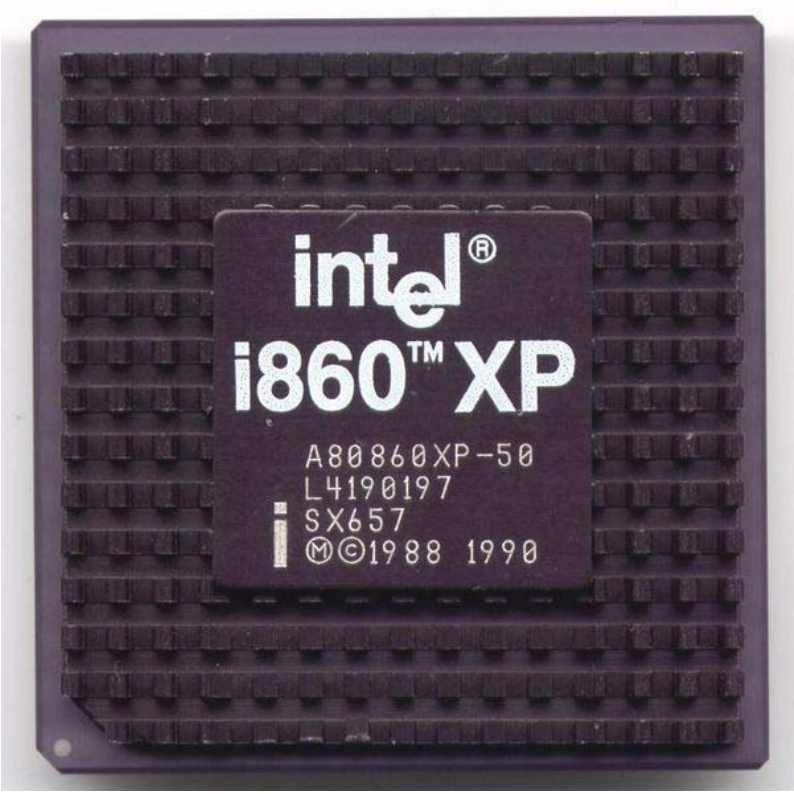
- **Hardware pipeline interlocks prevent execution of a machine instruction until it's OK to proceed**
  - Load-Use interlock prevents old register data from being used
  - Branch interlock prevents wrong instruction from being executed after a branch
    - Delay slot after the branch allows unconditionally executed code to be executed after the branch
    - Compiler may or may not find a way to use the delay slot
- **Processors that omitted interlocks:**
  - Original Stanford MIPS processor (~25K gates)
    - Microprocessor without Interlocking Pipeline Stages
    - Current MIPS processors all have hardware pipeline interlocks
  - Intel i860 floating-point unit (~ 1M transistors)
    - "explicit pipelining" for floating-point results

Pitfall 8: Omitting Pipeline Interlocks

- **Omitting the interlock was an ill-fated attempt to save hardware and nanoseconds**
  - Saves a few hundred gates, at best ~1% of a 25K-gate processor
    - Goes against Moore's Law, which provides more transistors
  - May save a few percent on instruction-cycle time.
  - But may not save any time if the interlock is not in the critical path.
- **Let the compiler schedule code using delay slots after loads and branches to eliminate the need for pipeline interlocks**
  - NP-complete scheduling problem
  - Interrupts and context switching become problematic

Pitfall 8: Omitting Pipeline Interlocks

- **Very, very bad design – DO NOT USE!!!**



Pitfall 9: Non-Power-of-2 Data Words in General-Purpose Processors

- **Early computer designs used a variety of odd data widths for a variety of reasons**
  - 12, 18, 24, 36, 48 bit word widths were common
  - 36 bits enough for integer scientific computing
  - 6-bit character codes encouraged multiples of 6 bits
    - But you need to divide addresses by 3 and 6 for 6-bit character indexing within a word
    - Bytes and double bytes are the standard for character codes now
- **This sort of processor design virtually eliminated**
- **Still used for specialized processors**
  - Example: 24-bit audio DSPs

Pop Quiz: Digital Equipment Corp

Digital Equipment Corporation (DEC) was a highly successful minicomputer company founded in 1957 by Ken Olsen and Harlan Anderson, who worked on the MIT TX-0 and TX-2, early transistorized computers with 18- and 36-bit data words respectively.

