INTERNATIONAL ATOMIC ENERGY AGENCY
UNITED NATIONS EDUCATIONAL, SCIENTIFIC AND CULTURAL ORGANIZATION

INTERNATIONAL CENTRE FOR THEORETICAL PHYSICS
34100 TRIESTE (ITALY) - P.O.B. 586 - MIRAMARE - STRADA COSTIERA 11 - TELEPHONE: 2240-1
CABLE: CENTRATOM - TELEX 460392-1

SECOND SCHOOL ON ADVANCED TECHNIQUES
IN COMPUTATIONAL PHYSICS
(18 January - 12 February 1988)

SMR.282/28

COMPUTER ARCHITECTURES

C.Halatsis
Aristotle University of Thessaloniki, Thessaloniki, Greece

COMPUTER ARCHITECTURES

by

C. Halatsis

Will review and present state-of-the-art on

1. VLSI uniprocessor architectures
   - CICS
   - RISC
2. Parallel processor architectures
   - SIMD
   - MIMD
3. Novel architectures
   - Systolic
   - Data Flow
   - Reduction
   - Logic

WHAT IS COMPUTER ARCHITECTURE?

- "The view of the programmer"
- whom programmer?
  - (micro-programmer) HOST MACHINE
  - machine/assembly language programmer
    - OS programmer ISP/PMS
    - compiler-write programmer ISP+
    - application programmer ISP-
  - HLL programmer

  view is language-dependent
- Thus, ARCHITECTURE ≠ IMPLEMENTATION
  however, interrelated (cost/speed)
- Traditionally, architecture is that of the Instruction Set Processor (ISP) level
- It moves towards the HLL level
- This move is encouraged by the evolution of HLLs/Algorithms/Concepts

Procedural → Declarative

Serial → Parallel

von Neumann → non von Neumann

left brain → right brain

CATEGORIES OF ARCHITECTURES

LOW LEVEL	CONTROL FLOW	(Procedural)
	DATA FLOW	(Single Assigment)
	DEMAND FLOW	(Applicative/Reduction) Functional
HIGH LEVEL	ACTOR	(Object-oriented)
	LOGIC	(Predicate logic)

RESPECTIVE LANGUAGES

PROCEDURAL : FORTRAN, PASCAL, ADA

SINGLE ASSIGNMENT : ID, VAL, VALID

APPLICATIVE : FP, SASL, LISP, MIRANDA

OBJECT-ORIENTED : SMALLTALK, POOL

PREDICATE LOGIC : PROLOG

CATEGORIES OF CONTROL DRIVEN COMPUTERS

- Differentiated by Parallelism of DATA and INSTRUCTIONS

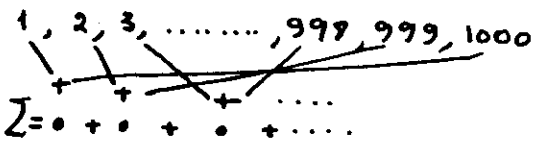
DATA STREAMS \ INSTRUCTION STREAMS	Single	Multiple
Single	SISD	MISD
Multiple	SIMD	MIMD

SISD : Single Instruction Single Data
MISD : Multiple Instruction Single Data
SIMD : Single Instruction Multiple Data
MIMD : Multiple Instruction Multiple Data

SISD : Conventional Uniprocessors - Serial
MISD : Pipelining & Vector Processors
SIMD : Associative & Array Processors
MIMD : Parallel processors (conventional)

WAYS TO SPEED-UP COMPUTATIONS : EXAMPLE

PROBLEM: Given



SOLUTIONS

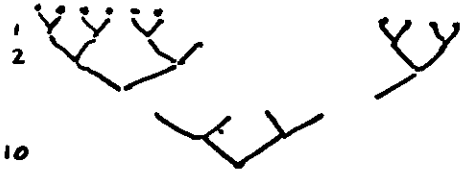
1) Serial (1 person / 1 processor)

No of steps = 500 + 500 − 1 = 999

2) Pipeline (1 2-person/2-processor pipe)

No of steps = 500 + 1 = 501

3) Parallel (500 persons / 500 processors)



No of steps = $\lceil \log_2 1000 \rceil = 10$

4) Intelligence: Recognize that Σ is an arithmetic progression

then $\Sigma = \frac{n(n+1)}{2}$

hence

no of steps = 3 !

