PHYS. BRIEF 85-602

THE S-1 PROJECT: Advancing the Digital Computing Technology Base for National Security Applications

An Open House Presentation for
Members and Guests of the Navy and DoE Communities

Livermore, CA
11 April 1985

S-1 Technical Staff
Lawrence Livermore Nat'l. Lab
(415)422-0758

Agenda

- Overview
- SCALD
- Mark IIA Architecture and Implementation
- Software Overview
- Current Status
- Architectural Studies
- AAP — Advanced Architecture Processor
- Advanced Electronic Packaging
- Laser Pantography Overview

Overview

Lowell Wood

S-1 Project History

- Formally commenced in FY '77 with ONR support
    - Ab initio high performance emphasis and MIMD multiprocessor emphasis
    - Tool-building and -using theme
    - 6.2 support
- NAVELEX supervision commenced in FY '79
    - Technology transfer to industry mandate
    - SCALD I distribution
    - First major user codes on Mark I system
- First multiprocessor-destined supercomputer (Mark IIA) completed construction in 1981
    - SCALD II distribution
    - Commencement of CAD/CAM/CAE industry
- Billion bit high speed memory attached to Mark IIA in 1984
    - Major user codes exercised on system
- Multi-user operating system and HOL support in FY '85
    - Advances T&E support

The S-1 is a Computing Technology Development Project Funded by the Navy and DoE

- Design System
  - Widely applicable tool for rapid, low cost design
  - Supports reimplementation to capture technology advances
  - Conceptual basis of new CAE industry
- Uniprocessor Systems
  - Combined signal processor and general purpose processor
  - Highest performance implementation consistant with cost-efficiency
- Multiprocessor Systems
  - Uniquely great throughput
  - High reliability through automatically invoked redundancy
  - Uniprocessor cost-effectiveness
- Software
  - Support evaluation of uni- and multi-processor systems
  - High level languages emphasized for transportability
  - Multi-tasking, real-time operating system support

S-1 Family of Computer Systems

- Mark I
  - 5300 ECL-10K chips
  - Operational in 1979
  - Vehicle for SCALD development
- Mark IIA
  - 27,000 ECL-10K and ECL-100K chips
  - Operational in 1984
  - Addresses DoD and DoE applications
  - Available to outside users upon software installation
- AAP (Advanced Architecture Processor)
  - Vehicle for advanced architecture packaging studies
  - ECL-100K and semicustom ECL gate-arrays implementation
  - High-density, water-cooled packaging
  - Suitable for WSI implementation
- Mark V
  - Wafer-scale integrated packaging
  - Full military environmental compatibility
    - Certifiably wartime rad-hard

SCALD

Mike Farmwald

SCALD System Overview

- Proven system for design of complex systems
  - Used in design of S-1 family of computer
  - Permits small design staffs
  - Minimizes design and manufacturing errors
- Comprehensive automation of design process
  - Inputs a graphics-based hierarchical design
  - Verifies logical and electrical consistency
  - Outputs instructions for automatic assembly and debugging
  - Maintains up-to-date documentation
- Transferred to industry
  - Over 200 sites have received LLNL release
  - Dozen companies offering SCALD-like products

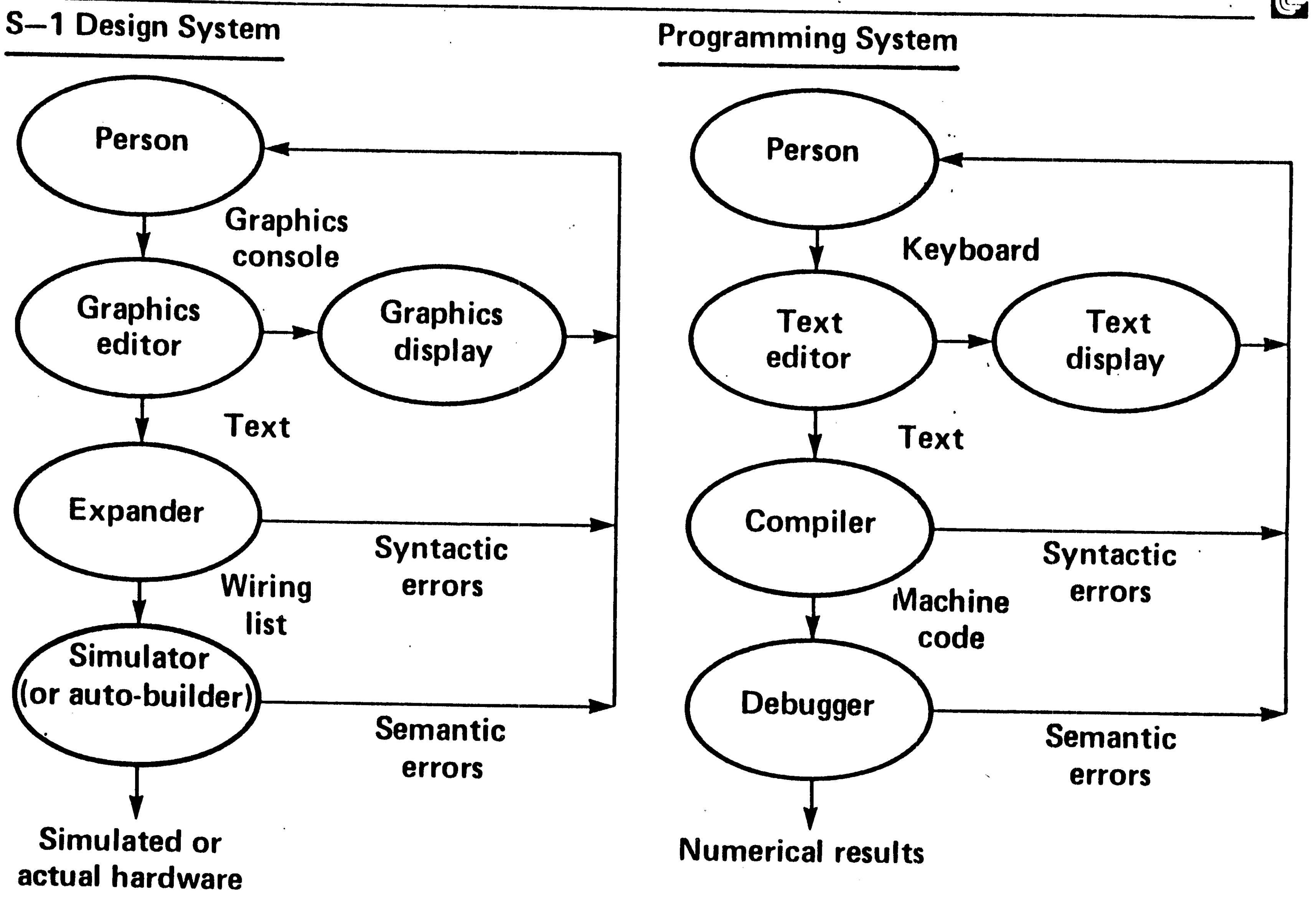
Conventional Logic Design

- Designers use one or a few fixed levels of abstraction
  - Gates, flip-flops, and other available devices
- Computer-aided layout and wire-listing is often available
- Computer-assisted drawing is sometimes available
- Large computer developments typically cost $\geq$ 100 man-years in the design stage
  - Amdahl
  - Burroughs
  - CDC
  - IBM
- Design costs have usually been small fractions of total product cost (high volume systems)
- Economic penalty is in technological obsolescence of marketed systems
  - Has become stiff only recently (LSI revolution)
  - Industry is beginning to automate logic design

SCALD: The Fundamental Difference

- SCALD is a high-level hardware-language compiler
  - Closely analogous to a high-level software-language compiler
  - Inputs a high-level description
  - Outputs hardware
- Arbitrary modules are designed
  - Each in terms of a few other modules
  - Relatively independently
  - To communicate through well-defined interfaces
- SCALD advantages are:
  - Increased understandability of resulting design
    - Reducing design time
    - Enhancing design correctness
  - Facilitation of final documentation
  - Increased changeability of design
  - Increased computer-verifiability of design

COMPUTER AIDED LOGIC DESIGN VERSUS COMPUTER AIDED PROGRAM DESIGN



S-1 Project's Philosophy of CAD/CAM Development

- Develop CAD/CAM systems in the immediate context of doing design
  - Only way to really understand what is needed
    - What are bottlenecks in getting a large design done?
  - Provides rapid feedback about effectiveness of various algorithms
  - Eliminates Tower-of-Babel-ism
  - Only create capabilities that are needed to get job done

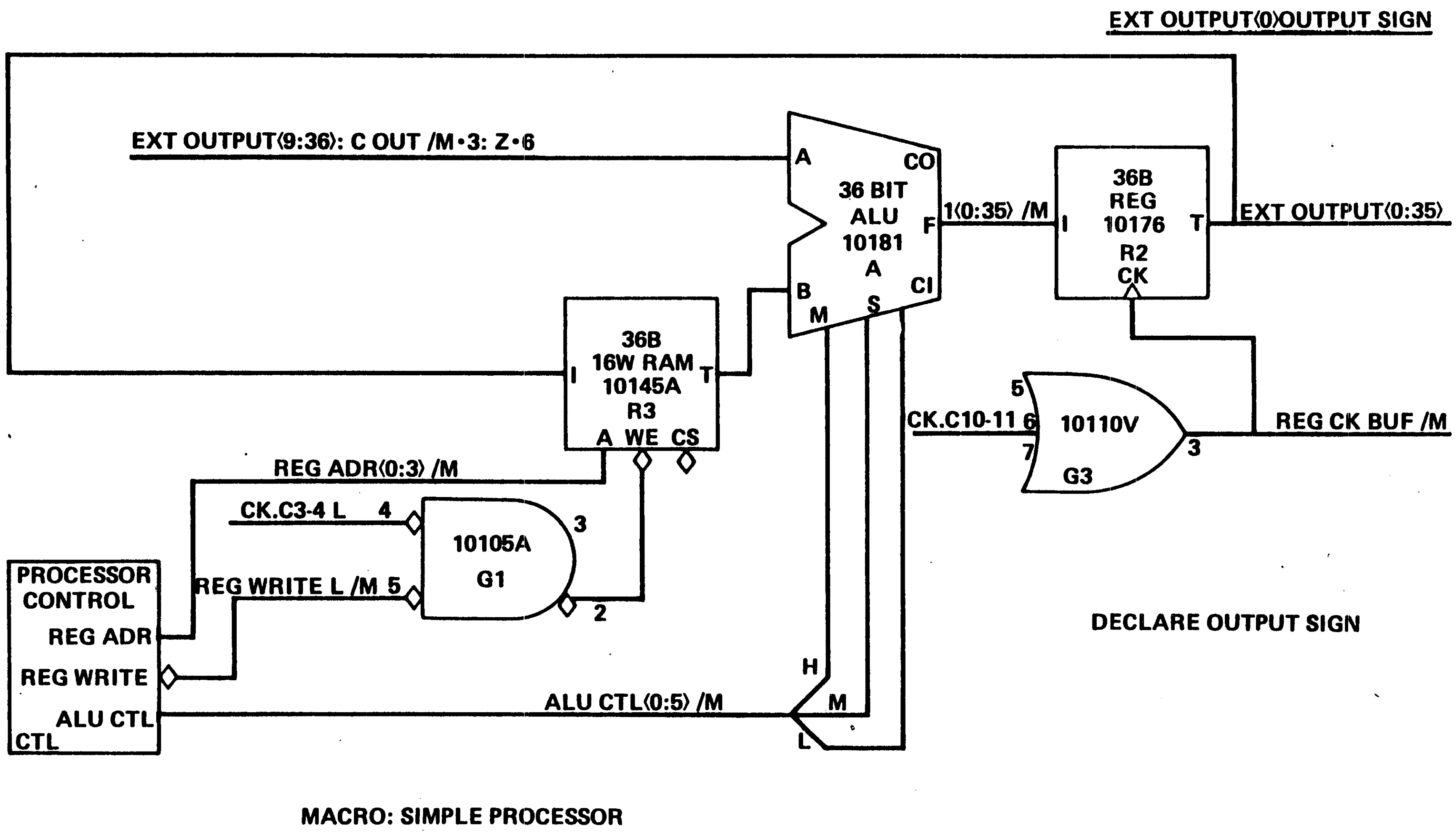
SCALD System Components

- Available
  - Drawing system (external)
  - Macro expander
    - Compiles hierarchical design
  - Timing verifier
    - Checks for timing errors
  - Logic simulator
    - Interactive debugger for designs
  - Packager
    - Checks electrical rules
    - Writes implementation tapes
  - Micro debugger (MD)
    - Interactive debugger for hardware
  - Micro assembler
    - Produces binary mirocode from symbolic descriptions of format and code

SCALD System Components (continued)

- Under development
  - Test pattern generator
    - Automated hardware diagnostics
  - Placement and routing software
    - Automated generation of PC, gate-arrays, WSI

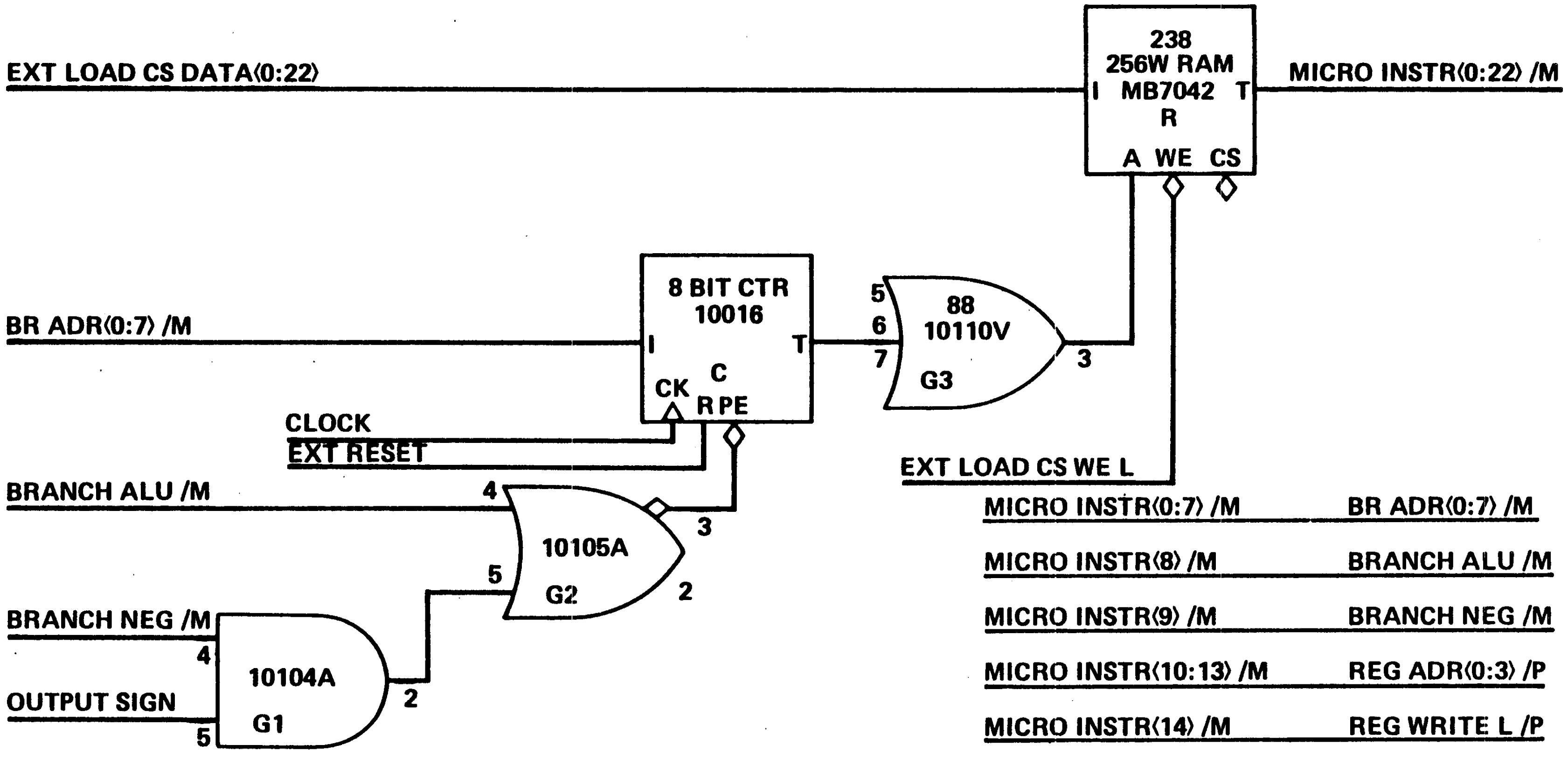
EXAMPLE SCALD MACRO DEFINITION—SIMPLE PROCESSOR



EXAMPLE SCALD MACRO DEFINITION–PROCESSOR CONTROL

PARAMETER

REG ADR⟨0:3⟩
REG WRITE L
ALU CTL⟨0:6⟩



MACRO: PROCESSOR CONTROL

MICRO INSTR⟨0:7⟩ /M	BR ADR⟨0:7⟩ /M
MICRO INSTR⟨8⟩ /M	BRANCH ALU /M
MICRO INSTR⟨9⟩ /M	BRANCH NEG /M
MICRO INSTR⟨10:13⟩ /M	REG ADR⟨0:3⟩ /P
MICRO INSTR⟨14⟩ /M	REG WRITE L /P
MICRO INSTR⟨15:20⟩ /M	ALU CTL⟨0:5⟩ /P

S-1 Design Statistics

	Mark I	Mark IIA
IC population (DIPs)	5500	25000
Drawings (pages)	280	1000
Design effort (person-years)	2	8
Scalar rate (MFLOPS)	1	8
Vector rate (MFLOPS)	-	50
HW f-p FFT (MFLOPS)	-	250

SCALD Timing Verifier

SCALD Timing Verifier Goals

- To verify all timing constraints in large synchronous sequential digital systems
- To verify timing constraints early and throughout the design cycle
- To be driven mainly from the design description
  - Avoiding complex auxiliary files that the designer must generate
- To verify as much as possible of the timing constraints in a "value-independent" fashion
  - To minimize the number of cases that need to be tested
    - To reduce CPU time