Pop Quiz: Digital Equipment Corp

Which of these DEC minicomputers used a power-of-2 data word?

PDP-1
PDP-2
PDP-3
PDP-4
PDP-5
PDP-6
PDP-7
PDP-8
PDP-9
PDP-10
PDP-11
PDP-12
PDP-13
PDP-14
PDP-15
PDP-16

Pop Quiz: Digital Equipment Corp

Which of these DEC minicomputers used a power-of-2 data word?

PDP-1	18 bits	PDP-9	18 bits
PDP-2	24 bits (never built)	PDP-10	36 bits
PDP-3	36 bits	PDP-11	16 bits
PDP-4	18 bits	PDP-12	12 bits
PDP-5	12 bits	PDP-13	Never designed
PDP-6	36 bits	PDP-14	12 bits
PDP-7	18 bits	PDP-15	18 bits
PDP-8	12 bits	PDP-16	Roll your own

http://www.village.org/pdp11/faq.pages/Crispin.html

Pop Quiz: Digital Equipment Corp Special Credit

Which minicomputer was DEC's biggest seller?

PDP-1	18 bits	PDP-9	18 bits
PDP-2	24 bits (never built)	PDP-10	36 bits
PDP-3	36 bits	PDP-11	16 bits
PDP-4	18 bits	PDP-12	12 bits
PDP-5	12 bits	PDP-13	Never designed
PDP-6	36 bits	PDP-14	12 bits
PDP-7	18 bits	PDP-15	18 bits
PDP-8	12 bits	PDP-16	Roll your own

http://www.village.org/pdp11/faq.pages/Crispin.html

Pop Quiz: Digital Equipment Corp Special Credit

Which minicomputer was DEC's biggest seller?

PDP-1	18 bits	PDP-9	18 bits
PDP-2	24 bits (never built)	PDP-10	36 bits
PDP-3	36 bits	PDP-11	16 bits
PDP-4	18 bits	PDP-12	12 bits
PDP-5	12 bits	PDP-13	Never designed
PDP-6	36 bits	PDP-14	12 bits
PDP-7	18 bits	PDP-15	18 bits
PDP-8	12 bits	PDP-16	Roll your own

http://www.village.org/pdp11/faq.pages/Crispin.html

Pop Quiz: Digital Equipment Corp Special Credit #2

What does PDP stand for?

- Programmable Digital Processor
- Programmable Data Processor
- Program Da Processor
- Processor Design Perfection

http://www.village.org/pdp11/faq.pages/Crispin.html

Pop Quiz: Digital Equipment Corp Special Credit #2

What does PDP stand for?

- Programmable Digital Processor
- Programmable Data Processor
- Program Da Processor
- Program Dis Processor

Reason: At DEC's founding, the US Congress decided that the US government had bought too many computers, and decreed that no more would be bought until those machines already acquired were fully loaded. So DEC sold PDPs, not computers.

