Lecture 6: Vector Processing

Professor David A. Patterson

Computer Science 252

Fall 1996

Review

- Dynmamic Branch Prediction
  - Branch History Table: 2 bits for loop accuracy
  - Correlation: Recently executed branches correlated with next branch
  - Branch Target Buffer: include branch address & prediction
- Superscalar and VLIW
  - CPI < 1
  - Dynamic issue vs. Static issue
  - More instructions issue at same time, larger the penalty of hazards
- SW Pipelining
  - Symbolic Loop Unrolling to get most from pipeline with little code expansion, little overhead

Instructon Level Parallelism

- High speed execution based on *instruction level parallelism* (ilp): potential of short instruction sequences to execute in parallel
- High-speed microprocessors exploit ILP by:

  1) pipelined execution: overlap instructions

  2) superscalar execution: issue and execute multiple instructions per clock cycle

  3) Out-of-order execution (commit in-order)
- Memory accesses for high-speed microprocessor?
  - For cache hits

Problems with conventional approach

- Limits to conventional exploitation of ILP:

  1) *pipelined clock rate*: at some point, each increase in clock rate has corresponding CPI increase

  2) *instruction fetch and decode*: at some point, its hard to fetch and decode more instructions per clock cycle

  3) *cache hit rate*: some long-running (scientific) programs have very large data sets accessed with poor locality

Vector Processors

- Vector processors have high-level operations that work on linear arrays of numbers: "vectors"

  e.g., A = BxC, where A, B, C are 64-element vectors of 64-bit floating point numbers
- Properties of vectors:
  - Each result independent of previous result
    => long pipeline, compiler ensures no dependencies
  - single vector instruction implies lots of work (≈ loop)
    => fewer instruction fetches
  - vector instructions access memory with known pattern
    => highly interleaved memory
    => amortize memory latency of over ≈ 64 elements
    => no caches required!
  - reduces branches and branch problems in pipelines

Styles of Vector Architectures

- *vector-register processors*: all vector operations between vector registers (except load and store)
  - Vector equivalent of load-store architectures
  - Includes all vector machines since late 1980s: Cray, Convex, Fujitsu, Hitachi, NEC
- *memory-memory vector processors*: all vector operations are memory to memory

Components of Vector Processor

- *Vector Register*: fixed length bank holding a single vector
  - has at least 2 read and 1 write ports
  - typically 8-16 vector registers, each holding 64-128 64-bit elements
- *Vector Functional Units* *(FUs)*: fully pipelined, start new operation every clock
  - typically 4 to 8: FP add, FP mult, FP reciprocal (1/X), integer add, logical, shift
- *Vector Load-Store Units* *(LSUs)*: fully pipelined unit to load or store a vector
- *Scalar registers*: single element for FP scalar or address
- Cross-bar to connect FUs , LSUs, registers

Example Vector Machines

Machine	Year	Clock	Regs	Elements	FUs	LSUs
Cray 1	1976	80 MHz	8	64	6	1
Cray XMP	1983	120 MHz	8	64	8	2 L, 1 S
Cray YMP	1988	166 MHz	8	64	8	2 L, 1 S
Cray C-90	1991	240 MHz	8	128	8	4
Cray T-90	1996	455 MHz	8	128	8	4
Conv. C-1	1984	10 MHz	8	128	4	1
Conv. C-4	1994	133 MHz	16	128	3	1
Fuj. VP200	1982	133 MHz	8-256	32-1024	3	2
Fuj. VP300	1996	100 MHz	8-256	32-1024	3	2
NEC SX/2	1984	160 MHz	8+8K	256+var	16	8
NEC SX/3	1995	400 MHz	8+8K	256+var	16	8

Vector Linpack Performance

Machine	Year	Clock	100x100	1kx1k	Peak(Procs)
• Cray 1	1976	80 MHz	12	110	160(1)
• Cray XMP	1983	120 MHz	121	218	940(4)
• Cray YMP	1988	166 MHz	150	307	2,667(8)
• Cray C-90	1991	240 MHz	387	902	15,238(16)
• Cray T-90	1996	455 MHz	705	1603	57,600(32)
• Conv. C-1	1984	10 MHz	3	--	20(1)
• Conv. C-4	1994	135 MHz	160	2531	3240(4)
• Fuj. VP200	1982	133 MHz	18	422	533(1)
• NEC SX/2	1984	166 MHz	43	885	1300(1)
• NEC SX/3	1995	400 MHz	368	2757	25,600(4)