Timing Verification in the SCALD System

- Checks all timing constraints in large synchronous digital systems, taking into account:
  - Component timing properties
    - Minimum and maximum propagation delays
    - Set-up and hold constraints
    - Minimum pulse width constraints
  - Minimum and maximum interconnection delays
    - User-specified limits
    - Values calculated based on routing, capacitance, and transmission line characteristics
  - Additional designer-specified constraints

TIMING VERIFIER – SIGNAL VALUES

Value	Meaning
0	False
1	True
S	Stable
C	Changing
R	Rising edge
F	Falling edge
U	Undefined (Initial value)

Experience in Using Timing Verifier

- Provided daily feedback about timing errors as the S-1 Mark IIA design proceeded
- Meeting both minimum and maximum delays required a significant amount of work
  - Typically two or three timing errors are introduced in a given day of design work
    - With constant feedback, designers learned to make fewer timing errors
      - During initial part of design, many errors would be made during a day's work
    - A number of circuits had to be entirely redesigned to meet worst-case timing constraints
- To verify a section of logic consisting of 6357 chips
  - Required 12 minutes of CPU time
    - Executed on S-1 Mark I processor
      - Comparable in performance to 370/168
  - Required 6 Megabytes of memory

Conclusions

- The Timing Verifier allowed constant feedback to the designer with very little cost by the designer
- Use of the Timing Verifier encouraged conventions which greatly improved design readability
- The system resulted in a significant reduction in design time
  - When designing a new section, existing signals can be look up in a summary listing to see when they are changing
  - Timing errors are found early in the design, before they have a chance to propagate
  - A significant amount of time was saved by not needing to do as many hand calculations while doing the design
- The system allowed a design to be done which executes faster
  - By providing quick feedback about timing, the design could be optimized for execution speed more readily

SCALD Logic Simulator

SCALD Logic Simulator

- General purpose logic simulator
- Driven by SCALD logic drawings directly
  - One data base used to construct, simulate, and timing verify design
    - Eliminates possible transcription errors
  - Makes it easy to do simulation
    - No large input file to generate by hand

Interactive Display-Oriented System

- Signals of interest are displayed on CRT
  - As the simulation is stepped along, values of displayed signals are continuously updated
  - Locations in memory arrays can also be displayed
- Can examine and deposit in any signal value, register, or memory location
- Has built-in loaders for micro-code and simulated main memory
  - Loads micro-code assembled by the SCALD micro-code assembler

Circuit Modeling

- Uses maximum delay on circuit elements
  - Timing verifier is used to do worst-case timing analysis
    - Greatly simplifies simulator not to have to worry about timing analysis
- Uses two-value system to model circuit
- Event driven circuit evaluation scheme
  - Each event represents a bus of from 1 to N bits
    - Can simulate a 36-bit bus as fast as a 1-bit bus
- Bus symmetries used to reduce memory requirements needed to represent circuit
  - Gives order-of-magnitude reduction in memory required
- High-level primitives greatly improve simulation speed
- Block compilation can result in 10-25$\times$ performance increases

Chip Definitions

- Given in SCALD Hardware Description Language in terms of primitive functions built into simulator
  - Primitives can operate on arbitrary width buses
- Current simulator has 31 different primitives definitions built-in
  - Different sizes of AND, OR, and XOR Gates
  - Multiplexers (2, 4, 8, and 16-input types)
  - N-word Memory Elements
  - Adders, ALU's, Lookahead Units, Comparators, etc.
  - Registers and Latches

Conclusions

- Taking advantage of bus symmetries can reduce memory requirements and execution times by an order-of-magnitude
- Separating timing verification from logic verification simplifies and speeds up both tasks
- Debugging with logic simulation seems to be at least twice as fast as debugging the hardware without simulation
- Mark IIA hardware that was simulated worked essentially first time
  - Unfortunately, we did not simulate all of the Mark IIA
- Direct code generation and "block" compilation of design can improve performance by 10-25$\times$
  - Results in same performance as expensive hardware logic simulators

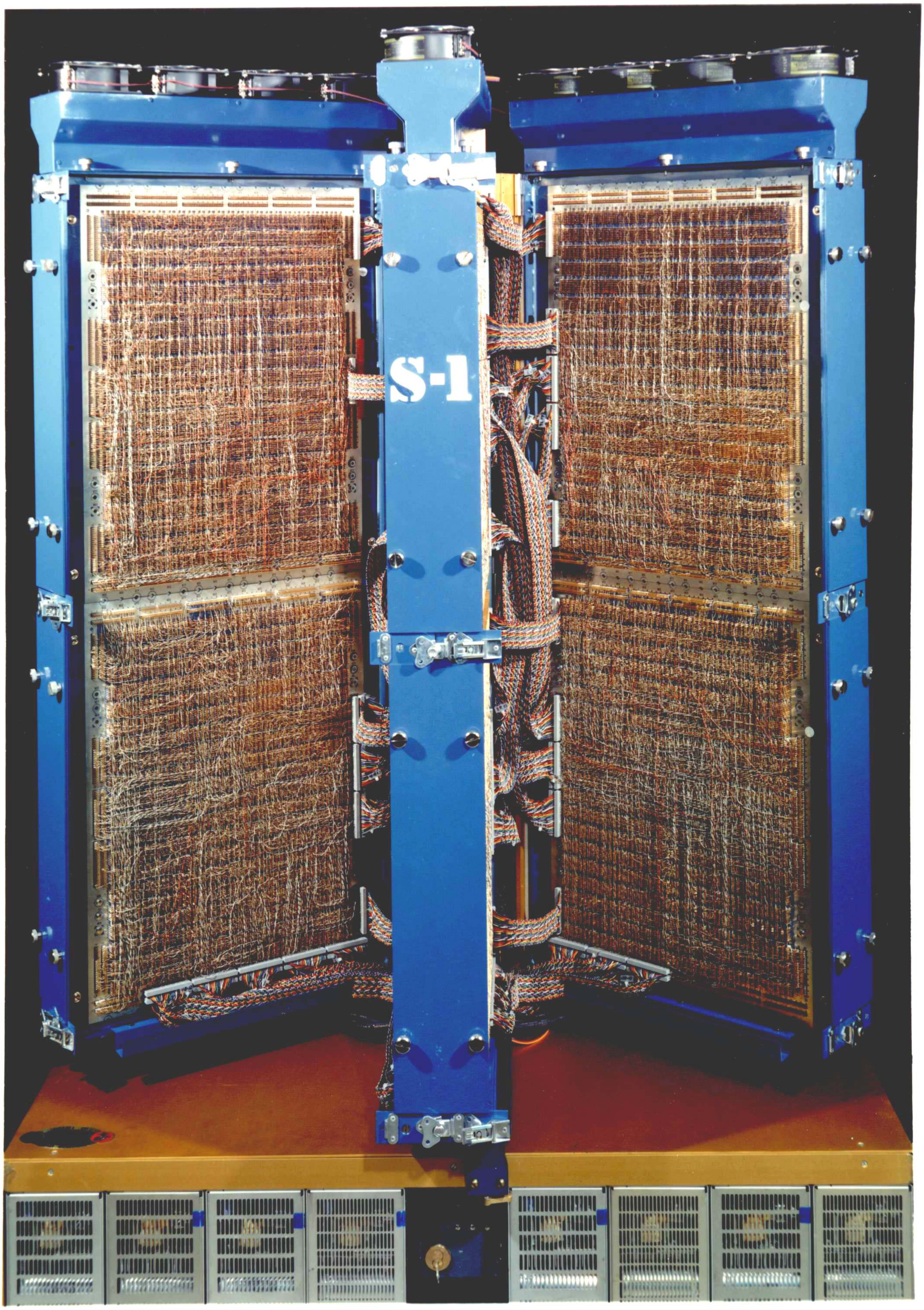
SCALD Microcode Debugger

What can MD do?

- Set and show CPU scan-logic ("vision") registers
- Enable and disable CPU clocks
- Load and verify microcode
- Log CPU parity and memory ECC errors
- Load small bootstrap programs

MD is friendlier than an oscilloscope

- MD knows by name all the scan-logic registers, most microstores, and many signals within the CPU
- MD displays the values of these on a terminal, updating them as the user single-steps the CPU
- The user can set breakpoints to occur when arithmetic expressions involving these values change from true to false or vice versa


S-1

MD is versatile

- Information about the scan-logic is not frozen
  - MD reads a file generated by the SCALD layout program which describes the scan-logic
  - Most changes in the hardware (e.g. ECOs) are reflected in MD without requiring reprogramming
- The user can customize MD to fit the task at hand with:
  - Command files
  - Loops and conditional commands
  - Display-formatting files

Mark IIA Architecture and Implementation

Mike Farmwald

General Characteristics

- Memory
    - 36 bit words, uniformly addressable as quarter-, half-, single-, or double-words
    - 16 gigabyte physical address space reduces swapping
    - 2 gigabyte uniformly addressed virtual address space makes programming easier
- Registers
    - 32 general purpose 36 bit registers
    - 16 sets of such registers for fast context switching
    - Separate status registers
        - Processor-status for operating system interests
        - User-status for user's interests (e.g. rounding mode)
- Segments of varying size
    - Promote sharing of code
    - Increase reliability due to intra-program bounds checks
- Hardware security mechanisms
    - Provide four "rings" (concentric levels of privilege)
    - Support simpler user-space/exec-space systems as well
    - Validate arguments as well as calls themselves

Data Types

- Boolean (9, 18, 36 and 72-bit)
- Integer (9, 18, 36 and 72-bit)
- Floating-point
  - Two's complement with hidden bit
    - 18-bit (1-bit sign, 5-bit exponent, 12-bit fraction)
    - 36-bit (1-bit sign, 5-bit exponent, 26-bit fraction)
    - 72-bit (1-bit sign, 15-bit exponent, 56-bit fraction)
  - Complete set of rounding modes
  - Special symbols (e.g., infinity, not-a-number)
- Complex
  - Pairs of integer or floating-point
- Vectors
  - Floating-point, integer, complex or boolean
- Matrices
  - Integer or floating-point

High Level Instructions

- Quick-sort inner loop
- Matrix operations
  - Transposition, multiply
- Elementary functions
  - Sin, log, sqrt
- Signal processing
  - FFT, filtering, convolution

Vector Operations

- Optimal match of memory bandwidth with pipeline speed
  - Through parallel use of Adder and Multiplier
- Complete set of operations for efficient coding
- Floating-point, integer, complex, and Boolean operations supported
- Mark IIA does not have invisible chaining as Cray-1 does
  - Implies user must explicitly chain, but
  - User benefits from chaining without fine-tuning the code as is necessary on the Cray-1
  - Can retain intermediate result at high precision internally
- Step-size of 1 element for all vector computations
  - Generalized transpose for sparse vectors

Typical Vector Functions

- Floating-point square root
- Complex magnitude
- Lengthwise minimum and maximum
- Floating-point to integer conversion
- 3-dimensional distance calculation
- Bit-vector shift
- Integer 2nd order recursive filter
- Vector $X(i) = S * [Y(i) + Z(i)]$

Symbolic Processing Support

- 23 pointer types reserved for user-specified data structures
  - Dispatch on pointer type
  - User-specified argument pointer type-checking
- Rich set of addressing operations well-suited for accessing symbolic types of data structures
- Multiple precision arithmetic intensively supported
- Mark IIA uniprocessor estimated rate of 5-20K logical inferences/second

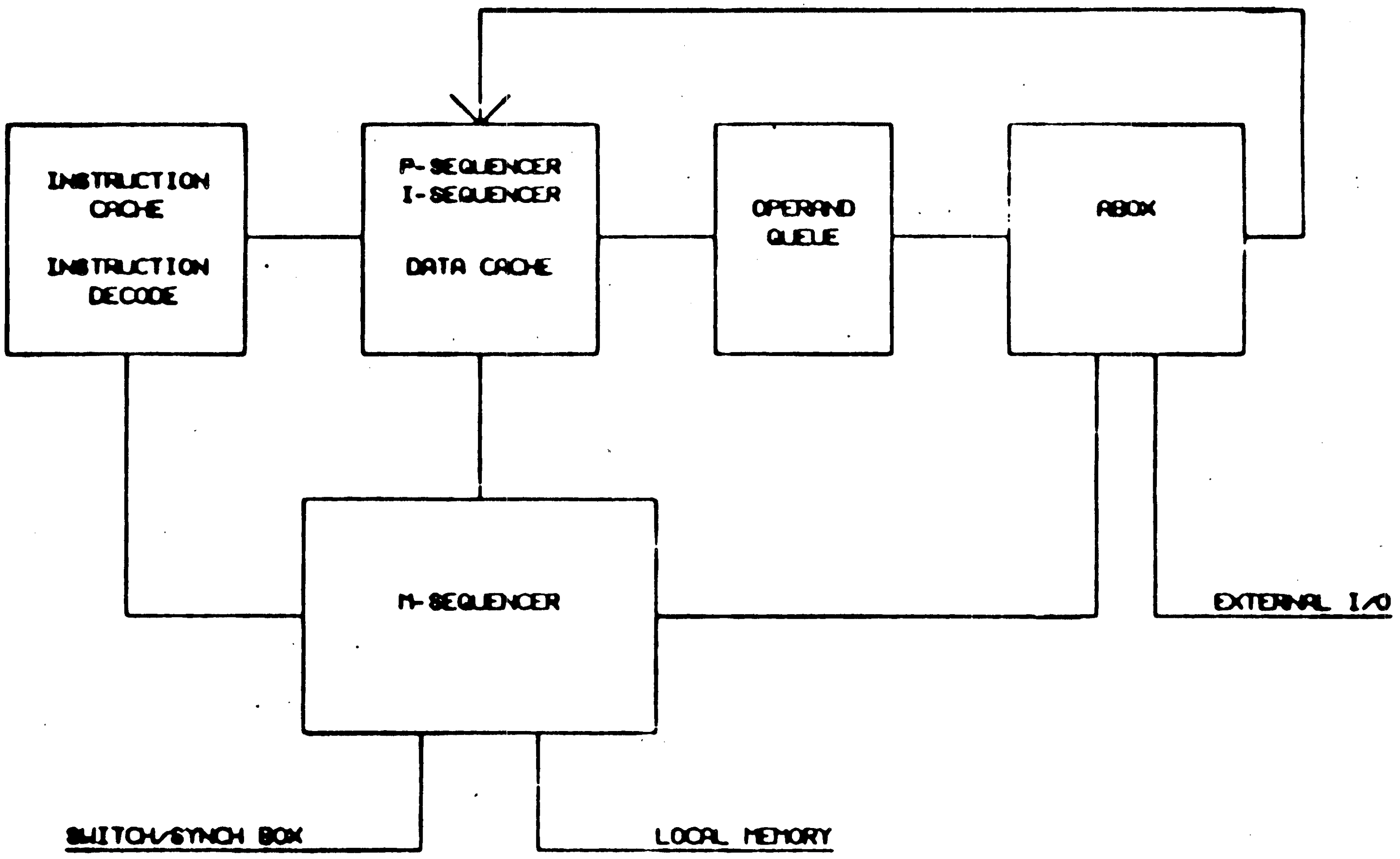
Interrupts and Input/Output

- High performance
  - Vectored interrupts, individually enabled and disabled
  - 32 priority levels for interrupts and processor itself
- Adaptable
  - Many I/O channels
  - One peripheral processor per channel
  - I/O channels are microcoded, can accommodate advancing peripheral processor technology
  - Peripheral processor and S-1 procesor
    - Synchronize via interrupts
    - Exchange control and data through shared memory
    - Can map I/O memory into a user's space to improve performance or to debug independently of the kernel
  - I/O instructions translate data to interface an 8-bit world with a 36-bit machine

Mark IIA Processor Implementation Overview

- Major sections
  - IBOX - Instruction and operand preparation unit
  - ABOX - Arithmetic and vector processing unit
- Both units are pipelined
  - 40 ns cycle time
  - Maximum instruction issue rate of one three-word instruction every other cycle
  - Maximum computation pipeline rate of one calculation per cycle per execution unit
  - Maximum data throughput rate - 450 million bytes/sec
  - Maximum calculation rate - 250 million floating operations/sec
- Both units are heavily microcode-controlled
  - Total of 2.7 million control store bits
  - Total micro-word width of more than 1400 bits

Mark IIA Processor Functional Overview



Pipelining of Instruction Preparation/Execution

- Used to exploit parallelism in sequential instruction stream
- Instruction execution is like making cars
- Multiple instructions (cars) in pipeline (assembly line) at one time
- Instruction passes through these typical stops in pipeline
    - Fetch instruction byte(s)
    - Decode instruction and operand descriptors
    - Calculate operand addresses
    - Read source operands
    - Execute instruction
    - Store result operands
- What is limit of pipelining in speeding up instruction processing?
    - Need for previously computed result
        - Indexing
        - Source operands
    - Conditional data-dependent branches

Value Prediction

- Indexing off of recently computed values causes pipeline interlock
    - Instruction unit is forced to wait for result from execution unit
- Pre-computing index values makes them instantly available and avoids interlock
- Easy to predict simple instructions
    - Move from cache, constant, or register to register
    - Increment/decrement loop index and test
    - Add/subtract small constant from array index
- Covers most commonly occurring indexing cases in compiled code
- Always predicts correctly

Branch Prediction

- Branch prediction
  - Predicting the outcome of a conditional branch before it is executed
- Use opcode
  - Always, never, normally, rarely branches
- Look at dynamic history for instruction from given location
  - What did instruction do last time it was executed?
    - Assume history repeats itself, e.g., loops
- A simple scheme
  - Decode RAM gives initial prediction for each opcode
  - Store extra bit with each word in cache
    - Says to use opposite strategy as given by decode RAM
  - This scheme works for 98% of instructions executed for a Pascal compilation on the S-1 Mark I processor

Pre-Decoding Instructions in Instruction Cache

- Allows for a shorter instruction pipeline
- Supports faster branching
  - S-1 Mark IIA can execute branch instructions in one cycle
    - Pre-computed and stored in cache
      - Length of instruction
      - Branch offset if branch instruction
      - Branch prediction bit
      - Starting address for microcode which controls operand address calculations

Add Functional Unit

- Four cycle latency
- Fully pipelined — a new result generated every cycle
- All precisions of integer or floating point add and subtract
- Simultaneous floating point add/subtract operation
- Half-word complex add and subtract
- Byte, boolean, shifts, rotaes, bit count, bit first, etc.