WAYS TO SPEED UP COMPUTATIONS : SOLUTIONS

1) Serial : Better architecture, Faster circuit technology } Combine

2) Pipeline : Vectorizing & optimizing compilers

3) Parallel : Parallel algorithms and parallel programming languages

4) Intelligence : KIPS -

Knowledge Information Processing Systems

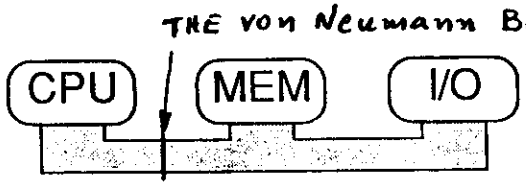
EXPERIENCE : Not so easy ways.

Problems start to appear as soon as we leave the Serial Solution

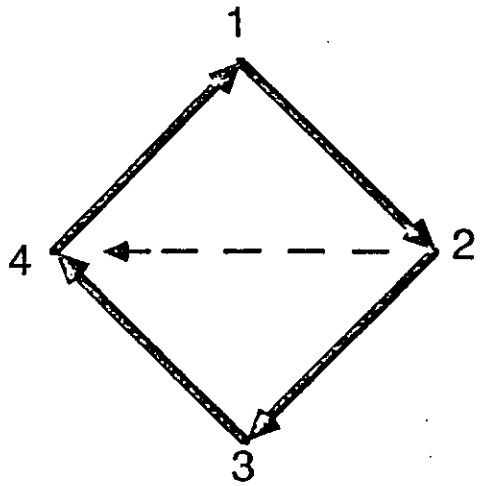
HOWEVER : VLSI & THEORY ARE THE DRIVING FORCE

Single Instruction Single Data (SISD)

Von Neumann Machine



1-Instruction fetch
2-Instruction decode
3-Operand fetch (if needed)
4-Instruction execution and write result.



SISD von Newmann ARCHITECTURES

DESIGN CONSIDERATIONS

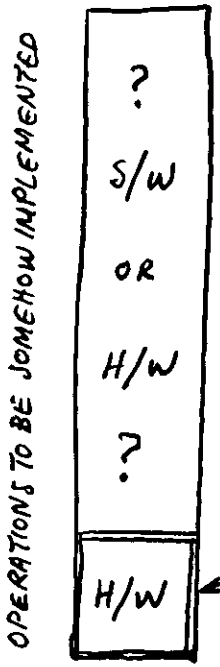
1) What to put into the architecture
   hardware/software tradeoff

2) Address space
   How big ?
   what organization / hierarchy ?

3) Word size
   what size ?
   Fixed vs. variable length

4) Instruction Set Architecture

HARDWARE/SOFTWARE TRADEOFF



← Absolute bare minimum H/W that can execute programs

other higher level operations:
in H/W or S/W ??
An operation:
if in H/W then one instruction/op
if in S/W then several instructions/op.

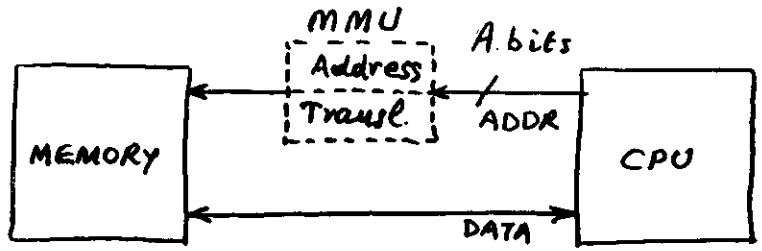
Optimization Rule: minimize

$$\sum_{i=1}^{n} \left(\begin{array}{c}\#\text{ of times} \\ \text{Operation}_i \\ \text{is executed}\end{array}\right) \cdot \left(\begin{array}{c}\text{time} \\ \text{to execute} \\ \text{Operation}_i\end{array}\right)$$

DO NOT minimize JUST this OR JUST this

What about the COST?