"DLXV" Vector Instructions

Instr.	Operands	Operation	Comment
• `ADDV`	`V1,V2,V3`	V1=V2+V3	vector + vector
• `ADDSV`	`V1,F0,V2`	V1=F0+V2	scalar + vector
• `MULTV`	`V1,V2,V3`	V1=V2xV3	vector x vector
• `MULSV`	`V1,F0,V2`	V1=F0xV2	scalar x vector
• `LV`	`V1,R1`	V1=M[R1..R1+63]	load, stride=1
• `LVWS`	`V1,R1,R2`	V1=M[R1..R1+63*R2]	load, stride=R2
• `LVI`	`V1,R1,V2`	V1=M[R1+V2i,i=0..63]	indir.("gather")
• `CeqV`	`VM,V1,V2`	VMASKi = (V1i=V2i)?	comp. setmask
• `MOV`	`VLR,R1`	Vec. Len. Reg. = R1	set vector length
• `MOV`	`VM,R1`	Vec. Mask = R1	set vector mask

DAXPY (Y = a * X + Y)

Assuming vectors X, Y
are length 64

Scalar vs. Vector

```
LD     F0,a          ;load scalar a
LV     V1,Rx         ;load vector X
MULTS  V2,F0,V1      ;vector-scalar mult.
LV     V3,Ry         ;load vector Y
ADDV   V4,V2,V3      ;add
SV     Ry,V4         ;store the result
```

```
        LD      F0,a
        ADDI    R4,Rx,#512   ;last address to load
loop:   LD      F2, 0(Rx)    ;load X(i)
        MULTD   F2,F0,F2     ;a*X(i)
        LD      F4, 0(Ry)    ;load Y(i)
        ADDD    F4,F2, F4    ;a*X(i) + Y(i)
        SD      F4 ,0(Ry)    ;store into Y(i)
        ADDI    Rx,Rx,#8     ;increment index to X
        ADDI    Ry,Ry,#8     ;increment index to Y
        SUB     R20,R4,Rx    ;compute bound
        BNZ     R20,loop     ;check if done
```

578 (2+9*64) vs.
6 instructions:

64 operation vectors +
no loop overhead

also 64X fewer pipeline
hazards

CS 252 Administrivia

- **Projects this Friday?**

Vector Execution Time

- Time = f(vector length, data dependicies, struct. hazards)
- *Initiation rate*: rate that FU consumes vector elements (usually 1 or 2 on Cray T-90)
- *Convoy*: set of vector instructions that can begin execution in same clock (no struct. or data hazards)
- *Chime*: approx. time for a vector operation
- *m* convoys take *m* chimes; if each vector length is n, then they take approx. *m* x *n* clock cycles (ignores overhead; good approximization for long vectors)

```
1:  LV      V1,Rx      ;load vector X
2:  MULS    V2,F0,V1   ;vector-scalar mult.
    LV      V3,Ry      ;load vector Y
3:  ADDV    V4,V2,V3   ;add
4:  SV      Ry,V4      ;store the result
```

4 conveys
=> 4 x 64 ≈ 256 clocks
(or 4 clocks per result)

DLXV Start-up Time

- *Start-up time*: pipeline latency time (depth of FU pipeline); another sources of overhead
- Operation Start-up penalty (from CRAY-1)

• Vector load/store	12
• Vector multply	7
• Vector add	6

Assume convoys don't overlap; vector length = n:

Convoy	*Start*	*1st result*	*last result*	
1. LV	0	12	11+n	
2. MULV, LV	12+n	12+n+12	24+2n	*Load start-up*
3. ADDV	25+2n	25+2n+6	30+3n	*Wait convoy 2*
4. SV	31+3n	32+3n+12	42+4n	*Wait convoy 3*

Vector Load/Store Units & Memories

- **Start-up overheads usually longer fo LSUs**
- **Memory system must sustain 1 word/clock cycle**
- **Many Vector Procs. use banks vs. simple interleaving:**

  1) support multiple loads/stores per cycle
  => multiple banks & address banks independently