Multiplier Functional Unit

- Six cycle latency
- Half-word complex multiplication every cycle
- Single-word multiplication every cycle
- Double-word multiplication every two cycles
- Single-word reciprocation or square root every cycle

Elementary Functions by Taylor Series

- Fully exploits Multiplier hardware features, e.g., pipelining
- Produces results of full architectural precision
- Table look-up in large, very fast RAMs for starting values
- Piecewise quadratic approximation to popular elementary functions
  - Same speeds as multiplication (1 cycle) for
    - Reciprocation
    - Square root
  - Twice multiplier latency (2 cycles) for
    - Sine
    - Cosine
    - Arctangent
    - Exponential
    - Logarithm
    - Error function

S-1 Mark IIA Performance

Execution times, expressed in cycles, are:
Vector pipeline time/Scalar pipeline time/Total execution latency

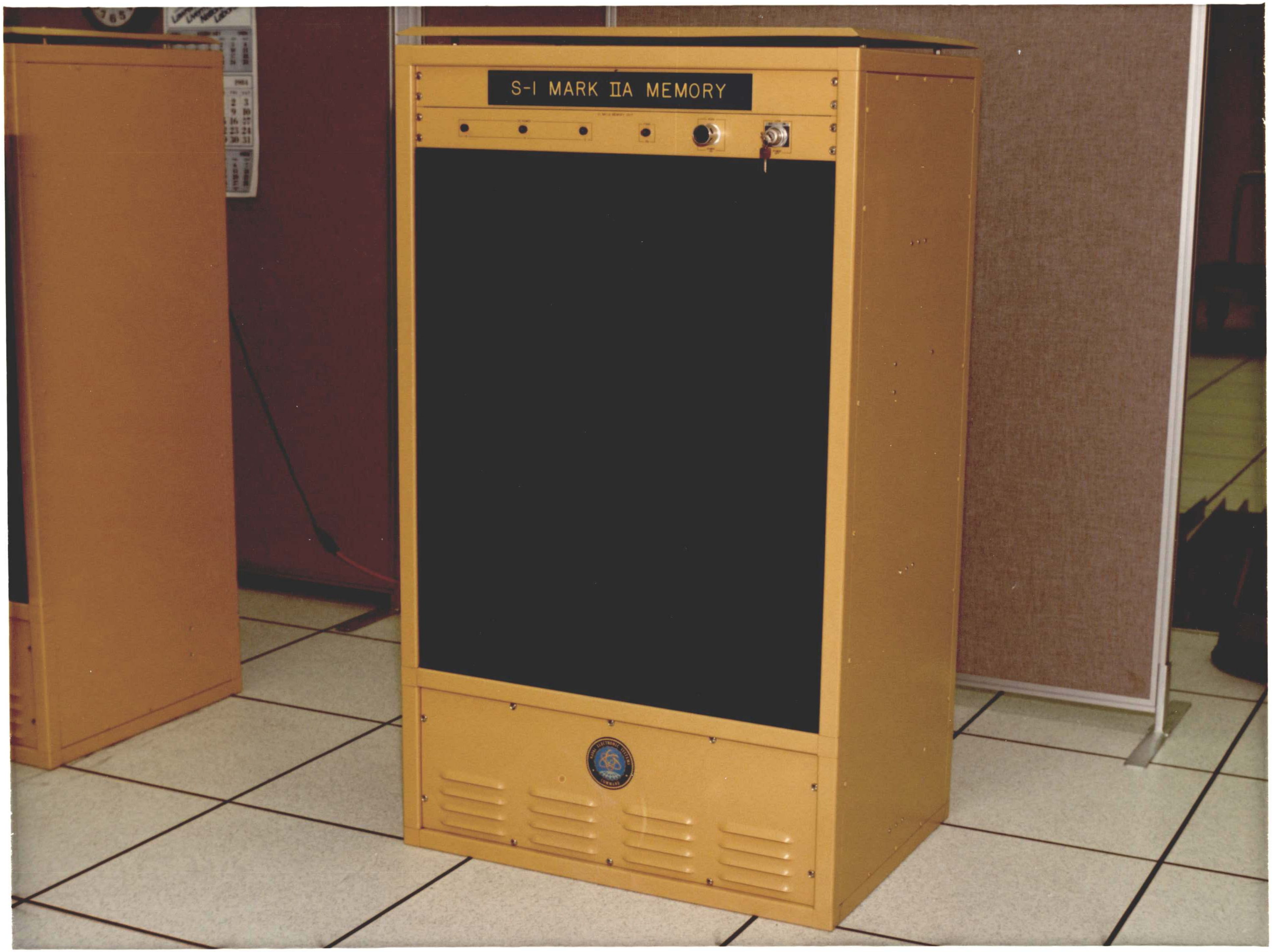
	Single Precision (36 bits)	Double Precision (72bits)
Move (register to memory)	NA/2/2	NA/2/2
Move (memory to register)	1/4/4	2/4/4
Integer add	1/2/4	2/2/4
Integer multiply	1/2/6	2/2/6
Shift	1/2/4	2/2/4
Load or deposit byte	1/2/4	2/2/4
Floating point add	1/2/4	2/2/4
Floating point multiply	1/2/6	2/2/6
Floating point reciprocate	1/2/6	3/14/18
Floating point square root	1/2/6	3/14/18
Floating point divide	2/8/12	4/14/18
Floating point logarithm	1/8/10	14/46/50
Floating point exponential	1/4/8	10/34/38
Floating point sine or cosine	2/16/20	14/38/42
Floating point arctangent	3/18/22	22/52/56

Lessons Learned in Mark IIA Development

- Packaging is as important as architecture
- Simulate absolutely everything before building anything
- Implementation must be 100% testable
- Design must be readily understandable
- Automate everything possible
- Verify architectural complexities are cost-efficient

Operational experience with S-1 Mark IIA

- We made, in retrospect, some poor implementation choices
  - Serials 1 and 2 use 72 wire-wrap boards
  - 2500 interboard cables
  - Air cooled
  - Design was not 100% scan-testable
- This led to poor reliability and much longer (than expected) debugging times
- We have learned from our mistakes


S-I MARK IIA MEMORY

Software Overview

Jeff Broughton

S-1 Software Development

- Rationale for work
  - Support test and evaluation by DoD, DoE
  - Support development of design tools and future hardware generations
- Requirements
  - Permit transport/development of high-level language programs
  - Provide timesharing services
  - Facilitate effective utilization of multiprocessor systems
- Major areas of work
  - Programming languages
  - Multiuser Operating System - Unix
  - Advanced Operating System - Amber

Implicit Goals in S-1 Software Efforts

- Sharing
  - Adhere to standards
  - Promote use of library routines
  - Capture existing software
- Portability
  - Use high-level languages
  - Use machine-independent programming techniques
- Productivity
  - Presume people to be most expensive element
  - Provide tools to automate chores
  - Exploit excess capacity
- Durability
  - Plan for future developments

Programming Languages Supported

- Pascal
  - Special extensions for systems programming
  - Separate compilation for modular decomposition
  - Exception handling
- FORTRAN
  - FORTRAN-77 dialect
  - LRLTRAN compatibility option
  - FORTRAN-8$\times$ vector extensions (in development)
  - Vectorization by preprocessor (in development)
- LISP
  - Common LISP dialect (new DoD standard)
  - Extensions for S-1 features and multiprocessing
- "C"
  - Supports capture of Unix tools

Pastel

- Improved type definition
    - Parametric types
    - Explicit packing and allocation control
    - Additional parameter passing modes
- Additional control constructs
    - Set iteration
    - Loop-exit form
    - Return statement
- Module definition
- Exception handling
- General enhancements
    - Conditional boolean operations
    - Constant expressions
    - Variable initialization

Algebraic Language Implementation

- High performance implementations facilitate use
    - Instruction set tailored for HOLs
    - Special hardware support for peculiar language features

- Standard optimizations routinely done
    - Common subexpression elimination
    - Code motion
    - Inline procedure expansion
    - Register allocation

- Certain optimizations are of special importance for S-1 systems
    - Minimization of pipeline interlocks
    - Vectorization
    - Loop blocking

LISP

- LISP Dialect
  - Upward compatible with "Common LISP"
  - Extensions to access S-1 features
- Implementation
  - Interpreter, compiler and runtime written in LISP
  - Efficient execution of numerical programs
  - Special architectural support exploited
- Possible Applications
  - Macsyma
  - Artificial intelligence
  - Program development
- Unique capabilities
  - Memory/computation intensive applications
  - Mixed symbolic/scientific applications
  - Multiprocessor applications

Multiuser Operating System - Unix

- Provides prompt multi-user access to Mark IIA uniprocessors

- Simple uniprocessor timesharing executive
    - Originally developed at Bell Laboratories for PDP-11s
    - Transported to many architectures
    - Unspecialized
        - Does not exploit full capabilities of S-1 systems

- Allows immediate capture of DoD investment in Unix tool developments

Advanced Operating System - Amber

- Full functionality uni- and multi-processor executive
- Support a mix of applications in a modular fashion
    - Real-time systems (e.g., signal processing)
    - Compute-bound problems (e.g., theater weather forecasting)
    - Interactive use (e.g., program development)
- Support full use of S-1 architectural features
    - Large memory space
    - Multiple processors
    - Hardware redundancy
- Support timely test and evaluation of S-1 systems
    - Program development environment
    - Classified/unclassified ARPANET access
    - Extensibility to meet changing needs

AMBER SECURITY FEATURES

- Access to all objects may be controlled
    - Files, tasks, IO devices are all protected uniformly
    - Different operations controlled by different modes
        - E.g. read/write/execute for files
- Discretionary access control
    - Any individual may grant access to any other user or group
    - "User" may mean person, program or task
- Procedural access control
    - Limits object access to protected server task
    - Allows implementation of complex protection policies
- Nondiscretionary access control
    - Provides multilevel security partitioning
    - Implementation being explored for later version of system

Amber Storage System Features

- Combines functions of file system and virtual memory
- Hierarchical directory structure
  - Tree structure helps user organize information
  - Long, mnemonic file names aids documentation
  - Property lists store history information
- Files are represented as segments
  - Segments may hold data, programs, text files, etc.
  - Segments are mapped into the virtual memory and referenced as normal program data
  - Shared segments provide simple, high-bandwidth communication between different processes

Demand Paging

- Paging is invisible to the user
- Pages are copied directly between disk records and main memory
  - They are copied in as a response to page faults
  - They are removed by a kernel daemon task
    - Page replacement works globally on all of main memory
    - Approximation of least-recently-used algorithm is used for eviction
- This is not optimal for all applications
  - Real-time response can be degraded by page faults
  - Least-recently-used is not always a good policy
  - Transaction processing requires assurance that updates have been completed
- Solutions
  - User may temporarily "wire" pages into main memory
  - User may give "hints" about their reference patterns
  - User may request explicit updating to disk

Amber Multitasking Support

- Multilevel scheduling
  - Low level provides simple real-time mechanism
    - Priority scheduling with round-robin queues
    - Dedicated processor assignments
    - Interrupt dispatching
  - High level may implement complex policies
    - Resource allocation
    - Load leveling on multiprocessor configurations
- Communication techniques
  - Shared memory between tasks for direct communication
    - Ada-style sharing of entire address space
    - Added protection of sharing single segments
  - Message channels for "network" style data transmission
- Synchronization techniques
  - Software interrupts
  - Event notification
- Clock services
  - Real- and CPU-time interrupts
  - Time-outs on all event waits

Amber Reliability Features

- Dynamic reconfiguration of multiprocessor
  - Able to operate with a portion of hardware configuration
  - Able to change configuration while system operational
  - Exploits hardware redundancy

- Transaction processing in Storage System
  - Maintains consistency in face of system failure
  - Insures data integrity
  - Provides for system restart without time consuming salvage

- Kernel design philosophy
  - Modular structure, without hidden dependencies
  - Strict locking hierarchy to avoid deadlocks
  - Consistency checks of internal data bases
  - Timeouts on all waits
  - Extensive metering and logging
    - Performance measurement
    - Diagnosis of unusual conditions

Amber Program Development Environment

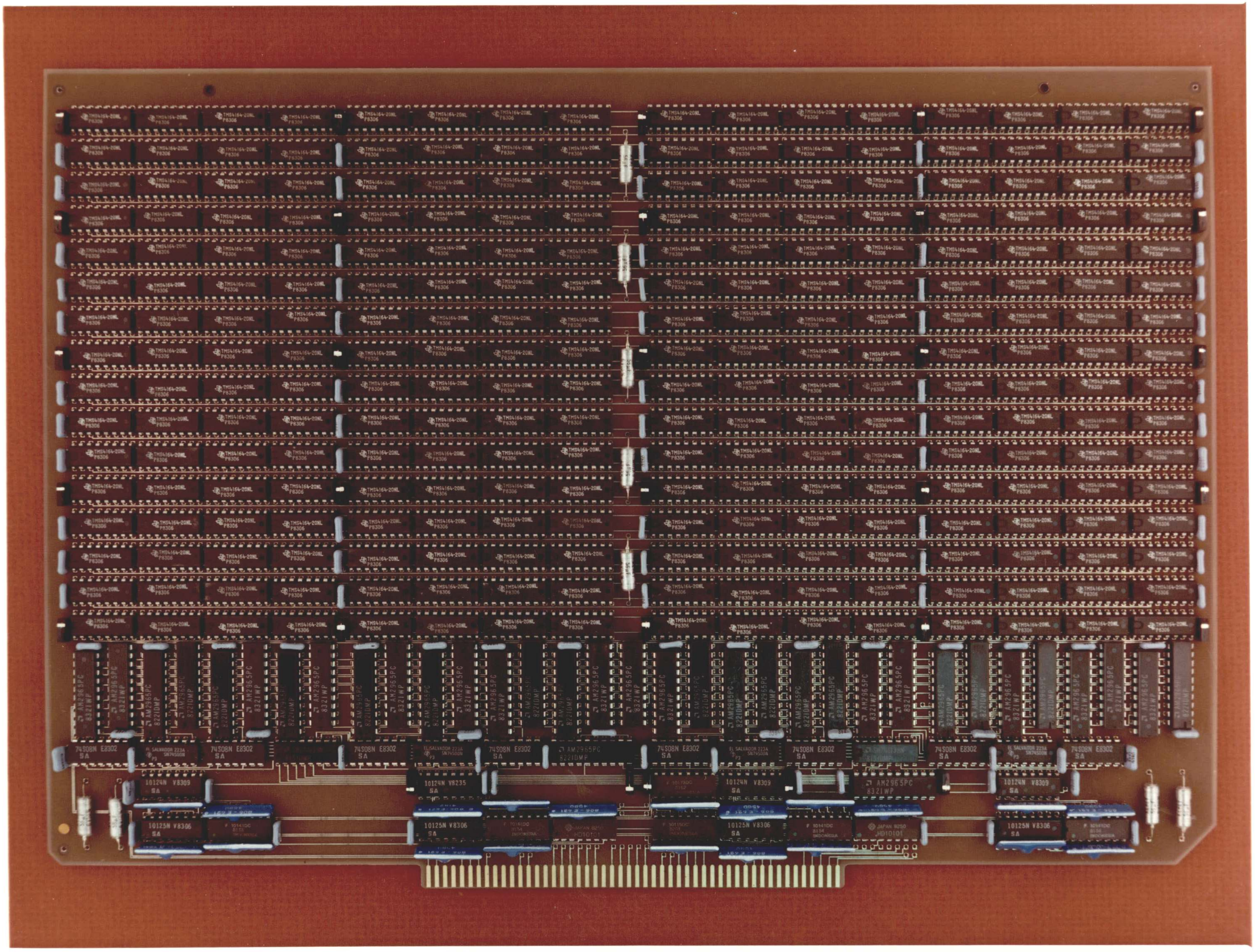
- Library packages provided
  - File management
  - Display management
  - Input line editing
  - Command processor
- Development tools
  - Pascal/FORTRAN compilers
  - Editor
  - Interactive debugger
  - Directory editor
- Unique programming services
  - Object-oriented programming
  - Garbage collection
  - Dynamic linking

Object-Oriented Programming in Amber

- Technique for writing flexible, durable software
    - General solution to the "device independence" problem
    - Runtime binding of functions to mechanism
- Message-passing approach
    - Protocols define the generic functions to be performed
    - Objects define the mechanisms they support
    - Default mechanisms define functions in terms of simpler protocols
- Some protocols defined in Amber
    - Serial I/O - for raw, 8-bit serial communications
    - Text I/O - for line at a time character input/output
    - Display I/O - for control of CRTs or windows
    - Directories - for management of catalogs
- Implementation
    - Pastel library package called from protocol modules
    - Functions are strongly-typed; objects are not

Dynamic Linking

- Linking performed on demand at runtime
    - External reference causes trap
    - Segment containing module is mapped in
    - Program is restarted with actual address
- Program sharing without multiple copies of the object code
    - Allows the development of interlocked subsystems
    - Programs automatically get updated versions of subroutines
    - Shares storage
- Aids in program development
    - Promotes modular design methodologies
    - Eliminates linking, shortening the debugging loop
    - Allows versions of a module to be changed on-the-fly
- A static linker/loader is provided
    - For use by stable subsystems
    - For use by time critical programs
    - For use in embedded applications requiring minimum support



Current Status

Jeff Broughton

S-1 Mark IIA Current Status

- Processors
  - Serial 1 machine is operational
  - Serial 2 machine runs some large programs
  - Serial 3 logic element installation beginning
  - Serial 4-6 construction commenced
- Peripheral Equipment
  - Two I/O Processors operational on each Mark IIA
  - One 32 megaword Memory Box operational on each Mark IIA
  - 1 Gigabyte disk storage installed on each Mark IIA
- Microcoding
  - Scalar architecture essentially complete
  - Operating system support complete
  - Many vector instructions complete

S-1 Mark IIA Current Status (continued)