ADDRESS SPACE A = ? HOW MANY BITS

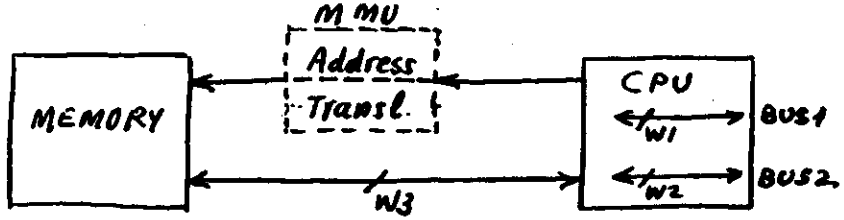


(VIRTUAL) ADDRESS SPACE = 2^A

A = 16 was enough in the late 60's
A = 20 to 24 barely enough today
A = 32 enough for some more years

⊛ Difficult and costly to recover from not providing enough address bits

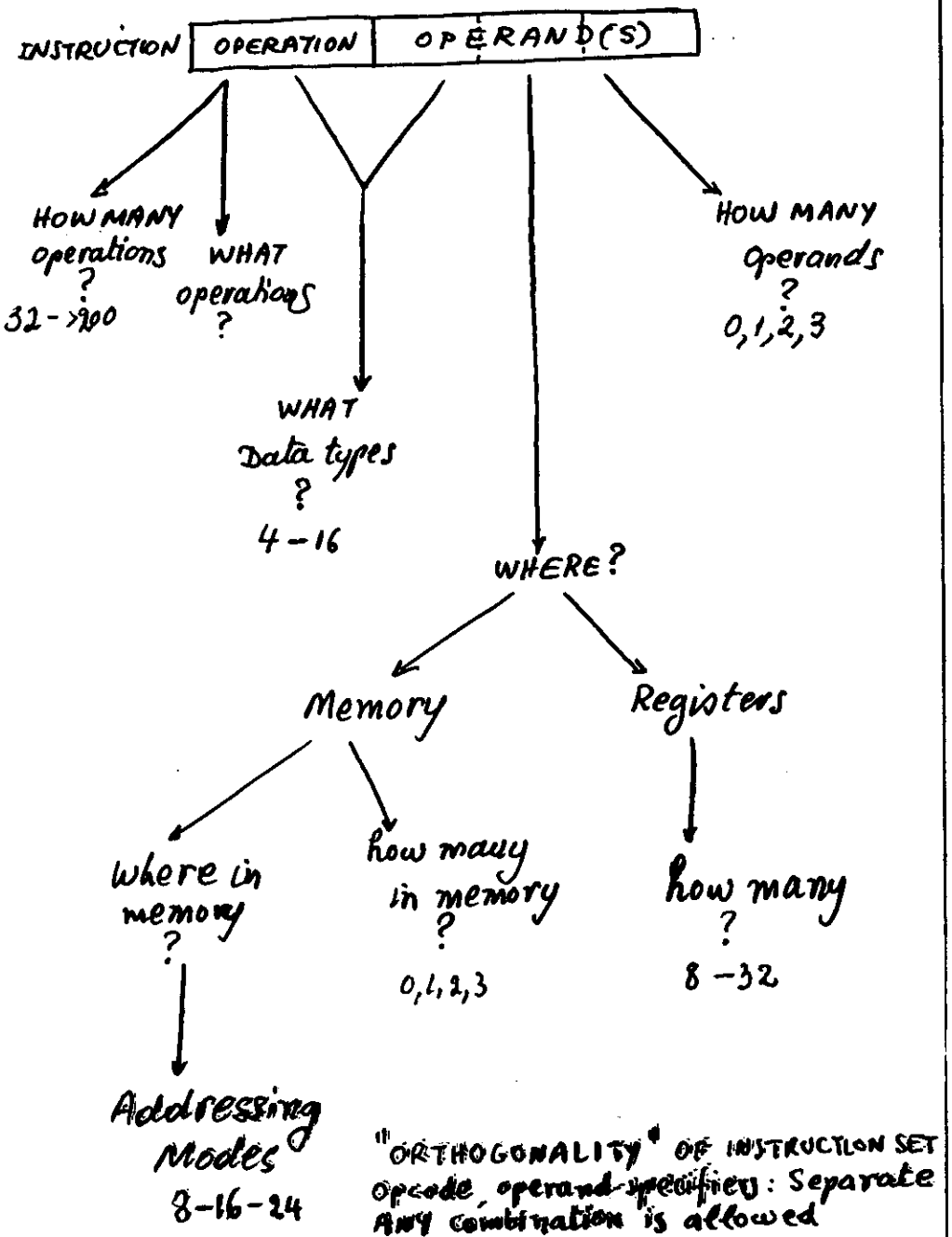
WORD SIZE W = ?