  2) support non-sequential accesses (see soon)
- **Note: No. memory banks > memory latency to avoid stalls**
  - ***m* banks => *m* words per memory lantecy *l* clocks**
  - **if *m* < *l*, then gap in memory pipeline:**

clock:	**0**	**…**	***l***	***l*+1**	***l*+2**	**…**	***l*+*m*- 1**	**l+m**	**…**	**2 *l***
word:	**--**	**…**	**0**	**1**	**2**	**…**	***m*-1**	**--**	**…**	***m***

Vector Length

- **What to do when vector length is not exactly 64?**
- ***vector-length register*** **(VLR) controls the length of any vector operation, including a vector load or store. (cannot be > the length of vector registers)**

```
      do 10 i = 1, n
10    Y(i) = a * X(i) + Y(i)
```

- **Don't know n until runtime!**
  n > Max. Vector Length (MVL)?
- ***Strip mining*****: generation of code such that each vector operation is done for a size ≤ to the MVL**

Strip Mining

- Suppose Vector Length > Max. Vector Length (MVL)?
- *Strip mining*: generation of code such that each vector operation is done for a size ≤ to the MVL
- 1st loop do short piece (n mod MVL), rest VL = MVL

```
    low = 1
    VL = (n mod MVL)  /*find the odd size piece*/
    do 1 j = 0,(n / MVL)  /*outer loop*/
        do 10 i = low,low+VL-1  /*runs for length VL*/
                Y(i) = a*X(i) + Y(i)  /*main operation*/
10      continue
        low = low+VL  /*start of next vector*/
        VL = MVL  /*reset the length to max*/
1       continue
```

Vector Metrics

- R_∞—MFLOPS rate on an infinite-length vector
  - Real problems do not have unlimited vector lengths, and the start-up penalties encountered in real problems will be larger
  - (R_n is the MFLOPS rate for a vector of length n)
- $N_{1/2}$—The vector length needed to reach one-half of R_∞
  - a good measure of the impact of start-up
- N_V—The vector length needed to make vector mode faster than scalar mode
  - measures both start-up and speed of scalars relative to vectors

Vector Stride

- **Suppose adjacent elements not sequential in memory**

```
  do 10 i = 1,100
      do 10 j = 1,100
           A(i,j) = 0.0
           do 10 k = 1,100
10                  A(i,j) = A(i,j)+B(i,k)*C(k,j)
```

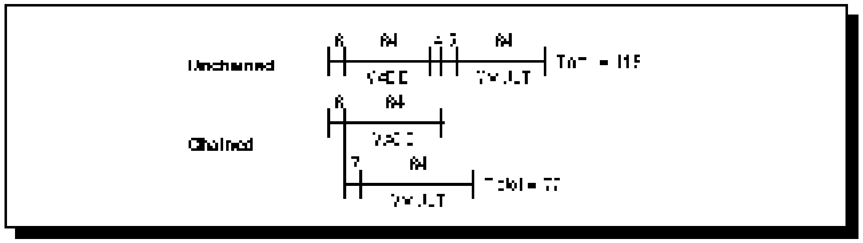
- **Either B or C accesses not adjacent (800 bytes between)**
- ***stride*: distance separating elements that are to be merged into a single vector (caches do unit stride) => `LVWS` instruction**
- **Strides => can cause bank conflicts (e.g., stride = 32 and 16 banks)**

Compiler Vectorization on Cray XMP

Benchmark	%FP	%FP in vector
ADM	23%	68%
DYFESM	26%	95%
FLO52	41%	100%
MDG	28%	27%
MG3D	31%	86%
OCEAN	28%	58%
QCD	14%	1%
SPICE	16%	7%
TRACK	9%	23%
TRFD	22%	10%

Vector Opt #1: Chaining

- Suppose:
- MULTV <u>V1</u>,V2,V3
- ADDV V4,<u>*V1*</u>,V5 ; separate convoy?
- ***chaining***: vector register (V1) is not as a single entity but as a group of individual registers, then pipeline forwarding can work on individual elements of a vector
- ***Flexible chaining***: allow vector to chain to any other active vector operation => more read/write port
- As long as enough HW, increases convoy size



Vector Opt #2: Sparse Matrices

- Suppose:

```
             do   100 i = 1,n
100          A(K(i)) = A(K(i)) + C(M(i))
```