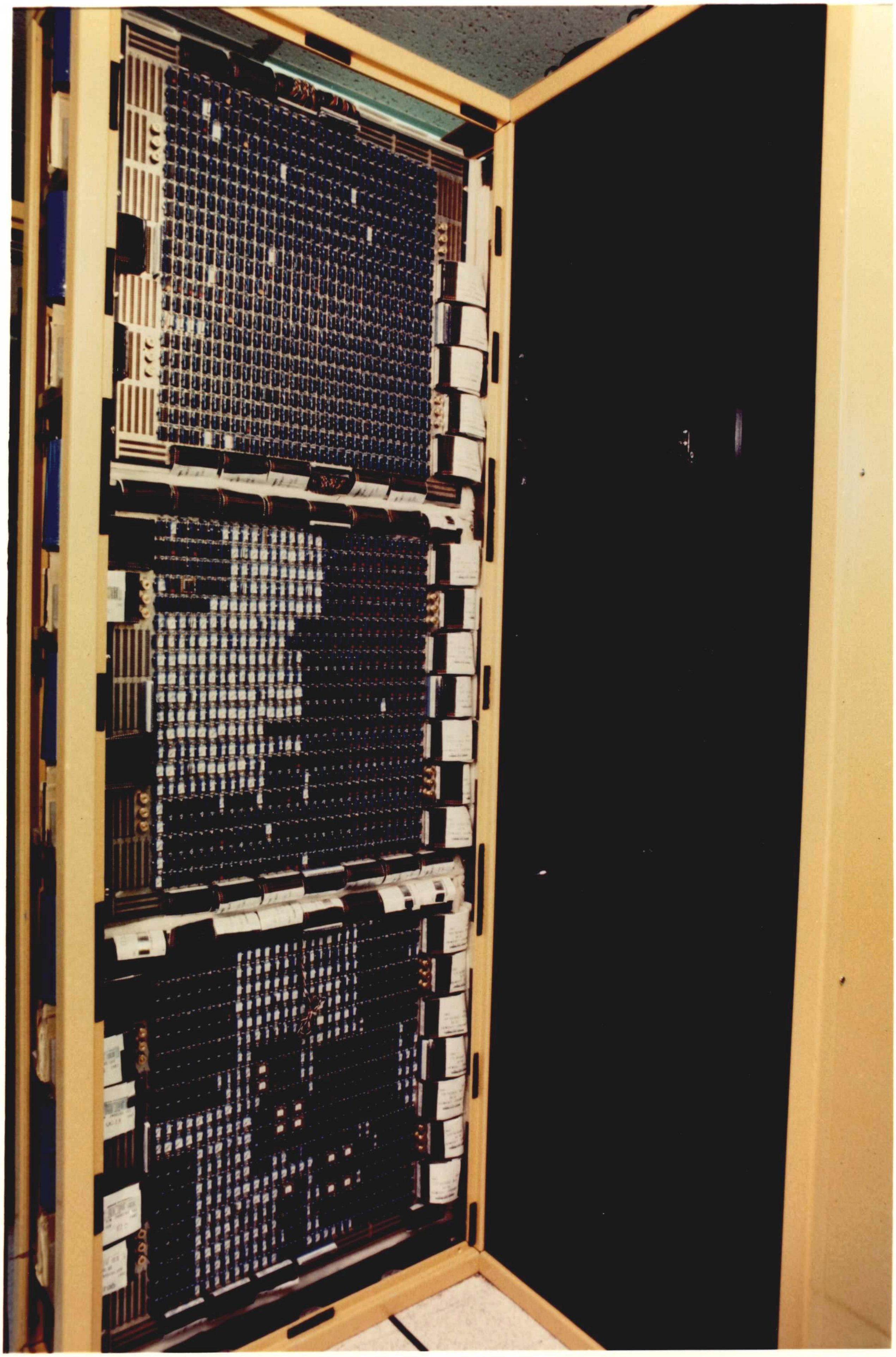
- Language software
  - Pascal and "C" programs fully supported
  - FORTRAN/LRLTRAN in "beta" test
  - FORTRAN vectorizer ready for evaluation
  - LISP system runs some programs, stand-alone
- Unix Operating System
  - Kernel operational for time-sharing
  - Compiler, editors, common utilities installed
  - Reasonably stable
- Amber Operating System
  - Kernel operational
  - Simple multi-user support
  - System still in development

S-1 Mark IIA Current Status (continued)

- Large application codes transported to Mark IIA
    - TIMI - Semiconductor physics modeling program
        - Operational
    - SUNTAN - Atomic physics modeling program
        - Operational
    - Synthetic Aperture Radar program
        - Beginning evaluation

S-1 Mark IIA Near-Term Outlook

- Commencement of Test and Evaluation under Unix
    - Immediate support for "C" applications
    - Support for Pascal and FORTRAN will follow shortly
- Completion of initial release of Amber
    - Shakedown continuing through end of spring
    - Operational installation on Serial 2 Mark IIA
    - Ongoing enhancements to Program Development Subsystem
- System available for remote use
    - Near-term terminal access via local hosts
    - Near-term file transfer via local hosts
    - Unix tape support in May
    - Direct MILNET access by fall
- Establishment of user support function
    - Coordinated through NAVELEX PMO

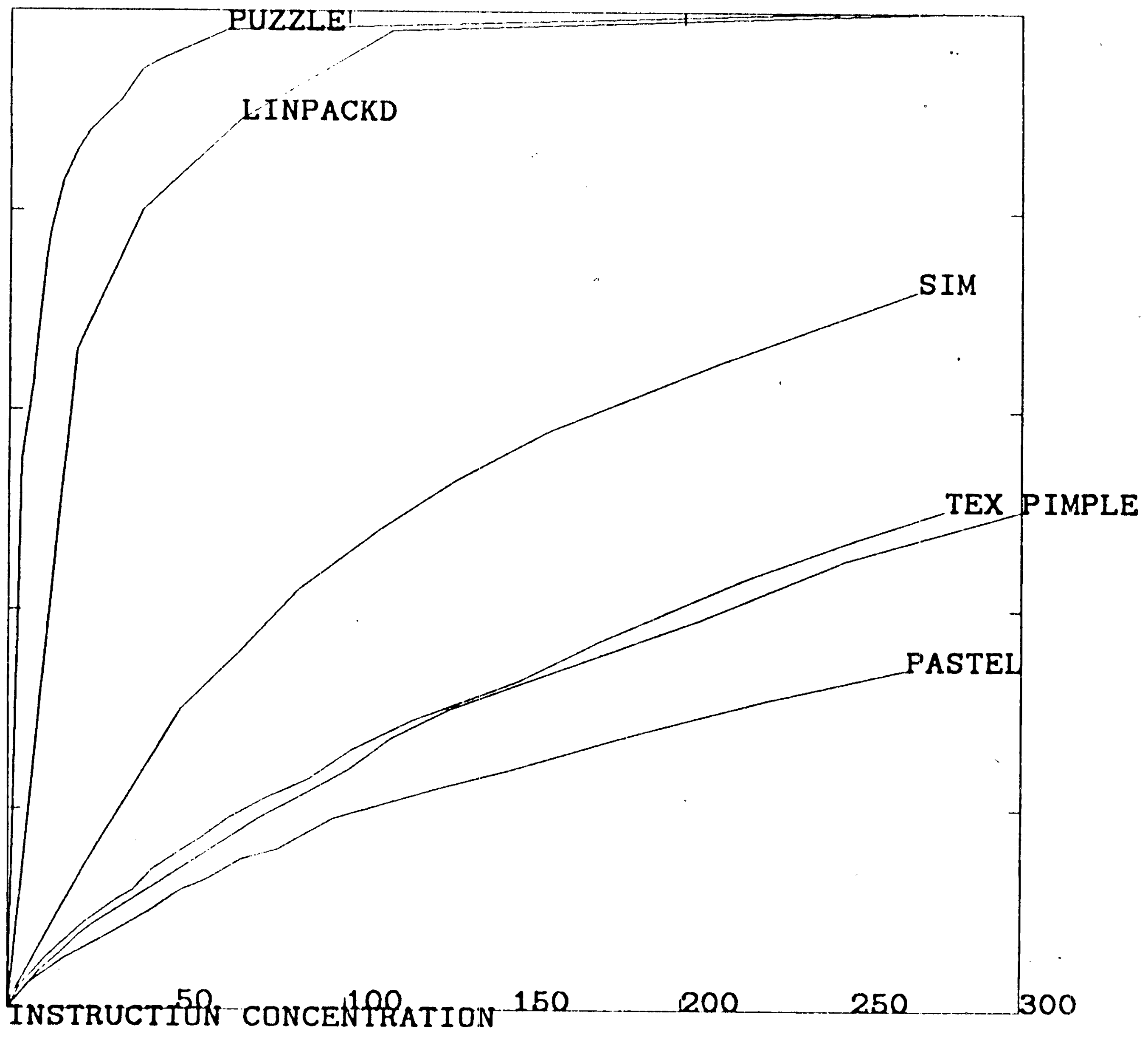


Architectural Studies

Jeff Broughton

Applicability of Statistics

- Only use statistics from real programs that are "too slow"
  - Some programs are already fast enough
    - Command processors, editors and other existing highly interactive programs
  - I/O limited programs won't benefit from instruction-set improvements
  - Toy programs and benchmarks may not be representative
- Need representative compiler
  - Good register allocation
  - Common subexpression elimination
  - Code motion



PUZZLE
LINPACKD
SIM
TEX
PIMPLE
PASTEL
50
100
150
200
250
300
INSTRUCTION CONCENTRATION

Programs Measured

- Pascal compiler
- TEX text formatter
- SCALD II logic simulator
- SCALD II micro assembler
- PIMPLE hydrodynamics code
- LINPACK Argonne National Labs linear algebra benchmark
- 2D semiconductor physics simulation

Statistics-Gathering Methodology

- Need to make measurements without high overhead
  - $\times$ 10 to $\times$ 100 for architectural simulation is incompatible with measuring programs that are "too slow"
- Combine basic block execution counts from a program run with basic block statistics from compile-time to determine overall execution statistics
  - About 25% overhead
- Optionally call subroutine for every memory reference
  - Write trace files for later processing to determine cache performance
- Can collect statistics for one architecture by running on another
- Pastel compiler used to insert measurements

Statistics as Architectural Guidelines

- Use speedup/slowdown as primary criteria
  - For example, don't omit an instruction used 1% of the time if the cost of simulating it is 50-100
- If insignificant impact on performance, leave it out, unless zero cost
- If significant performance gain possible, consider including architectural support, at least for the time being

Cache simulation

- Did not look at instruction caches
  - Expect instruction caches to perform better than data caches
- Simulation results include clearing cache every 100,000 references
- Data map cache
  - roughly 10^{-4} miss rates with reasonable design
  - page size is important
- Data cache
  - roughly 1-2% miss rates for compiler
    - Thus 5-10% slowdown for 12 cycle memory
  - roughly 10% miss rates for linear reference streams that exceed cache size
    - Thus 50% slowdown

Pastel compiler map cache miss rate

7699232 memory references

Data map cache size = 64 entries, page size = 1024 words

1 sets	.0019
2 sets	.00069
4 sets	.00047

Data map cache size = 128 entries, page size = 1024 words

1 sets	.00086
2 sets	.00043
4 sets	.00042

Data map cache size = 256 entries, page size = 1024 words

1 sets	.00066
2 sets	.00042
4 sets	.00042

Data map cache size = 512 entries, page size = 1024 words

1 sets	.00042
2 sets	.00042
4 sets	.00042

PIMPLE map cache miss rate

152657 memory references

Data map cache size = 64 entries, page size = 1024 words

1 sets	.00027
2 sets	.00027
4 sets	.00027

Data map cache size = 128 entries, page size = 1024 words

1 sets	.00027
2 sets	.00027
4 sets	.00027

Data map cache size = 256 entries, page size = 1024 words

1 sets	.00027
2 sets	.00027
4 sets	.00027

Data map cache size = 512 entries, page size = 1024 words

1 sets	.00027
2 sets	.00027
4 sets	.00027

Pastel compiler data cache miss and writeback rate

7699232 memory references

Data cache size = 4096 words

	4		8		16	
2 sets	.026	.37	.018	.39	.014	.4
4 sets	.025	.38	.016	.4	.012	.41

Data cache size = 8192 words

	4		8		16	
2 sets	.023	.39	.015	.42	.0098	.45
4 sets	.023	.39	.014	.43	.0087	.47

Data cache size = 16384 words

	4		8		16	
2 sets	.022	.4	.014	.43	.0085	.47
4 sets	.022	.4	.014	.44	.0082	.48

Data cache size = 32768 words

	4		8		16	
2 sets	.022	.4	.013	.45	.0079	.49
4 sets	.022	.41	.013	.45	.0078	.49

.19 writes

7843 different 16 word lines referenced

LINPACK Data Cache Miss and Write Back Rates

2.2 million memory reference

Data cache size = 4096 words

	4		8		16	
2 sets	0.16	.90	0.099	0.83	0.072	0.74
4 sets	0.16	.91	0.082	0.91	0.044	0.9

Data cache size = 8192 words

	4		8		16	
2 sets	0.13	0.89	0.065	0.89	0.035	0.89
4 sets	0.13	0.9	0.066	0.89	0.035	0.9

Data cache size = 16384 words

	4		8		16	
2 sets	0.63	0.88	0.034	0.87	0.019	0.87
4 sets	0.69	0.86	0.037	0.86	0.021	0.86

Data cache size = 32768 words

	4		8		16	
2 sets	0.36	0.97	0.019	0.96	0.099	0.96
4 sets	0.36	0.97	0.018	0.97	0.095	0.97

0.33 writes

1397 different 16 word lines referenced

Some Results

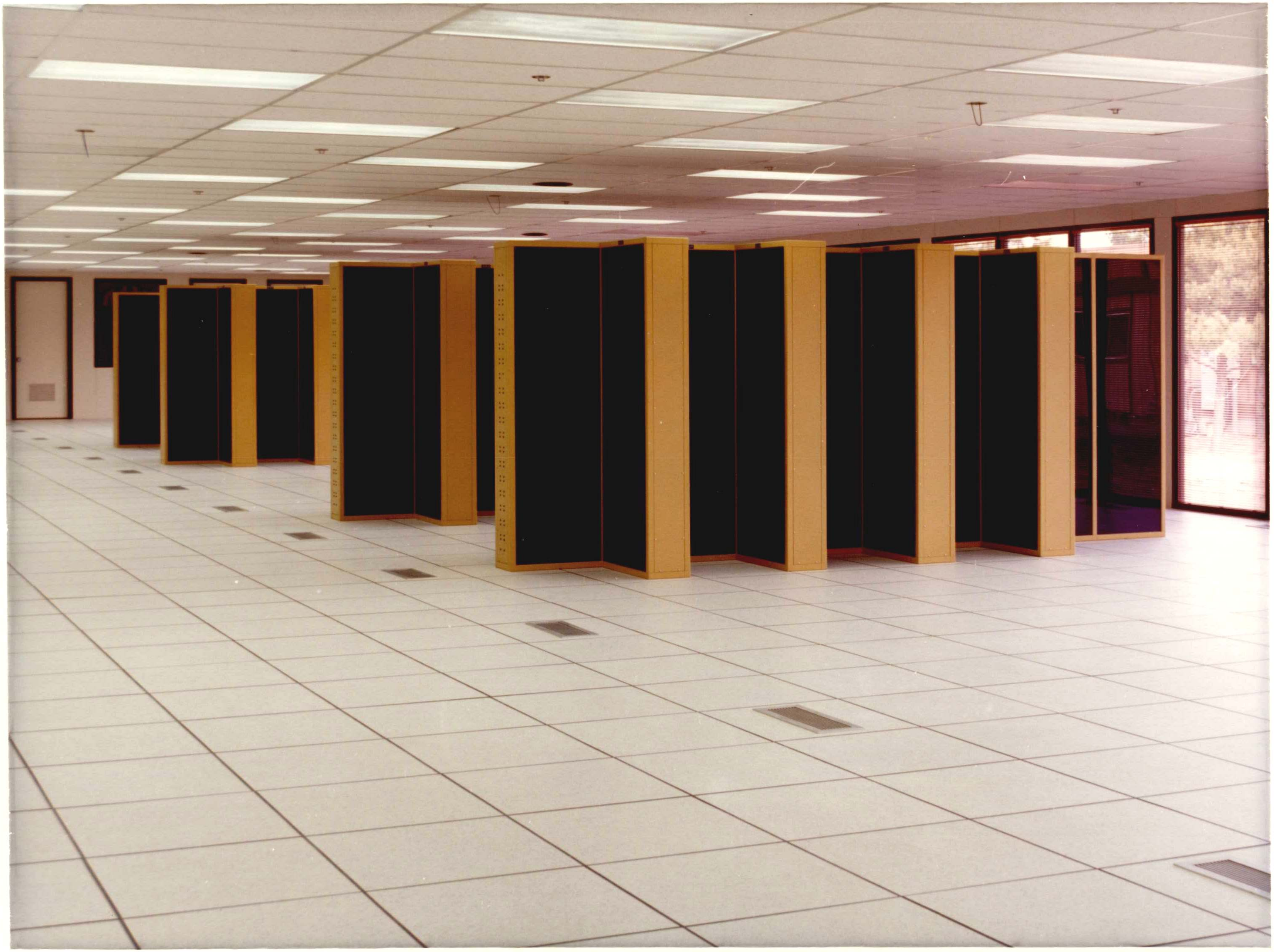
- Number crunching programs are very different from system code
  - Heavy use of indexing
  - High fraction of floating point operations
  - Large basic blocks
  - Many instructions per procedure call
- Conditional branches are important
- Indexing is important
- Procedure call cost is important for system code

