Programming for the 22nd Century:

We are not programming in 1969 (or even 1984) anymore

by

Jon "maddog" Hall

Executive Director

Linux International

and

Board Chair

Linux Professional Institute

and

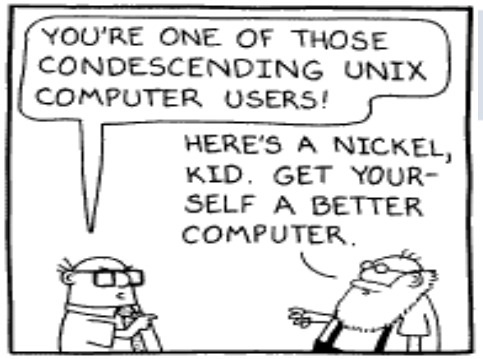
President, Project Cauã

and

CEO Emeritus of WIT

Who Am I?

- *Half* Electrical Engineer, *Half* Business, *Half* Computer Software
- In the computer industry since *1969*
  - Mainframes 5 years
  - Unix since 1980
  - Linux since 1994



- Companies (mostly large): Aetna Life and Casualty, Bell Labs, Digital Equipment Corporation (DEC, DECUS), SGI, IBM, Linaro, WIT
- Organizations: USENIX, Linux International, Linux Professional Institute
- ***Programmer***, Systems Administrator, Systems Engineer, Product Manager, Technical Marketing Manager, ***University Educator***, Author, Businessperson, Consultant
- Taught OS design and compiler design
- *Extremely* large systems to *extremely* small ones
- Pragmatic
- Vendor *and* an “open source” customer

Who Am I?
Linux

- May 1994 – funded Linus Torvalds at DECUS
  - Obtained Alpha for Linus to do port
    - CISC/RISC
    - 32/64 bit
  - Assembled DEC engineering team
- 1995 – Assumed ED role for Linux International
  - LMI – defended Linux Trademark
  - LPI – created SysAdmin certification program
  - LSB – Linux Standard Base
- Promote Linux Worldwide

Who Am I – Latin America

- 1994 – First Trip to Latin America (Venezuela)
- 1996 – Sao Paulo, Brazil
- 1998 – Montevideo, Uruguay
- 1996-2018 – Many trips to Latin America
  - 2001 – First LPI exams in Brazil
  - 2005 – Project Cauã
  - 2011 – Caninos Loucos
- 2019 – Grand Slam
  - *Build computers, create jobs, World Domination*

Linux Professional Institute

- Helped to start in 1999
  - Grow the Linux Community
  - Certify Linux Professionals
- Became Board Chair July 2015
- 2019
  - Twentieth Anniversary
  - 150,000 Certified people in 180+ countries

Who Are You?

- Entrepreneurs
- Programmers
- Future leaders of your country and the world!

"maddog, what **are** *you smoking???*

Who Needs Performance?

- “CPUs are fast enough”
  - I have been hearing that for over 50 years...
- “JAVA is the only language people need”
- “Nobody codes in assembler language any more”
- “Virtual machines make architecture knowledge obsolete”