- *gather* (`LVI`) operation takes an *index vector* and fetches the vector whose elements are at the addresses given by adding a base address to the offsets given in the index vector => a nonsparse vector in a vector register
- After these elements are operated on in dense form, the sparse vector can be stored in expanded form by a *scatter* store (`SVI`), using the same index vector
- Can't be done by compiler since can't know Ki elements distinct, no dependencies; by compiler directive
- Use `CVI` to create index 0,m, 2m, ..., 63m

Sparse Matrix Example

- Cache (1993) vs. Vector (1988)

	IBM RS6000	Cray YMP
Clock	72 MHz	167 MHz
Cache	256 KB	0.25 KB
Linpack	140 MFLOPS	160 (1.1)
Sparse Matrix (Cholesky Blocked)	17 MFLOPS	125 (7.3)

Vector Opt #3: Conditional Execution

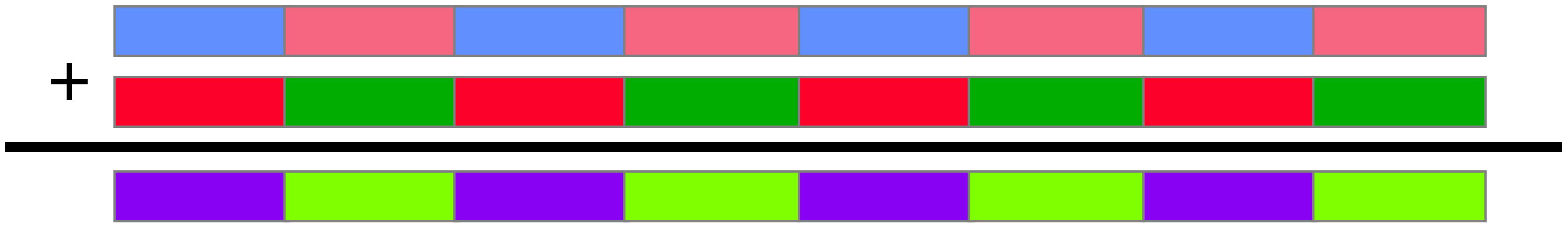
- Suppose:

```
    do 100 i = 1, 64
           if (A(i) .ne. 0) then
                  A(i) = A(i) – B(i)
           endif
100 continue
```

- *vector-mask control* takes a Boolean vector: when *vector-mask register* is loaded from vector test, vector instructions operate only on vector elements whose corresponding entries in the vector-mask register are 1.
- Still requires clock even if result not stored; if still performs operation, what about divide by 0?

Vector for Multimedia?

- MMX: 57 new 80x86 instructions (1st since 386)
  - similar to Mot. 88110, HP PA-71000LC, UltraSPARC
- 3 data types: 8 8-bit, 4 16-bit, 2 32-bit in 64bits
  - reuse 8 FP registers (FP and MMX cannot mix)
- ≈ short vector: load, add, store 8 8-bit operands



- Claim: overall speedup 1.5 to 2X for 2D/3D graphics, audio, video, speech, comm., ...
  - use in drivers or added to library routines; no compiler

MMX Instructions

- Move 32b, 64b
- Add, Subtract in parallel: 8 8b, 4 16b, 2 32b
  - opt. signed/unsigned saturate (set to max) if overflow
- Shifts (sll,srl, sra), And, And Not, Or, Xor
  in parallel: 8 8b, 4 16b, 2 32b
- Multiply, Multiply-Add in parallel: 4 16b
- Compare = , > in parallel: 8 8b, 4 16b, 2 32b
  - sets field to 0s (false) or 1s (true); removes branches
- Pack/Unpack
  - Convert 32b<–> 16b, 16b <–> 8b
  - Pack saturates (set to max) if number is too large

Vector Pitfalls

- Pitfall: Concentrating on peak performance and ignoring start-up overhead: N_V (faster than scalar) > 100!
- Pitfall: Increasing vector performance, without comparable increases in scalar performance (Amdahl's Law)
  - failure of Cray competitor from his former company
- Pitfall: Good processor vector performance without providing good memory bandwidth
  - MMX?

Vector Summary

- Alternate model accomodates long memory latency
- Much easier for hardware: more powerful instructions, more predictable memory accesses, fewer harzards, fewer branches, fewer mispredicted branches, ...
- What % of computation is vectorizable?
- Is vector a good match to new apps such as multidemia, DSP?