Some results

	Puzzle	Pastel	TeX	Micro*	LogSim	Pimple	Linpack	2dSemi**
Conditional branches								
	43%	16-23%	17%	13%	11%	7.4%	7.4%	8.4%
Basic block size								
	2.2	3.5-4.5	3.9	5.3	6	9.5	7.9	7.8
Procedure size								
	180	35	40	34	49	140	470	300
	.56%	2.9%	2.5%	2.9%	2%	.71%	.21%	.33%
Register save/restore (6 registers max)								
	1.8%	6.3%	4.3%	4.2%	6.7%	-	-	-
Arguments per call								
	-	1.7-2.1	.81	1.3	-	4.4	-	-
Load/stores								
	26%	38%	41%	-	-	35%	45%	31%
	26%	44%	43%	56%	41%	59%	-	-
Load/store cost								
	25%	32-36%	31%	42%	35%	46%	79%	31%
Indexing								
	24%	3.5-9.5%	5.8%	6.1%	15%	29%	44%	18%
Bytes								
	0	2-5%	4.7%	8.5%	0	0	0	.037%
Flops								
	0	.003%	.026%	0	0	34%	29%	31%

Mark IIA Performance Problems

- Cycle time is too long
    - Everything takes the same time
- Average scalar performance is 1/4-1/2 peak performance
    - Statistical effects hurt
    - Branch interlocks
    - Data pipeline interlocks
    - Cache-miss overhead



AAP — Advanced Architecture Processor

Mike Farmwald

AAP Design Goals

- Same fundamental goals as Mark IIA
  - Provide high performance across many applications
    - Numerical
    - Symbolic (e.g., artificial intelligence)
  - Support modern software (e.g., virtual memory system)
  - Explore multiprocessor effectiveness
- AAP design reflects some strategy changes
  - Optimize most common functions
  - Utilize multiple processors rather than vectors for high-end numerical performance
  - Stress scalar performance
  - Tailor design to compact packaging
  - Add special functional units for high performance specialty applications

AAP Design Issues

- Simplicity
  - Minimize design/debug time (1-2 years)
  - Reduce chip-count/increase reliability
  - Reduce size/increase performance
  - Reduce cost
  - Improve manufacturability
  - Exploit semiconductor and packaging advances
- Increase functional modularity
  - Permit subsetting and supersetting
  - Support special functional units
- Near-term implementation technology
  - ECL gate-arrays and MSI components
  - High density PC cards
  - Water cooling
- Wafer-scale implementation compatibility
  - Design suitable for both PC and WSI

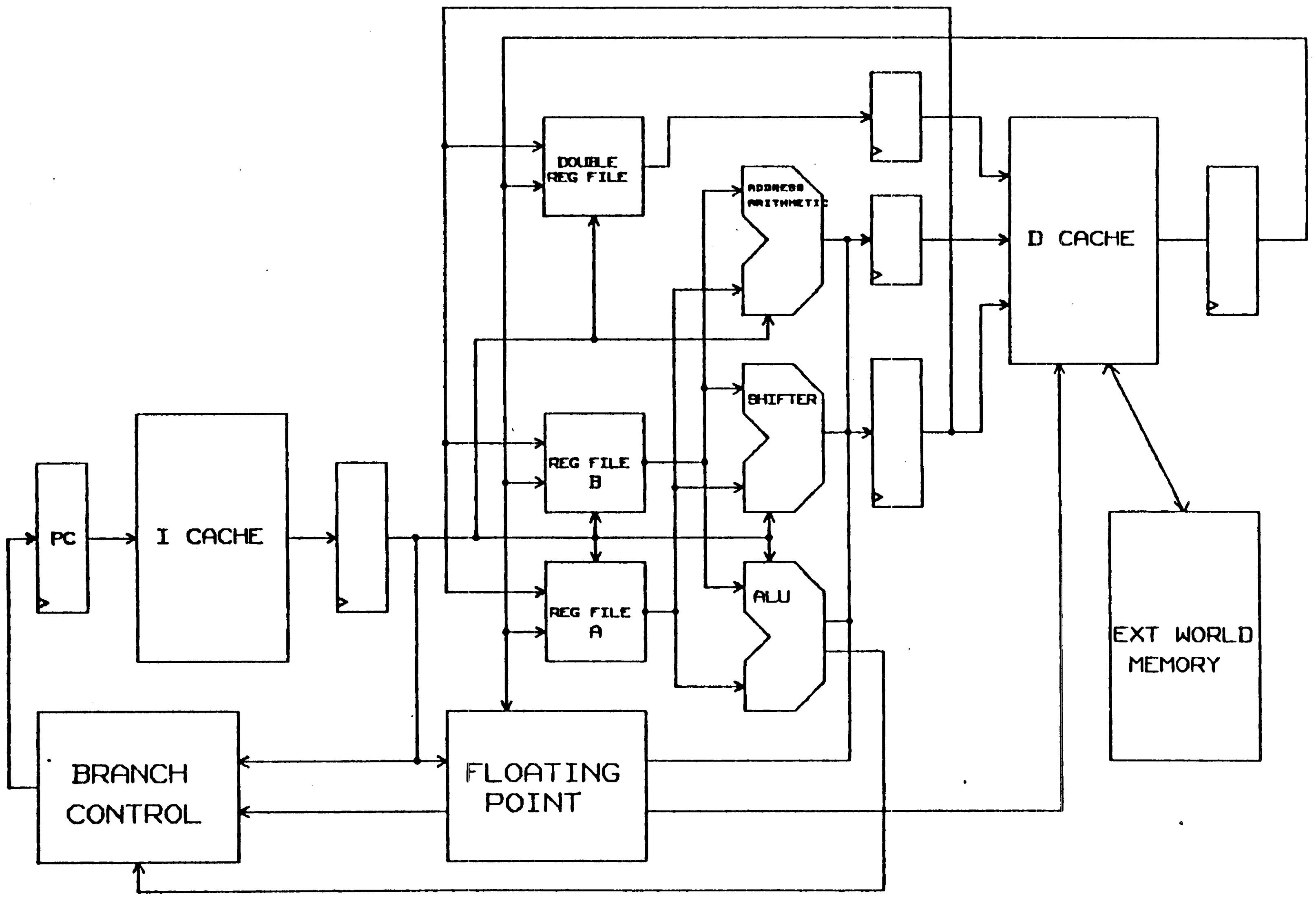
AAP Highlights

- Key design changes
  - Shorter pipeline
    - 3-stage v. 11-stage pipeline
  - Reduced cache-miss time
    - 350 ns vs. $>$ 1500 ns
  - Faster interprocessor communication
  - 100% automatic testability
- Improved components
  - 2500/3500-gate ECL gate arrays
  - Faster and denser ECL RAMs for cache/microstore
  - 256K dynamic RAMs for memory
- Smaller package
  - Single CPU is 24 inches $\times$ 24 inches $\times$ 6 inches
  - Multiprocessor fits in a single cabinet

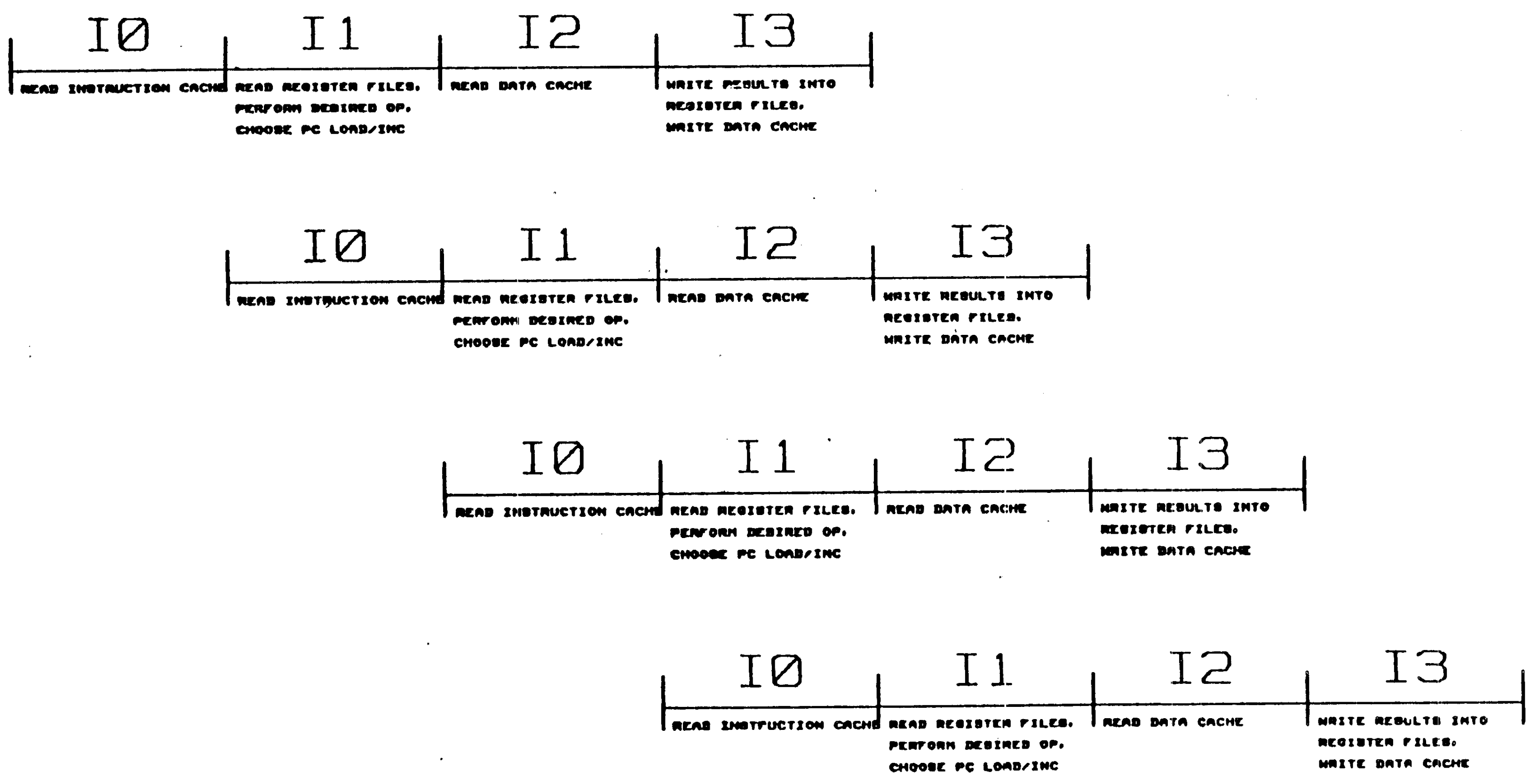
AAP Highlights (continued)

- Faster cycle time
  - 30 ns v. 80 ns for simple operations
  - Complex operations take multiple cycles
- Estimated performance
  - 5 times a Mark IIA on unstructured codes
  - 1/4 - 2 times a Mark IIA on vectorizable numeric codes
- Lower cost
  - Less than $150 K per CPU

AAP Dataflow Diagram



AAP Pipeline Diagram


I0
I1
I2
I3
READ INSTRUCTION CACHE
READ REGISTER FILES, PERFORM DESIRED OP, CHOOSE PC LOAD/INC
READ DATA CACHE
WRITE RESULTS INTO REGISTER FILES, WRITE DATA CACHE

Memory System Goals

- Highest possible density
  - Same cooling technology as processor
  - Hybrid adaptors
- Highest possible performance
  - Bandwidth foremost
    - 8 byte wide input and output busses
    - Pipelined
    - Massive parallelism: all RAMs can be cycled simultaneously
  - Latency is RAM-limited
    - Local caches and pre-fetching provide low latency

Memory Board Organization

- Four banks per board
  - Commands dispatched from input bus to appropriate memory bank
  - One DW wide write data bus shared by all bank
  - Data from all banks arbitrated onto DW wide return bus
- Memory banks handle independent, simultaneous operations
  - RAM cycle times are long compared to CPU cycle time
  - Each bank cycles one entire cache line of 336 RAMs
  - Seven hybrids per bank

RAM Hybrid Adaptors Look Like Huge ECL RAMs

- Each hybrid adaptor contains a 12-bit slice of each of the four DWs
- 52 RAMs, 2 gate arrays, many bypass caps
- All inputs are ECL, two loads each (per hybrid adaptor)
- All outputs are ECL
- ECL/TTL conversion done by gate arrays on the hybrid adaptor
- All TTL RAM signals will be confined to the hybrid adaptor

AAP Uniprocessor Performance

- Instruction issue rate
  - One instruction issued per 30 ns clock cycle
  - Majority of instructions take one cycle to execute
  - Peak performance 33 MIPS
- Cache miss fill time
  - 12 clock cycles
  - Processor continues when first word of line returned
  - Automatic pre-fetching of next cache line
- Pipeline latency
  - Load - 2 cycle
  - Floating add - 2 cycles
  - Multiply - 2 cycles
  - Read processor status - 2 cycle
- Multiple cycle instructions
  - Store byte - 2 cycles (only in special cases)
  - Divide - 18 cycles (64-bit result)
  - Kernel calls and traps - 5 cycles
  - Interrupts - 6 cycles

AAP Multiprocessor Communication

- Multiprocessor appears to have a uniform global memory
    - Each processor has a local memory
    - Processors communicate to exchange non-local data
- Communication is by message passing
    - Requests are forwarded neighbor-to-neighbor
    - Each node is a small cross-bar switch
    - Bidirectional ring is pipelined transport mechanism
    - Multiple node hops possible in a single cycle
- Cache coherence is maintained
    - Shared writes are broadcast
    - Synchronization is done by a fast distributed locking mechanism
- Same mechanism used for other purposes
    - Interprocessor messages
    - Input/output

Testability Considerations

- Design will be 100% automatically testable
  - No hidden state
  - Verified prior to construction
- Each module independently testable
  - Standalone in test rack
  - In system via "Spy Bus"

Advanced Electronic Packaging

Howard Davidson

Properties

- Accepts standard MSI and gate array packages
- Maintains low junction temperatures
- High component density
- High interconnect density
- Quiet power distribution

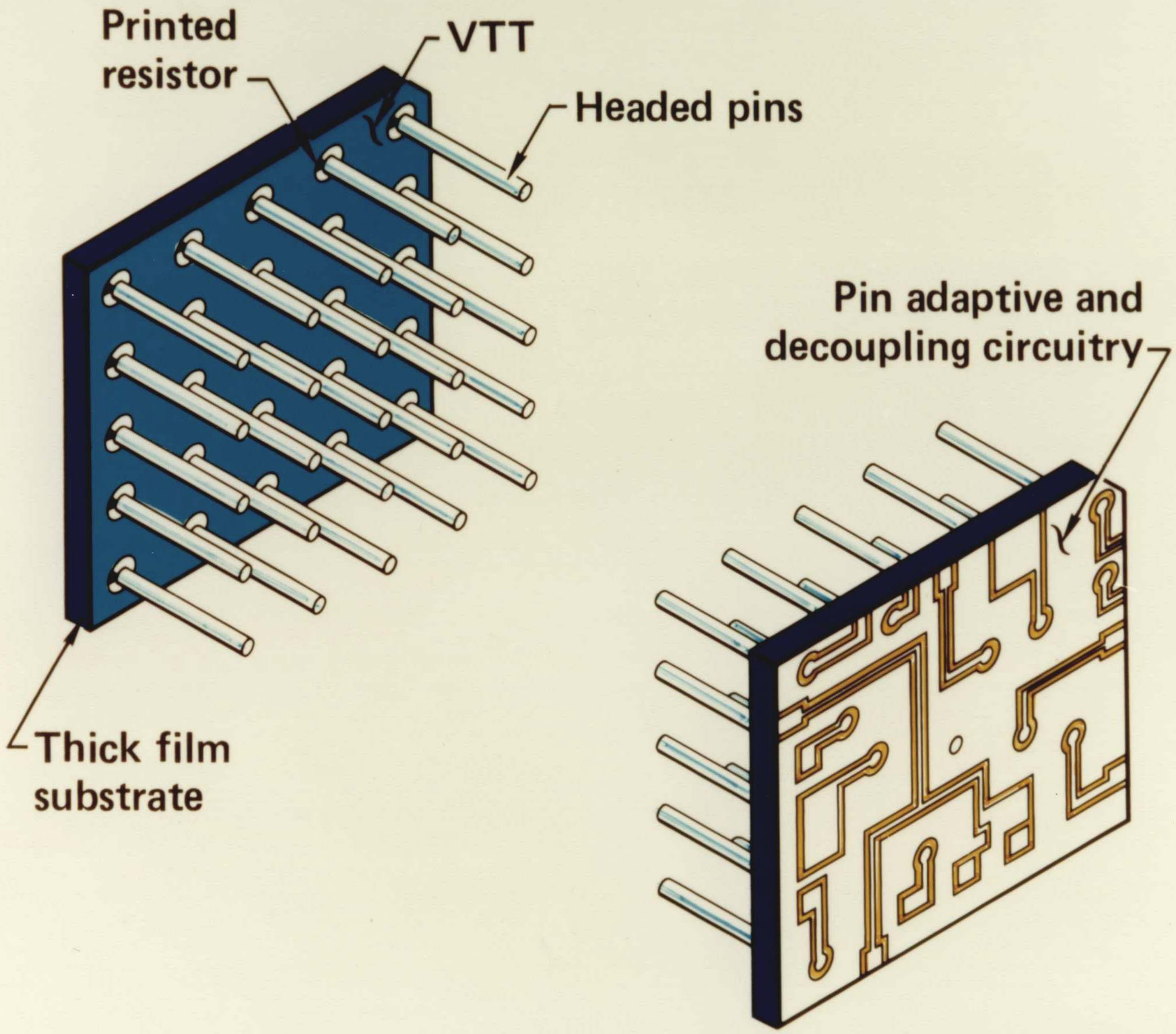
Cold Plate Characteristics

- Water cooled flat plate heat exchanger
- 0.050 inches thick, photoetched and brazed construction
- 0.4 GPM per kw water flow
- Will hold 35 C junction temperatures