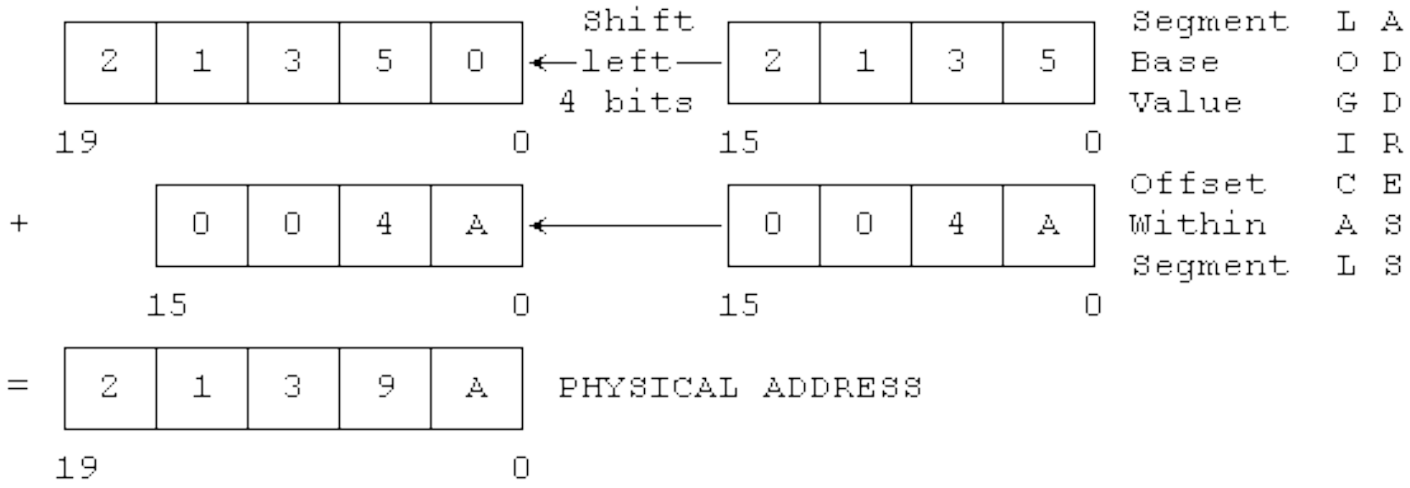
http://www.village.org/pdp11/faq.pages/Crispin.html

Pitfall 10: Overly Restricting Address Space

- **The pitfall that continues to succeed**
  - "There is only one mistake that can be made in computer design that is difficult to recover from—not having enough address bits for memory addressing and memory management." – Bell and Strecker
- **Fabulously successful designs with this flaw:**
  - DEC PDP-8, PDP-10, PDP-11
  - Intel 8051, 8080, 8086, and 80386
  - Motorola 6800
  - MOS Technology 6502
  - Zilog Z80
  - Cray 1 and Cray X-MP

Pitfall 11: Memory Segmentation

- **Patch to cure inadequate memory space**
  - Most infamous example: Intel 8086
- **Much better when coupled with an MMU**



8086 Memory-Segmentation Scheme

Pitfall 12: Multithreading

- **Remedies the divergence of processor clock speeds and memory-access times**
- **When processor stalls, waiting for memory access, give it another program thread to execute**
- **Goal: keep the processor busy**
- **Examples:**
  - Sun Niagara and Niagara II
  - IBM Power 5
  - Intel Core 2 Duo and AMD Multi-Core Opterons

Pitfall 12: Multithreading

- **Reflects “one big, expensive processor” thinking**
- **Encourages clock-rate escalation and associated rapid increase in energy consumption**

- **Alternative strategies to save energy:**
  - Chip multiprocessing
  - Potentially one processor per task (processors are cheap!)
  - Keep the clock rate down
  - Match the processor to each task

Pitfall 13: Symmetric Multiprocessing

- **One size processor fits all tasks**
- **Processors usually have a coherent view of main memory**
  - Coherent caches enforce this view
- **Examples:**
  - Sun Niagara
  - IBM Power
  - Intel dual- and quad-core Pentiums
  - AMD Multi-Core Opteron
  - ARM11 MPCore

Pitfall 13: Symmetric Multiprocessing

- **Problems with SMP**
  - One processor architecture does not match all tasks
  - Coherent caches incur high hardware costs
- **Appropriate for high-end server farms**
- **Problematic for embedded system designs due to hardware cost and task-match inefficiency**
- **AMP (asymmetric multiprocessing) a viable alternative, especially for embedded design**

Conclusions

- Plenty of negative examples to learn from
- Some mistakes apparently must be made over and over and over again
- Changing IC technology alters the balance of what works and what doesn't work
- System design evolves to absorb new technology
- Check your design assumptions frequently. If they're more than a year old, they're obsolete because technology marches on

Every Dinosaur has its Day

But occasionally, a meteor hits