5 minute Class Break

- **Lecture Format:**
  - ≈ 1 minute: review last time & motivate this lecture
  - ≈ 20 minute lecture
  - ≈ 3 minutes: discuss class manangement
  - ≈ 25 minutes: lecture
  - 5 minutes: break
  - ≈25 minutes: lecture
  - ≈1 minute: summary of today’s important topics

XSPEC ‘02

- eXperimental SPEC for 2002: create benchmark set of programs, data size of future apps
  - Why design computers of future using programs of past?
  - e.g., Speech understanding, Word processing of book, Video, 3D animation, sound, Object data base, Encryption, Just-In-Time compiler, Network apps, Games, …
  - Public domain and third party programs
  - At least 2 programs, 3 data set sizes (‘96, ‘99, ‘02), ported to at least 2 instruction sets (with optimizating compilers), characterize using performance monitors (cache misses, CPI, instruction types, I/O traffic, paging, ...)
  - success requires potability and publishing results
  - can beveral groups

JAVA vs. SPEC

- JAVA 1: characterization of cache behavior, register usage, branch behavior, and depth of calls (to name a few interesting data points) for Java applications.
  - Contrast these parameters with Spec benchmarks.
  - Analyze where differences come from
  - Compare with vanilla byte codes and Just-In-Time compilers
  - Suggested by Dileep Bhandarkar (Dileep_Bhandarkar@ccm.sc.intel.com)

SPEC95 Path Length

- Since you have a variety of architectures, take the gcc compiler and measure SPECint95 on some public version of Unix such as Linux or FreeBSD
  - You already know about ATOM (alpha) and EEL (sparc). Now there is also ETCH for x86. Look at:

    http://www.cs.washington.edu/homes/bershad/Etch/index.html

    Having similar tools for 3 architectures might allow you to have 3 groups of students look at similar stuff on 3 architectures.
  - Compare path lengths for the various architectures
  - Do static code size comparisons
  - See impact of optimizations
  - Suggested by Dileep Bhandarkar (Dileep_Bhandarkar@ccm.sc.intel.com)

SPEC95 Caches Tables

- **Repeat Mark Hill's SPEC92 cache analysis for SPEC95.**
  - **Gee, J.D.; Hill, M.D.; Pnevmatikatos, D.N.; Smith, A.J. “Cache performance of the SPEC92 benchmark suite.” *IEEE Micro*, Aug. 1993, vol.13, (no.4):17-27.**
  - **Abstract: The authors consider whether SPECmarks, the figures of meritobtained from running the SPEC benchmarks under certain specified conditions, accurately indicate the performance to be expected from real, live work loads. Miss ratios for the entire set of SPEC92 benchmarks are measured.**
  - **This may need multiple teams, N benchmarks per team.**
  - **See if it varies for x86 vs. RISC? Use NOW, PC clusters.**
  - **Suggested by Dileep Bhandarkar (Dileep_Bhandarkar@ccm.sc.intel.com)**

Measure OS Primitives

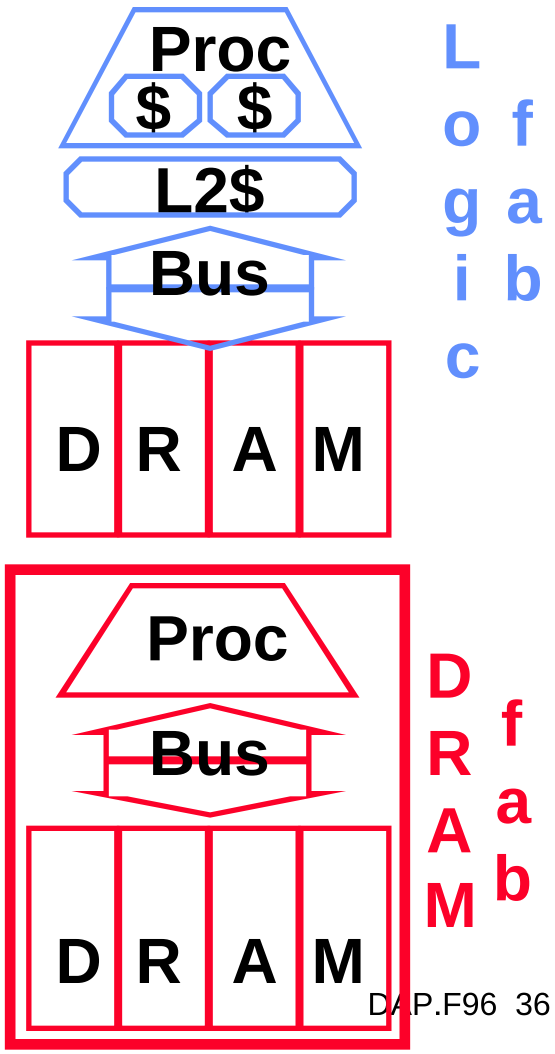
- Pick some set of OS primitives (process creation, synchronization etc) and measure the times for NT vs Unix on same platform.
  - See if it varies for x86 vs. Alpha?
  - Suggested by Dileep Bhandarkar (Dileep_Bhandarkar@ccm.sc.intel.com)