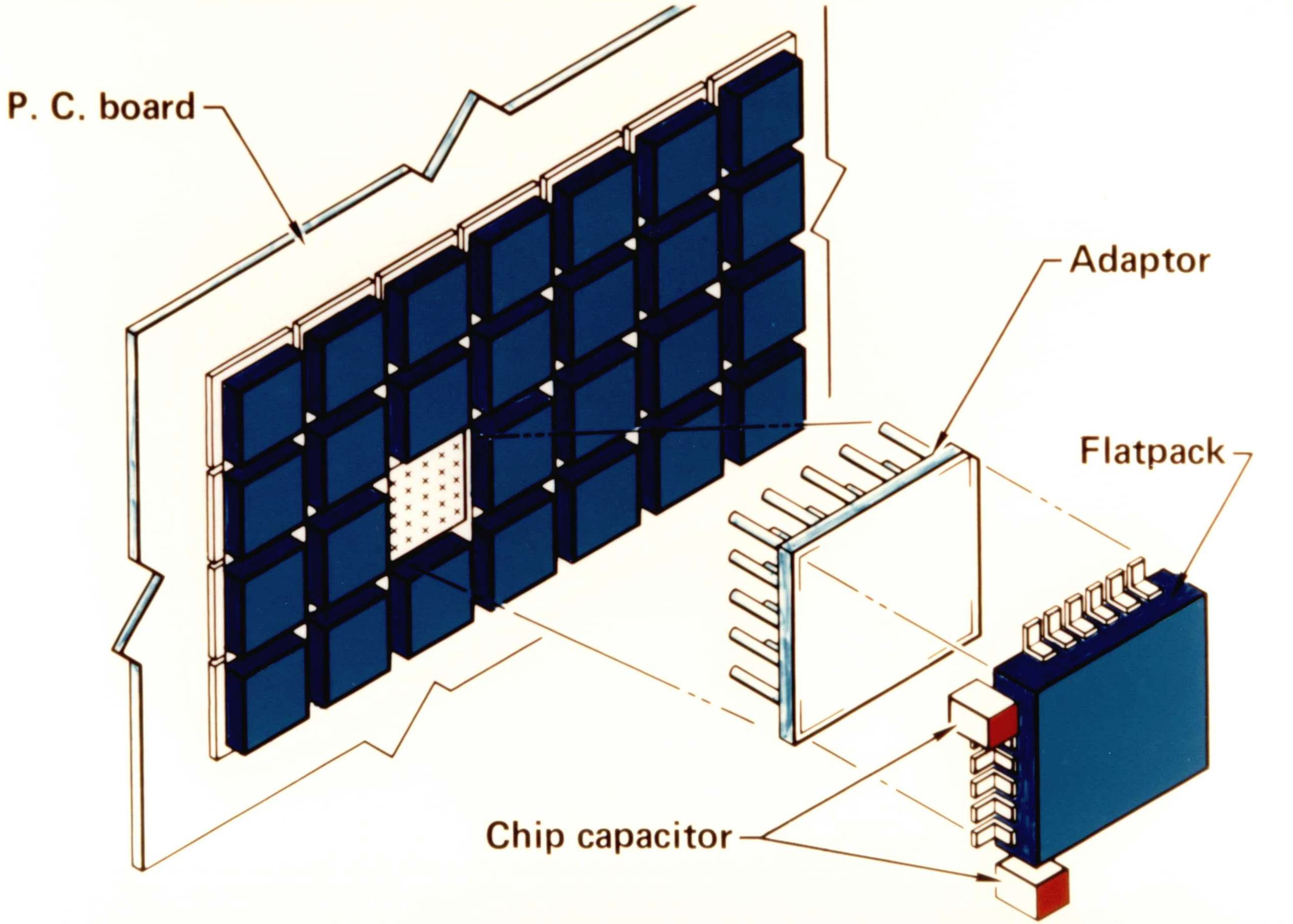
Interconnection Technology

- Flexible circuit interconnection
- Controlled impedance environment
- 990 signals and 990 grounds per board end

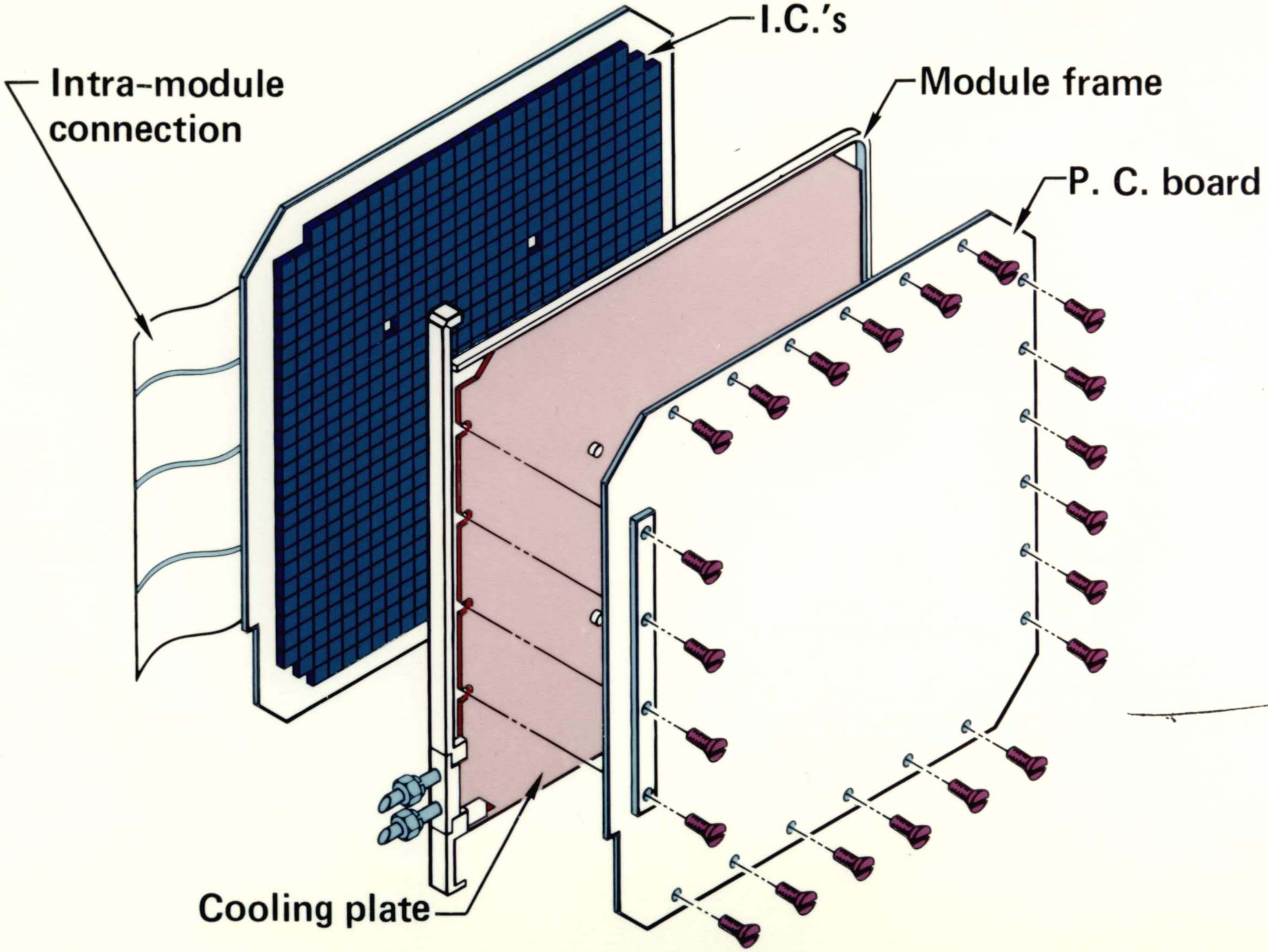
FLATPACK ADAPTOR



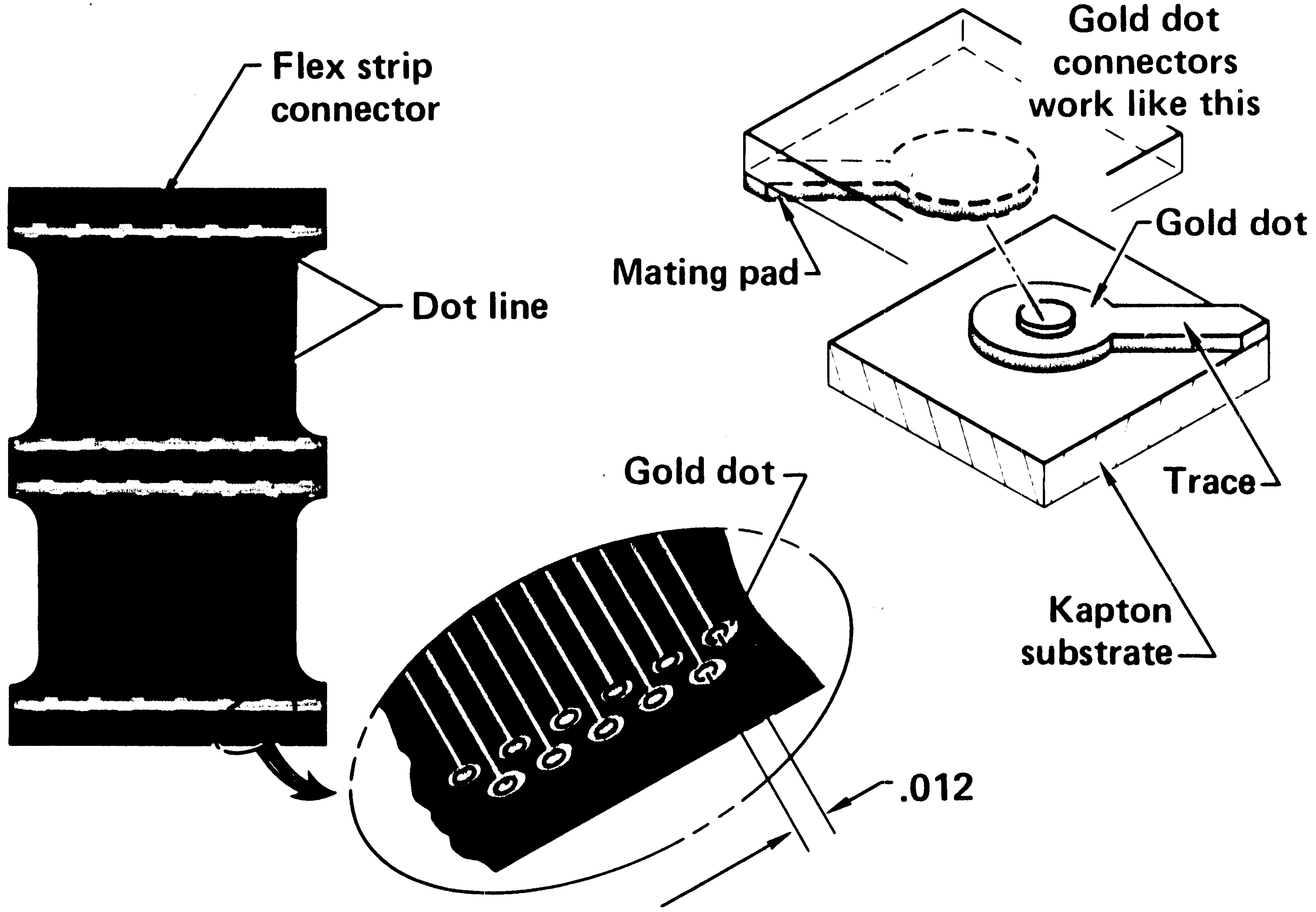
PRINTED CIRCUIT BOARD ASSEMBLY



PAGE ASSEMBLY



GOLD DOT FLEX STRIP CONNECTOR



Reliability of Packaging

- Water cooling brings junction temperature from >100°C to 35°C
  - Results in >100 × increase in chip lifetime
- Improved connector technology
  - Using missile-grade connectors
  - Hughes claims that Gold-Dot technology is most reliable ever made
- Absence of fan-induced vibration reduces connector wire-bond failures
- Redundancy in memory increases memory system MTBF
- Clean signal environment greatly reduces soft errors

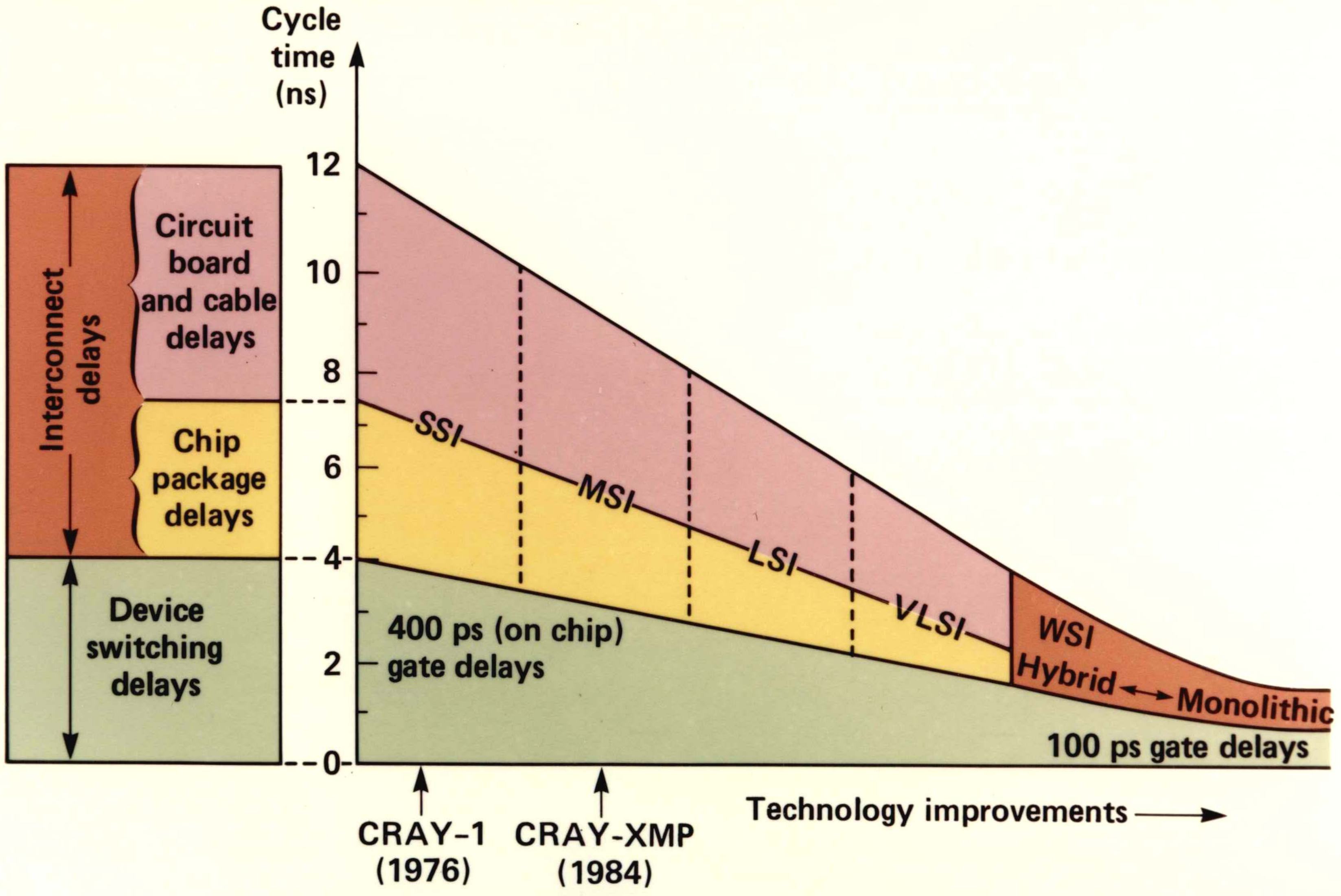
Laser Pantography Overview

Bruce McWilliams

Approaches to Enhancing Supercomputer Performance

- Increase the rate at which instructions are issued by:
    - Decreasing cycle time
    - Reducing memory access time
- Choose architectures and implementation specifically targeted for intended application

Impact of Technology Improvements on Supercomputer Speeds



Requirements on Process for Wafer-Scale Integration (WSI) of Supercomputers

- Order-of-magnitude speed improvement requires a process that
  - Supports compact integration of memory and logic units
  - Allows implementation of a high performance technology
- Short-term solution proposed is development of hybrid wafer-scale integrated circuits

Custom Architectures and Implementation for Scientific Computing

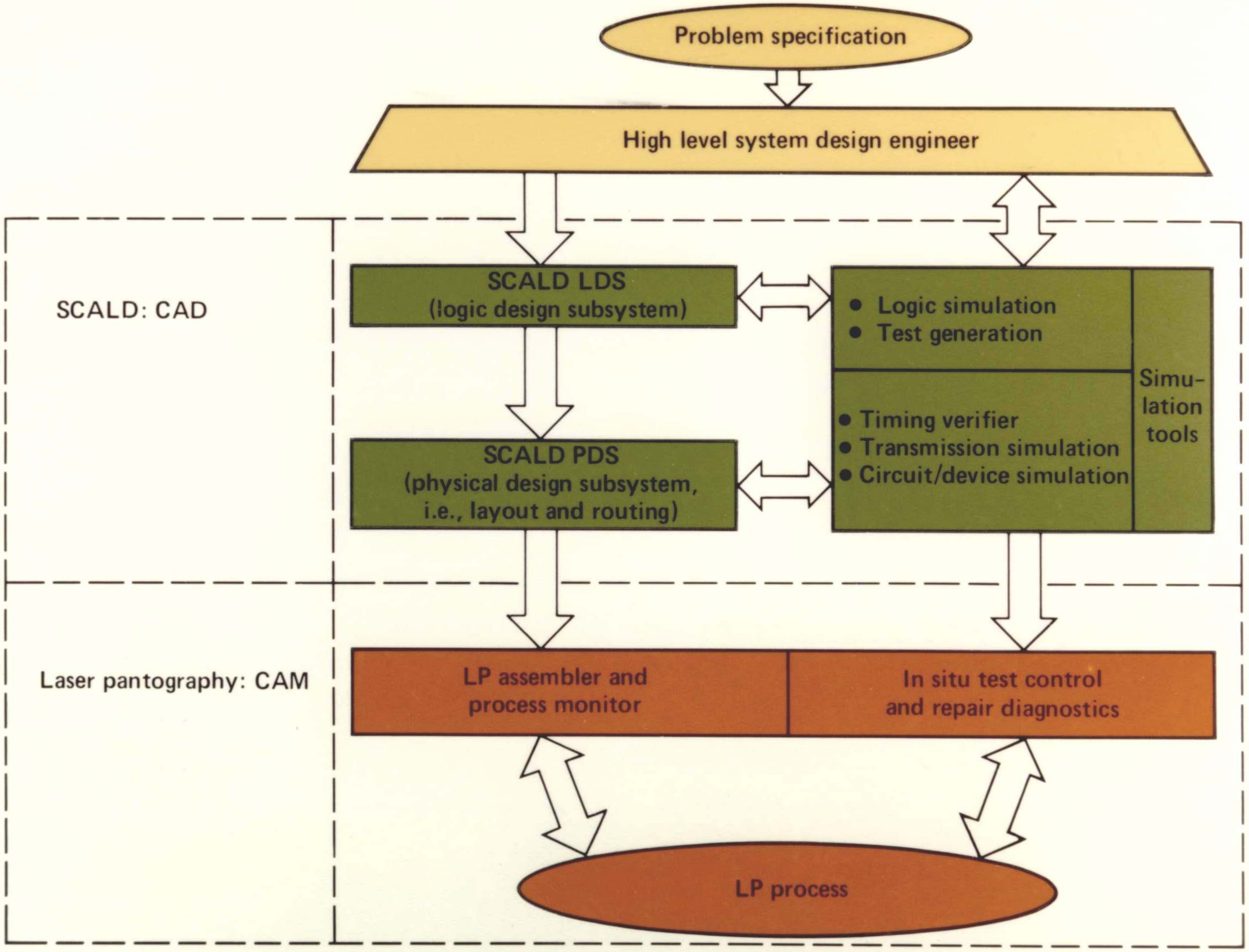
- At least two orders of magnitude increase in performance for wide variety of scientific computing applications