- W_1, W_2, W_3 ... Word Size (#bits/word)
- is indicative but non-binding characteristic
- can support sorter & longer data-types
- implementation may have narrower/wider datapaths
- Desirable $W \geq A$ addresses fit within words
  Nice $W = A$ for interchangability

INSTRUCTION SET ARCHITECTURES

- MANY OPTIONS -



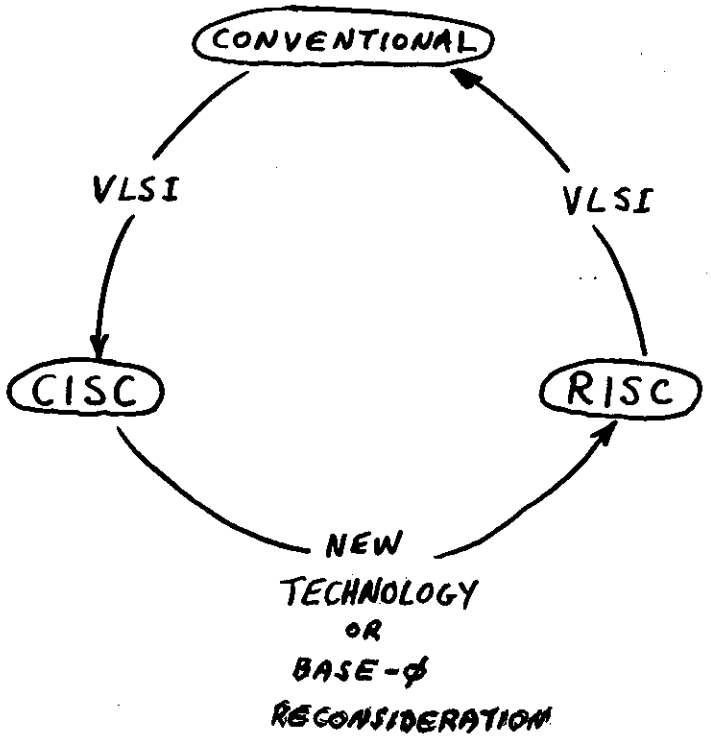
"ORTHOGONALITY" OF INSTRUCTION SET
Opcode, operand-specifier: Separate
Any combination is allowed

INSTRUCTION SET STYLES

CISC : Complex Instruction Set Computers
CONVENTIONAL
RISC : Reduced Instruction Set Computers

Examples:
CISC : VAX 11 , iAPX - 432
CONVENTIONAL : IBM/370 , M68000
RISC : RISC I, II , MIPS, IBM 801, (Cray)