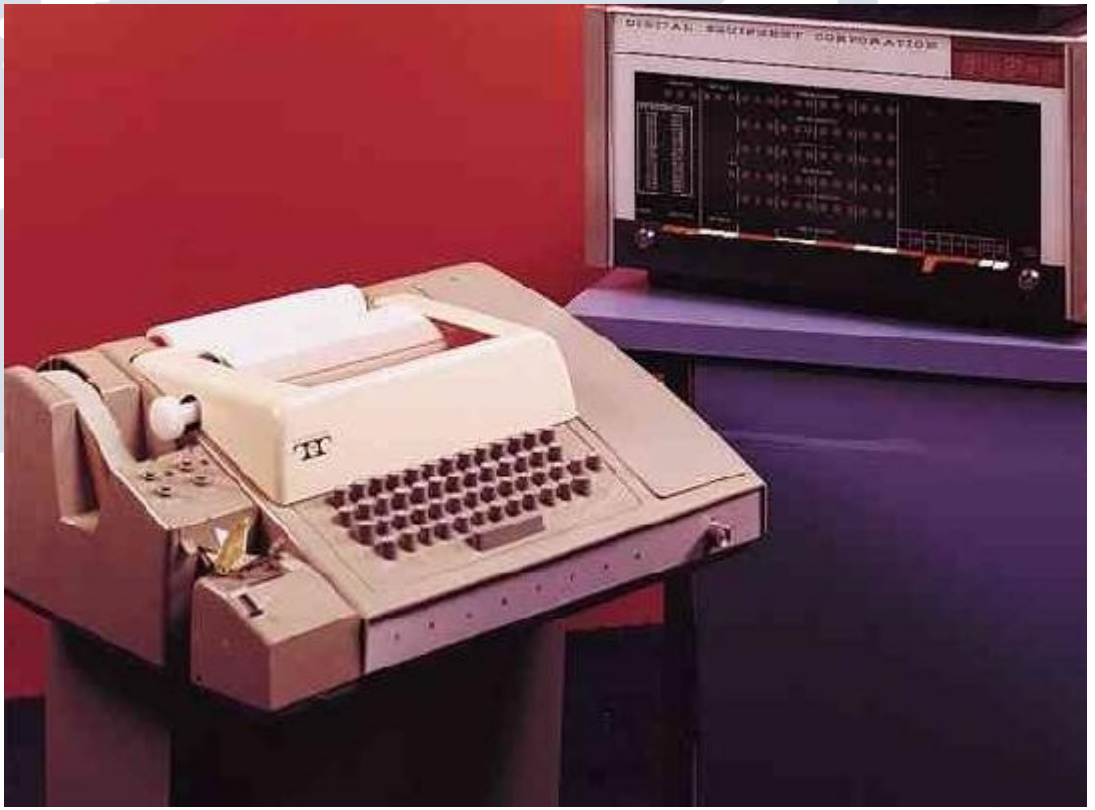
What is **your** time worth?

Performance

- "Real" problems
  - Petabytes of data, thousands of processors
- Real-time
  - REAL real-time
    - Lower those rods!
  - Linus and "soft real time"
  - Maintaining your thought path (< 1/3 second)
- Cell Phone Apps
  - Saving battery life
- Saving the environment!
  - Only 9000 servers!
- "New" (or at least newly affordable) advancements (FPGAs, DSPs)

Programming My “Second Computer”

- PDP-8 Assembler
  - 4000 12-bit words
  - 17 12-bit words
  - Data and Instructions same length
- From a book
  - Plenty of practice



Nobody told me “assembly was difficult”

To Write Really *Great* Code...

...you need to understand machine architecture and that includes machine/assembly language