- Not practical unless:

  - Computers can be designed at high level so that complexity and effort is comparable to that of writing large modern scientific modeling programs

  - Rapid turnaround and moderate fabrication cost can be realized for such systems

Computer-automated computer design and fabrication

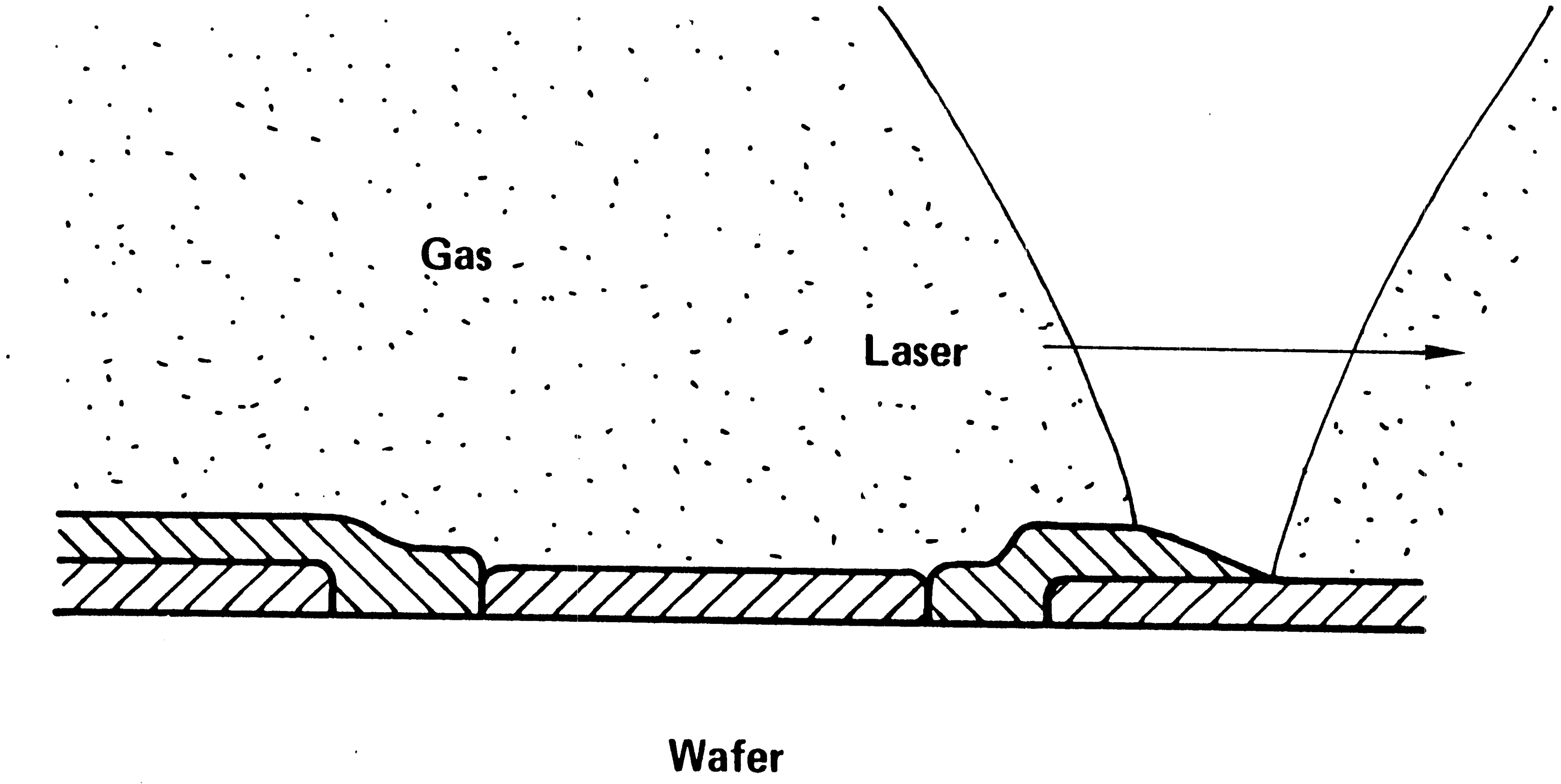


Long-Range Goals of Laser Pantography Project

- Wafer-scale integration of special-purpose computer architectures with computer-automated design and fabrication
  - Support 10^2 to 10^4-fold gain in computation rate by making possible efficient introduction of parallelism (special purpose computer architecture)
  - Allow short and affordable design completion-to-functioning prototype intervals (acceptable execution times and cost)

Laser Pantography

Direct-write process for IC fabrication that uses a laser beam focused directly on the wafer to induce local deposition or removal of material by means of gas/surface chemical reactions.



Direct-Write Laser Processes

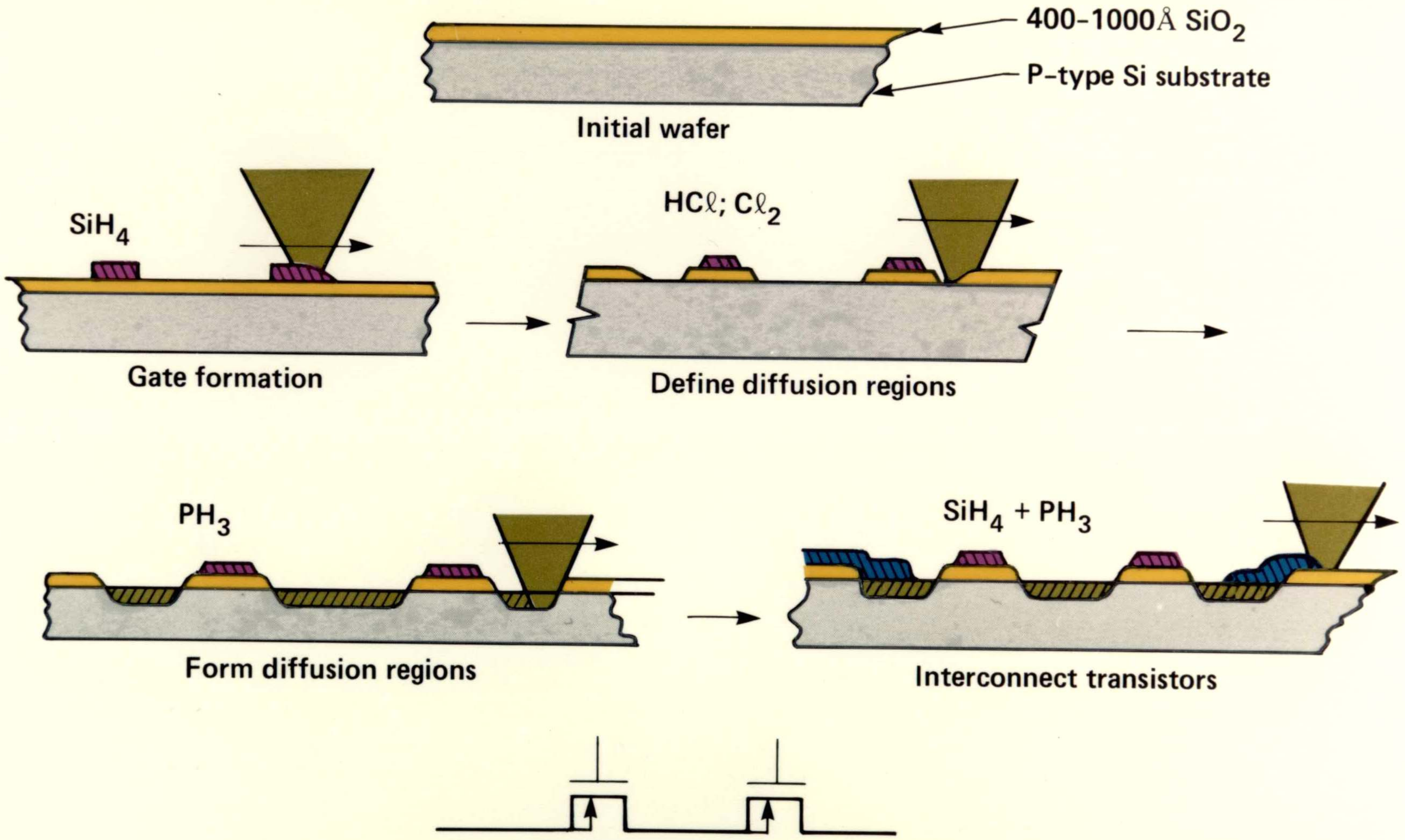
There are two basic types of laser direct-write processes:

- Pyrolytic
- Photolytic

They have been used to locally:

- Remove material from a semiconductor substrate
- Deposit materials on substrates
    - Semiconductors
    - Dielectrics
    - Metal
- Dope semiconductors

MOS Integrated Circuit Creation by Laser Pantography



Circuit equivalent

Reasons for Developing Laser Pantography

- Well-matched to modern CAD/CAM/CAE technologies
  - Designs in computer memories automatically 'pantographed' onto wafers
  - Lack of human participation supresses defects and increases speed
- 15-20 minute custom wiring of 2500 gate array chip
- Day-scale multilevel patterning of entire 5 inch diameter wafer
  - Entire supercomputer fabrication "overnight"

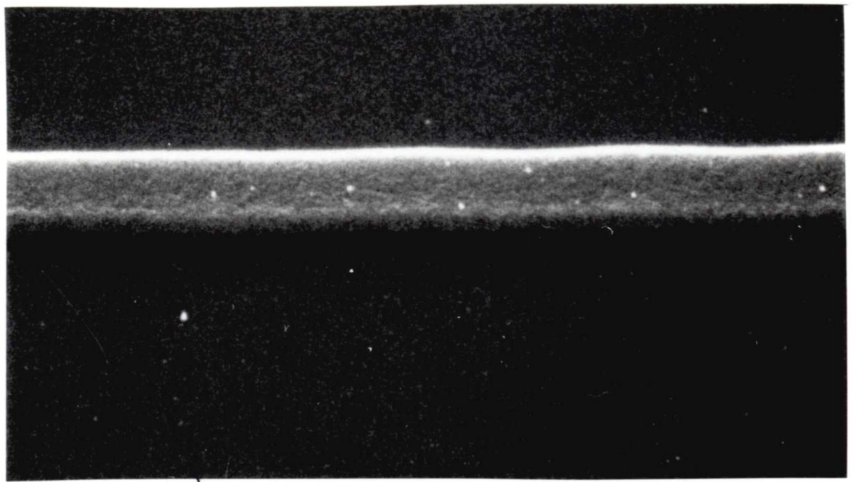
Reasons for Developing Laser Pantography

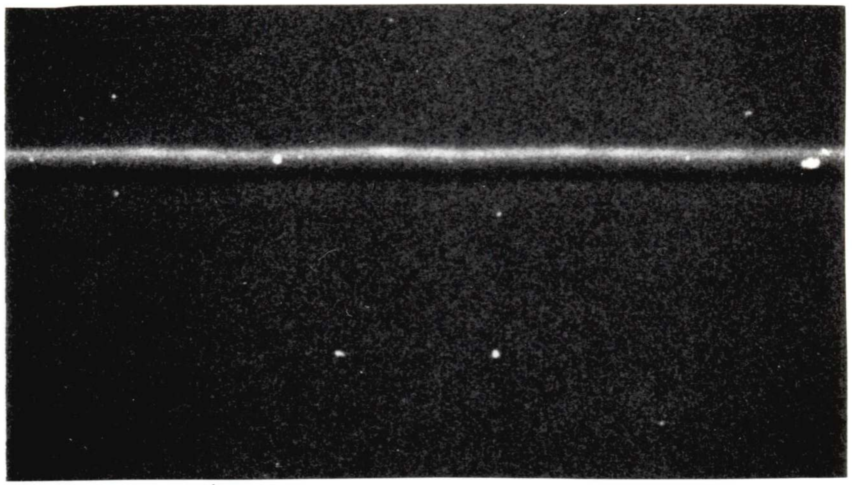
- Increased yield makes wafer-scale integration viable
  - Start-to-finish processing in a sealed environment
  - Maskless technology eliminates wet chemical processes, e.g., those involving photoresist
  - Serial nature of process permits defect correction after periodic in-process testing

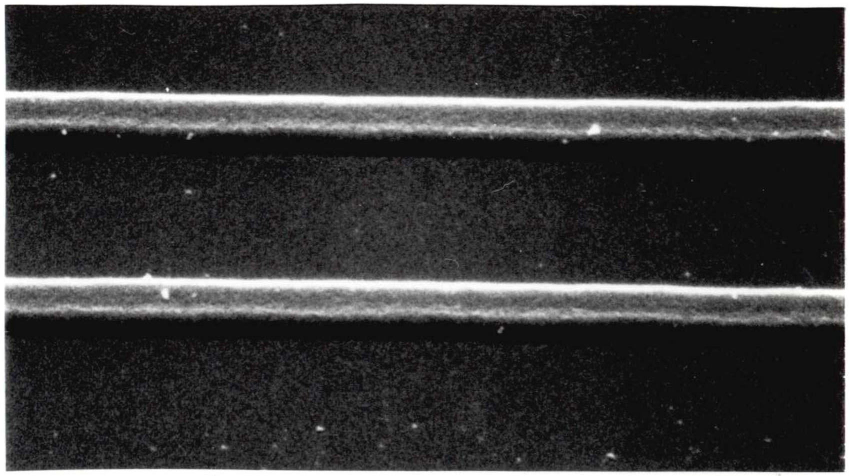
Laser Pantography Current Status

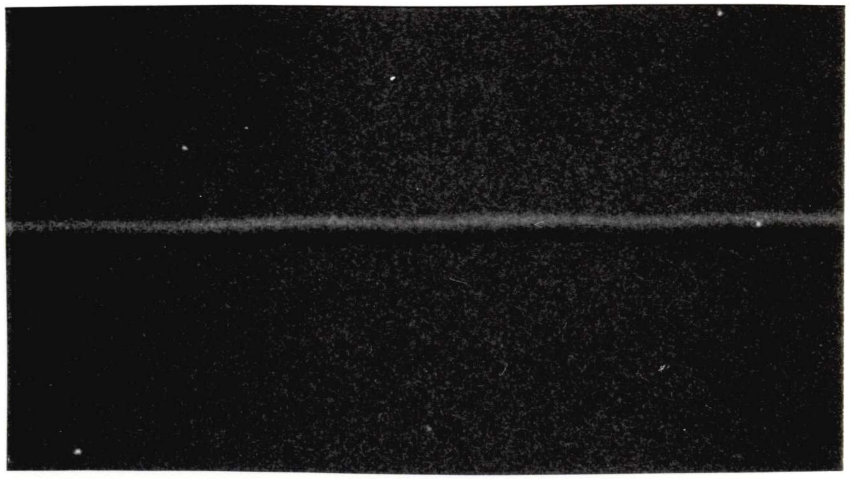
- Process development is focused on interconnecting metal structures for wafer-scale integration of gate arrays
  - One level CMOS gate arrays interconnect process approaching electrical characteristics available from best conventional lithography
  - Laser-written lines exhibit excellent surface morphology for 5.0-0.7 micron line widths
  - Six minute wiring of 1000 gate CMOS circuits will be possible once the new LP apparatus and software debugging is complete