A "voting" data-prefetch engine

- This H/W device has the following characteristics:
  - For any data reference stream, TWO independent prefetch devices make predictions: one is a standard load address stride predictor (which predicts strided accesses), and the other is a stream buffer, which basically reacts to cache miss history.
  - The challenge is to design a voting function that dynamically selects one or the other of the prefetch addresses to issue to the higher levels of the memory hierarchy.
  - This selection should be made conditional on whichever of the two predictors is currently generating the more accurate future address stream.
  - Accuracy is defined as the ability to reduce future data cache misses.
  - Suggested by Sharad Mehrotra (Sharad.Mehrotra@Eng.Sun.COM)

IRAM Vision Statement

- **Microprocessor & DRAM on single chip:**
  - **bridge the processor-memory performance gap via on-chip latency (5-10X) & bandwidth (100X)**
  - **improve power-performance (no DRAM bus: 2-3X)**
  - **lower minimum memory size (designer picks any amount)**



Potential IRAM CS252 Projects?

- **Large program/data solution**
  - **Get traces of large programs**
  - **Compare and contrast:**
    - **Cache Only Memory Architecture, all IRAMs**
    - **IRAM as cache, external DRAM as main memory**
    - **IRAM as low physical memory, external DRAM as high physical memory; paging policy (CS 262)?**
- **Examine on-the-fly compression, decompression for external BW, capacity**
- **Model processor redundancy to improve yield of IRAM**
  - **Dynamic pipeline and multiple functional units**
  - **Multiple small processors?**

Brass Vision Statement

- Microprocessor & FPGA on single chip:
  - use some of millions of transitors to customize HW dynamically to application
  - “Reconfigurable Computing”

Potential BRASS CS252 Projects?

- Comparison between full-custom, FPGA, and processor implementations
- There are a few instances (e.g. multipliers, FIRs) where we have custom IC, FPGA, and processor implementations and can compare the area-time efficiency of each.
- Pick an application which has a plausible custom implementation to compare against and develop/estimate a good processor and array (FPGA/GARP-array/etc.) implementation (perhaps include the processor+FPGA hybrid, as well).
  - Suggested by André DeHon (amd@CS.Berkeley.edu)

Potential BRASS CS252 Projects?

- **More suggestions by André DeHon; see him for more details (amd@CS.Berkeley.edu)**
- **FPGAs to accelerate common APIs**
- **Specialization Opportunity**
- **Coping with Finite Array Size**
- **Multitasking and Configurable Arrays**
- **Important Program Characteristics**
    - **(derivable) datasizes (1b vs 8b vs 16b)**
    - **retiming distances (space to save vs. wait)**
    - **richness of interconnect**

Architecture Archeology/ Endangered Species Act

- documenting architectural history might attempt to either collect or construct emulators for machines which are disappearing
  - The real wonder for the ARPAnet for me in 1973 was the diversity of architecture. I started on an IBM 360/75, I believed at that timethat the world revolved around EBCDIC. Over the next couple of years encountered my first DEC-10, ILLIAC-IV, CDC-6600, ...
  - The value of emulation history is going to take on interesting significance in the future. The challenge will be to preserve this software history as the base emulation machines themselves pass into history.
  - Write emulators in Java so can run anywhere?Simple assembler so can write programs?
  - Suggested by Eugene Miya (eugene@pioneer.arc.nasa.gov)