Examples From My Past

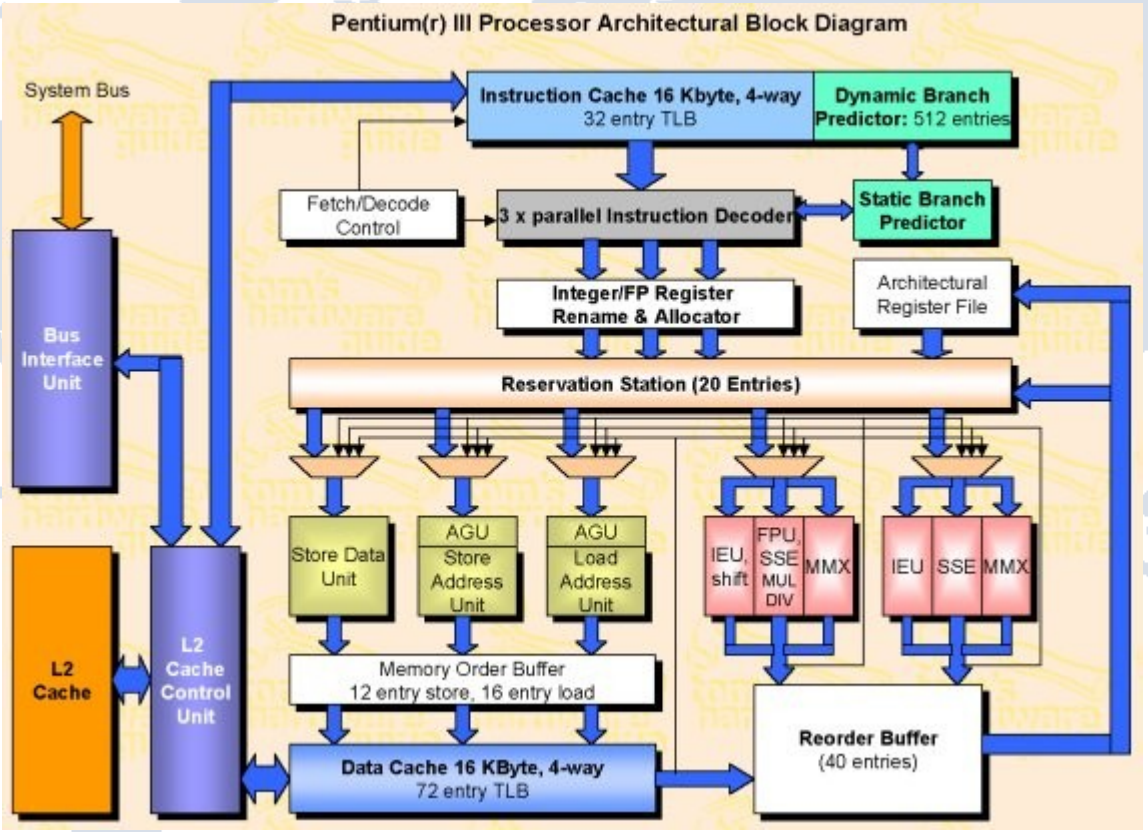
- Compiler errors (?!?)
- IBM
  - "Read Clock" - 99% of wall clock time
  - EBCDIC – the four slowest instructions
- Cache
  - Digital Unix – ½ the size, 7% faster
  - 40 times (David Mossberger-Tang)
  - 220 times (not 200%) – PDP-11/70 + RSTS-E
- Tapes (OMG!) - start/stop and streaming tapes

Today

- (Some) College Students learning
  - "Microsoft Office and Oracle" instead of "Office Systems and Databases"
  - JAVA and "IT/TI" instead of Assembly and Operating Systems
  - Virtual machines instead of "real iron"
- High school students
  - Games and HTML

How Most High School Students See Computers

How Computers Really Look



Ways of Holding Numbers

- EBCDIC/ASCII Codes
  - One “Digit” per byte
- Packed Decimal
  - Two “Digits” per byte
  - Just a “squidge less”
- Binary
- Floating point (mantissa and exponent)

Which One Is Used for Indexing?

Please do not say “EBCDIC/ASCII”

Remember:
Every Good Index Starts with Zero!

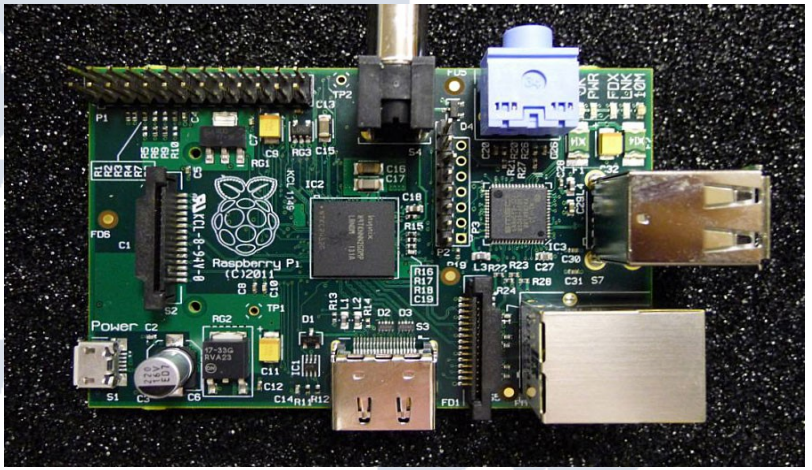
Real Life Effects of “Dumb Down”

“Incoming freshmen know less than they knew 20 years ago” - Raspberry Pi Foundation

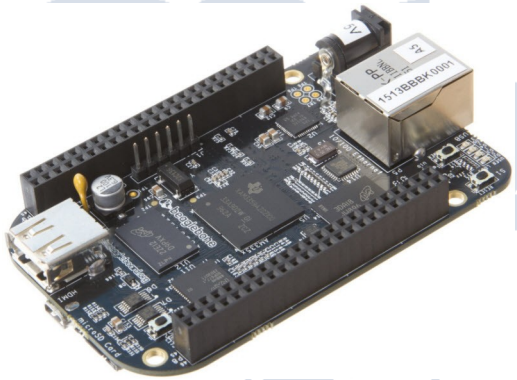
The Raspberry Pi was created to help fix this problem.