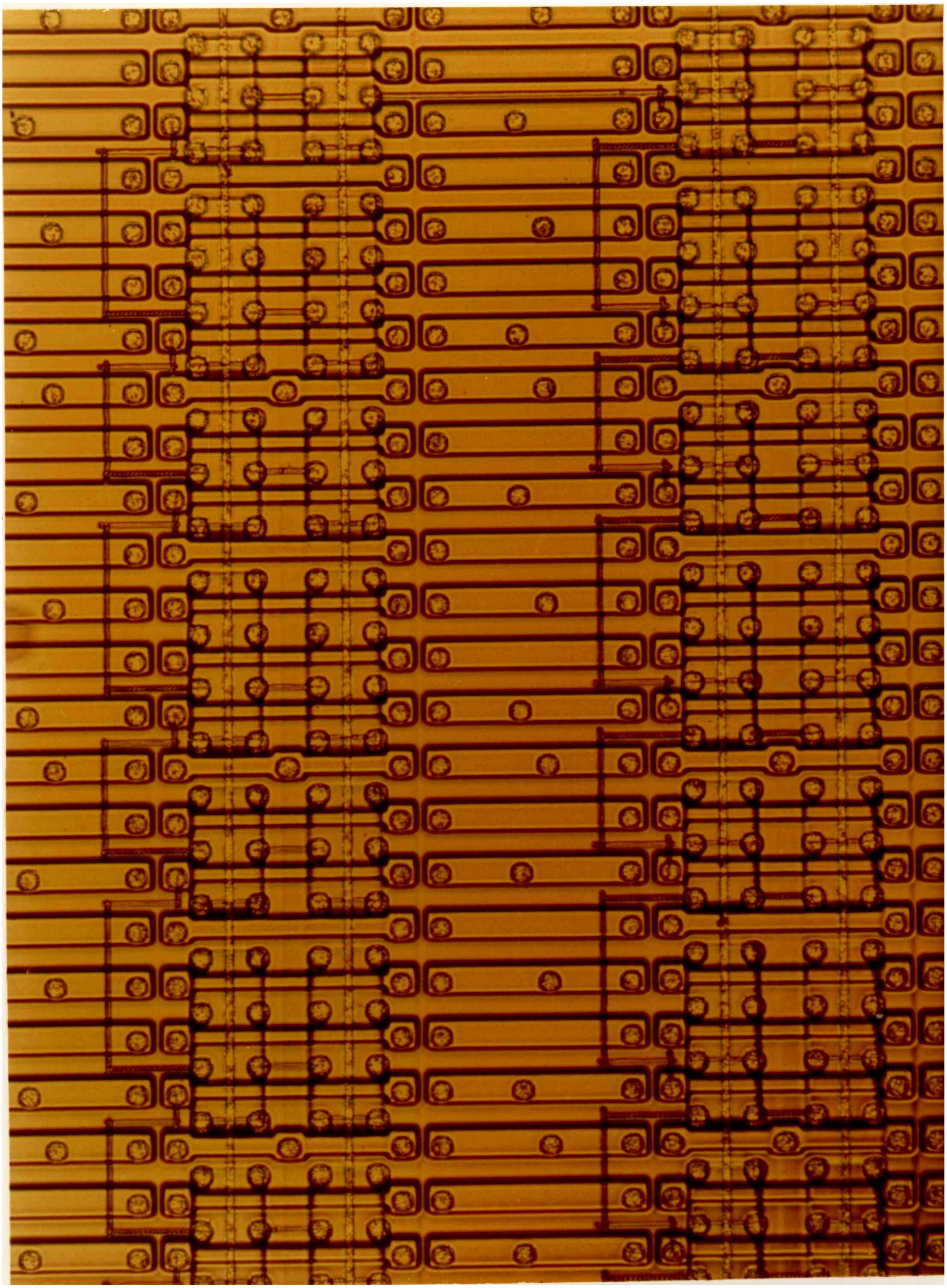
Polysilicon Interconnect Written by Laser Pantography Processes


Scale: |---4μ---|


Scale: |---4μ---|

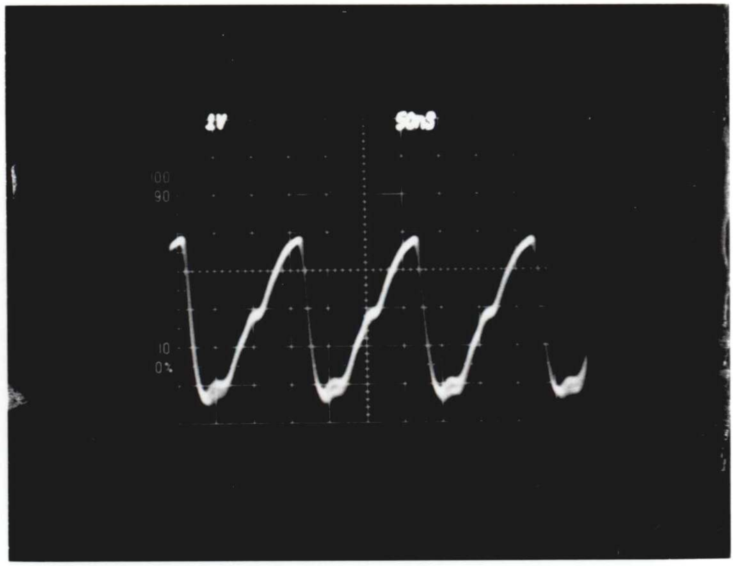

Scale: |---4μ---|


Scale: |---4μ---|

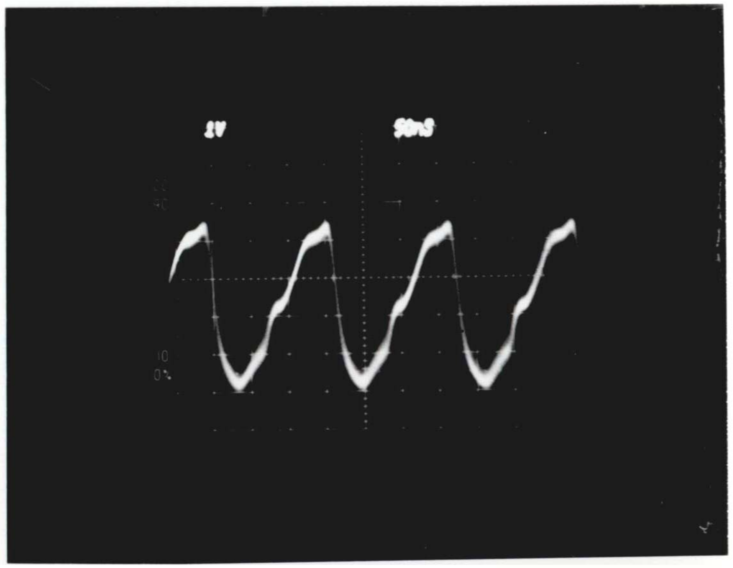


LP and Lithographically Patterned 31 Stage CMOS Ring Oscillators Perform Identically

Typical scope trace for 31 stage ring oscillator with interconnect fabricated by Lithography:



Typical scope trace for 31 stage ring oscillator with interconnect fabricated by Laser Pantography:



Time Required to Fabricate Circuits Using Laser Pantography

- Experiments indicate VLSI circuits can be fabricated at a rate of

$$10^4 - 10^5 \mu m^2/\text{sec}$$

- A state-of-the-art VLSI circuit with 10^6 devices covers roughly 1 cm^2 ($= 10^8 \mu$m^2) of substrate:

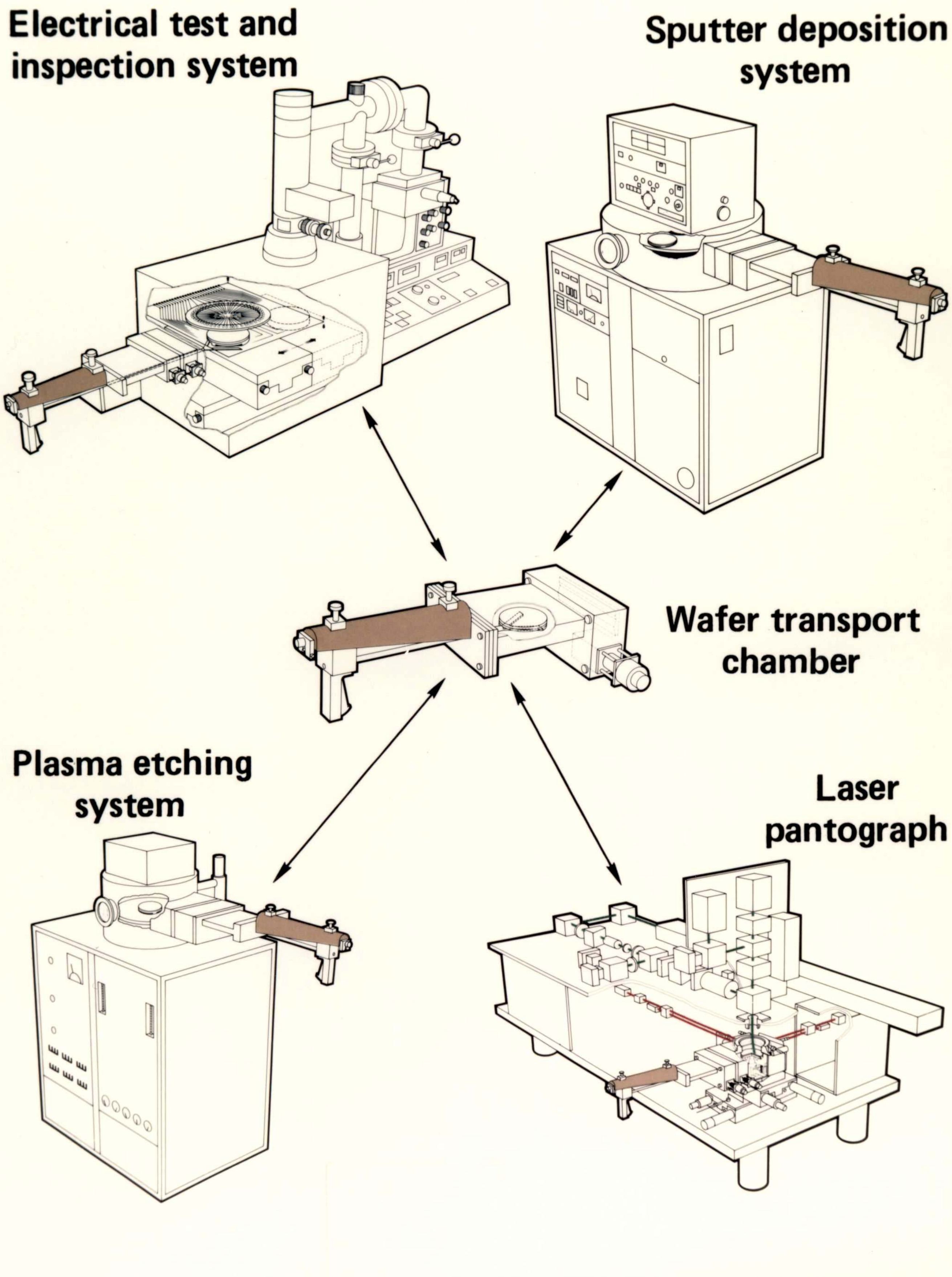
$$\begin{aligned}&\text{VLSI circuit fabrication time}\\ &= \frac{\text{area}}{\text{processing rate}} \simeq \frac{10^8 \mu m^2}{10^4 - 10^5 \mu m^2/\text{sec}}\\ &= 10^3 - 10^4 \text{ seconds} = 20 \text{ minutes} - 3 \text{ hours}\end{aligned}$$

- A supercomputer would consist of $\lesssim 10^2$ such circuits. Thus:

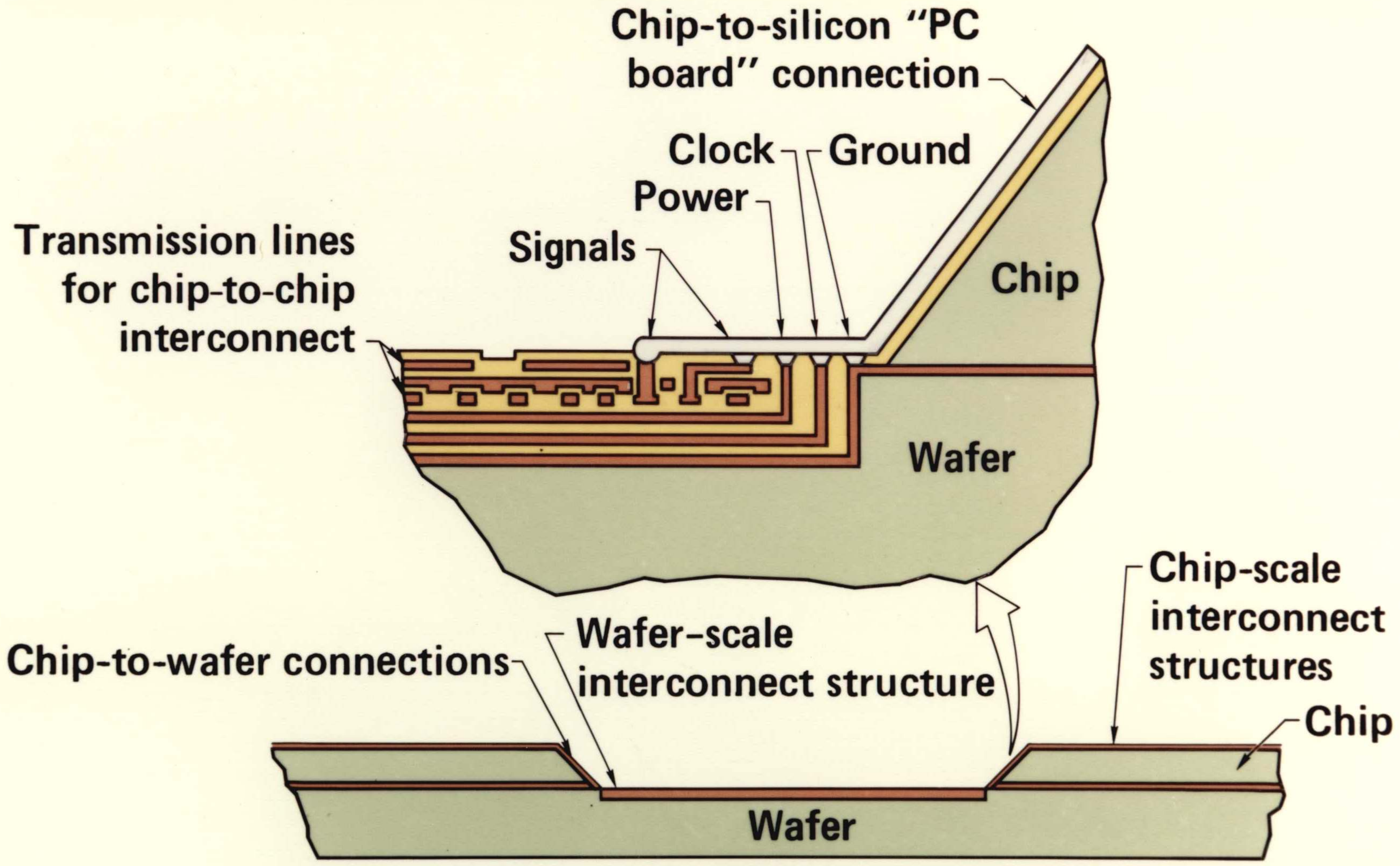
$$\begin{matrix}\text{supercomputer}\\ \text{fabrication time}\end{matrix} \lesssim 30 - 300 \text{ hours}$$

Technology Being Developed for Wafer-Scale Integration

- Equipment for fabrication and test of wafer-scale integrated circuits
- LP process for fabrication of semicustom VLSI components (e.g., gate arrays)
- Hybrid wafer-scale packaging technology
- Multilevel metal interconnect structures for wafer-scale integration circuits


Electrical test and inspection system
Sputter deposition system
Wafer transport chamber
Plasma etching system
Laser pantograph

Multilevel Metallization Scheme for Hybrid Wafer-Scale Integrated Circuits


Chip-to-silicon "PC board" connection
Clock
Ground
Power
Signals
Transmission lines for chip-to-chip interconnect
Chip
Wafer
Chip-scale interconnect structures
Wafer-scale interconnect structure
Chip-to-wafer connections
Chip
Wafer

Strategy of LP Experimental Efforts

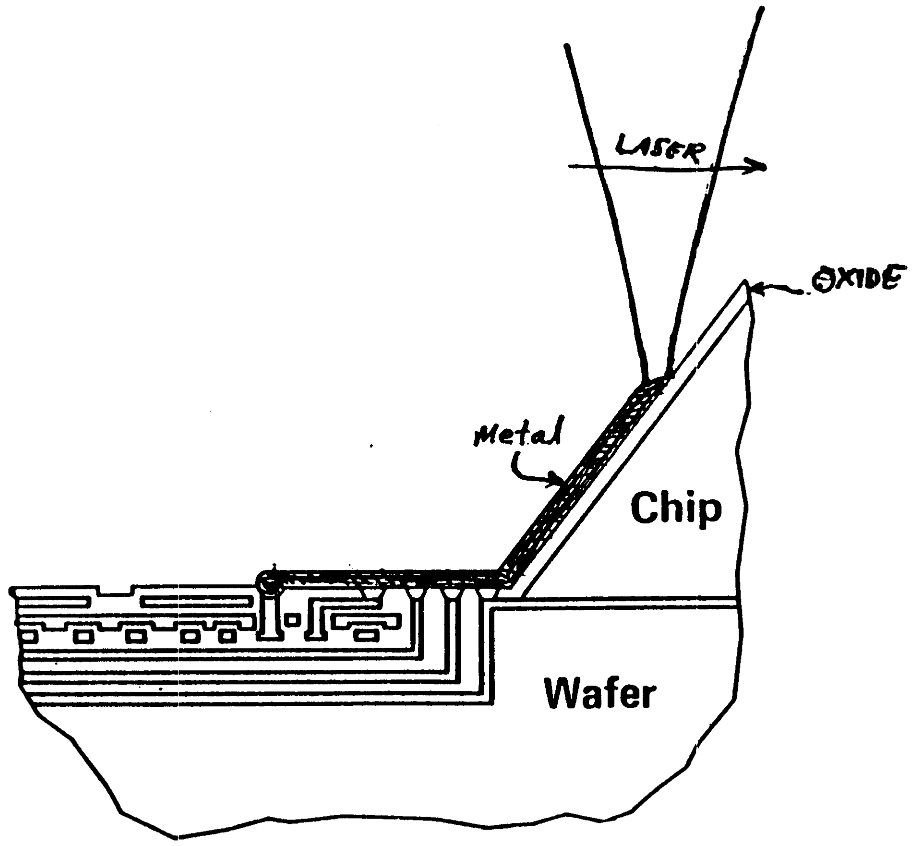
- Early efforts concentrated on laser fabrication of devices

- Present effort centers on refining direct-write processes for multilevel metal interconnect of gate arrays
  - Start with wafer covered with devices
  - Direct-write processes are used to pattern insulator and metal structures
  - Bulk processes for deposition and etching incorporated into fabrication process

- Long range plan is to refine complete set of laser processes for full custom IC fabrication

Application of LP to Hybrid Wafer-Scale Packaging Technology

- Unique feature of direct-write processes is its capability to write 3-D structures by using dynamic focus control
  - This feature will be utilized to connect chips to the wafer-scale interconnect structure


LASER
OXIDE
Metal
Chip
Wafer