Raspberry Pi – 35 USD

- Single Core ARM – 700Mhz
- ½ Gbyte Memory
- 3D GPU
  - Hardware video decode
- USB 2.0
  - 10/100 Ethernet
- HDMI
  - Analog AV also
- GPIO Pins
- 3W

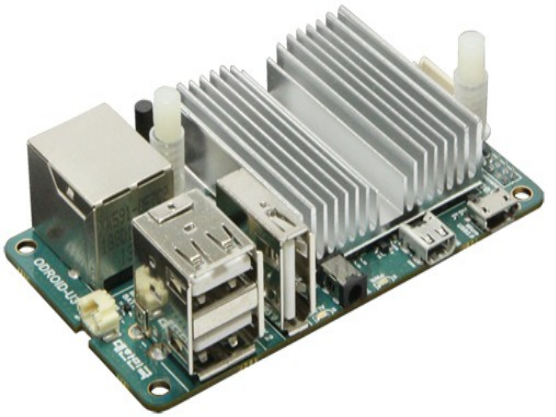


Many Little Computers: 45 USD – 199 USD

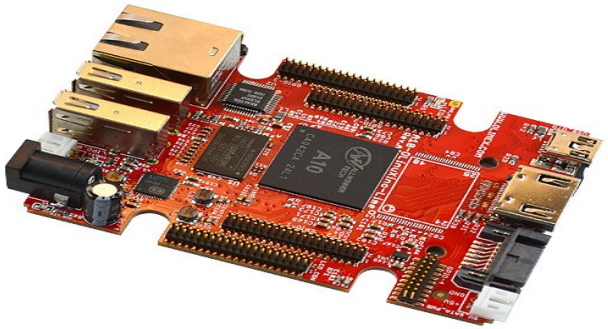


BeagleBoneBlack

Hackberry 10



ODROID-U3



OlimoX - LIME

Pandaboard

Galileo

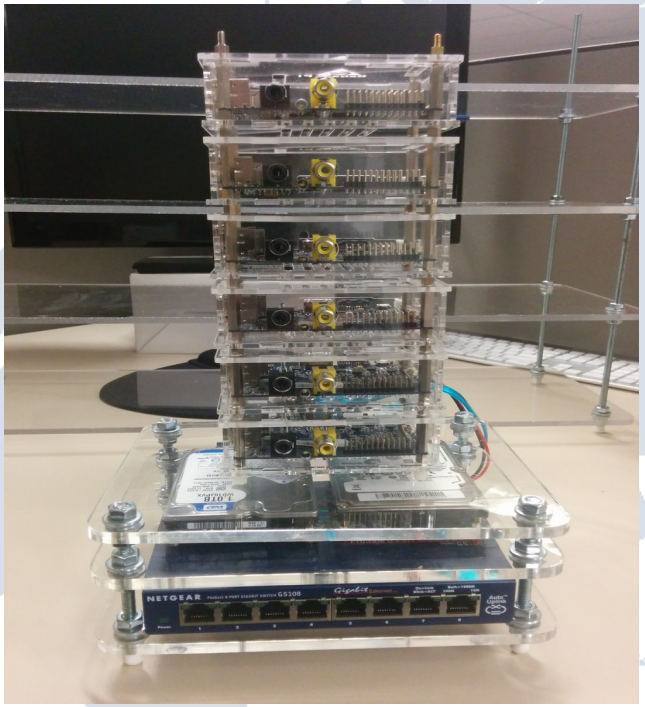
CaninosLoucos.org
IoT and more!

- Four core 32-bit ARM9 processor
- 2 Gbytes RAM
- 16GB emmc flash
- HDMI
- USB 2.0 and USB 3.0
- 10/100 Mbit/sec Ethernet
- 240 pin DIMM connector
- Open Design – four (and more) motherboards....

Why Do I Show You All This?

Because Of THIS!

- 12 ARMv7 Cores at 1 GHz each
- 6 GBytes of RAM
- 6 HDMI ports
- 6 SATA ports (currently driving two disks)
- IR on board
- 2 TB SATA disk
- 8 Port Gbit ETHERNET
- 70 Watts
- Fits in standard briefcase



Why Is This Interesting?

- Can be used to teach
  - HPC computing
  - HA computing
  - heterogeneous computing (programming and systems administration)
- Very portable, can be assembled in minutes
- Very modular
- Prototype cost: 500 USD
  - Currently using “Banana Pi”
- Production cost: < 400-1000 USD
  - Will use (6) new “Labrador/Model D”
    - Will increase from 12 up to 96 ARMv7 cores
    - Will increase from 6 GB of RAM to 48 GB RAM

Linux:
2019 - 25 Years since V1.0

- Memories have gotten larger
- CPUs have gotten faster
- CPUs are multi-core
- Algorithms have changed
- Pipelining and cache more prevalent
- Optimizations in compilers have gotten better
- Need for ***coding in*** assembly language decreased (and often is detrimental) but knowledge of it is important

Can we make the code better?

Linux Today

- All 500 of the fastest supercomputers
  - Two used to run Microsoft….
- 60% of all servers
- Most used operating system in embedded systems
- Basis of Android phones
- 10% of desktops, laptops
  - Chromebooks and tablets boost this
- Open Source (including BSD) rules the cloud

GNU/Linux: Programming For The *Future*

- "Beowulf" supercomputers (1994)
  - Non Uniform Memory Architecture (NUMA)
  - Please *do not* build one out of RPi Zeros
    - Not a big one, anyway…..
- GPUs – not just for graphics anymore
- Field Programmable Gate Arrays (FPGA)
  - More efficient than GPU
  - More flexible than an ASIC
- Quantum computing – nooooooooooo!

Can we make our code *better*?

Categories Of Performance

- % Speedup
- Memory utilization
- Cache utilization
- Algorithm replacement
- Compiler Intrinsic creation
- Compiler performance improvements
- Power efficiency (CPU vs GPU vs FPGA)
  - Battery life
- More categories?

What do we know/have today that we did not know/have 20-30 years ago?

Parallelism, Cloud and IoT

- Ways of having better performance without faster clocks
  - Parallelize code
    - Fine grained - SMP
    - Medium grained – MPI, PVM
    - Course grained - “cloud”
  - Use cloud for course grained resources
    - Be aware of security, latency and cost
    - Process raw IoT at the “edge”

“Big Cloud” vs “Small Cloud” vs “Peer-To-Peer”

- “Big Cloud” owned by “big companies”
  - usually in USA, under USA law
  - Little to no control over data
  - Never buy anything from you
- “Small Cloud” - owned by you
- “Peer-to-Peer”
  - Local storage and processing possible
    - Your country's laws
    - You can sell, buy or rent resources needed
  - Local processing of “IoT”
  - Better control on latency and costs

Side Effects of Studying Performance

- Learn really cool assembly language
- Learn code analysis techniques
- Create better code overall
- Open Source Portfolio
- Possible university level course in optimization
- Research into new optimization techniques
  - Automatic detection of race conditions?

Summary

- Learn ***SOME*** assembly/machine language (any assembly language)
- Choose your algorithms and data carefully
- Use the right language for the right job
- Examine the assembly/machine language that is generated
- Speedups of “only” 2-3x are “ok”….you don't need 1000x (but that is really cool)

Resources

- "The Definitive Guide to GCC: 2nd Edition" by William von Hagen (Apress, 2006)
- "The Art of Debugging with GDB, DDD, and Eclipse" by Matloff and Salzmann (No Starch Press, 2008)
- "Valgrind 3.3: Advanced Debugging and Profiling for GNU/Linux applications" by Seward, Nethercote et. al. (Network Theory Ltd., 2008)

More Resources

- “ARM Assembly Language – an Introduction” by J.R. Gibson (Lulu, 2007)

Questions, Comments, Ideas?