THE EVOLUTION CYCLE :



WAYS TO SPEED UP A CONVENTIONAL UNIPROCESSOR

CONVENTIONAL → CISC

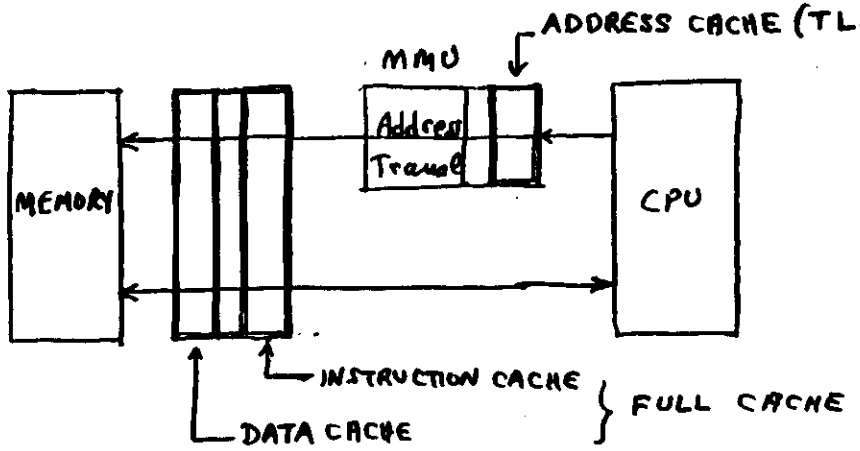
- MORE PARALLELISM (BUT STILL REMAIN SISD)
  - Larger word size
  - CACHE memory
  - Instruction PIPELINING
  - Separate DATA & PROGRAM memory (Harvard architecture)

- IN ADDITION / OTHERWISE
  - Special purpose instruction set
  - Special purpose hardware
  - More registers

LARGER WORD SIZE

- In the (μ) Processor world we have seen a move from 4 bits to 32 bits
- Thus a 32 databus allows fetch of integer data or pointer addresses of 32 bits in one memory cycle, whereas this requires 4 cycles in an 8-bit μP.
- At present we have 32-bit wide buses and we are contented with
  integers of 32 bits
  pointers of 32 bits
  floating point words of 64 bits
- Due to VLSI pin limitations it is very unlikely to move to 64 bit buses (and still handle 32-bit operands)
- Unless we widely employ WSI
  Wafer Scale Integration

CACHE (cacher = hide)



Address cache: Translate virtual address to memory address

it is called TLB: Translation Lookaside Buffer

Instruction cache: Saves only instructions
- exploits locality of control (small loops)
- avoids data coherence problems

Data cache: Saves data

PROBLEM: data consistency: CACHE COHERANCE

Full cache: Saves instruction + data
- cache coherence

ADVANTAGES:
- LESS MEMORY ACCESSES
- FASTER ACCESS TIME

QUICK REVIEW OF CACHE MEMORIES

- CACHE is a high speed buffer inserted between the CPU and Memory to capture that portion of memory which is currently in use
- CACHE is typically 5 to 10 faster than memory
- CACHE reduces the effective memory access time
- CACHE types: address, instruction, data, full
- CACHE characteristics:
  - Operation of a cache (hits - misses)
  - Design aspects
    - placement policy ← direct mapping, full associative, set associative (2,4)
    - fetch policy (on demand / prefetch / on write)
    - replacement policy (Random, FIFO, LRU)
    - main memory update policy ← copy back, write through
    - block size (4, 8, 16, 32, 64, 128, 256, 512) bytes
    - cache size (1, 2, 4, 8, 16, 32, 64, 128) kbytes
  - Effects of multiprogramming
    - process context switching
  - Data consistency - CACHE COHERENCE

    The problem is more difficult when we have ON-CHIP CACHES
  - Effective memory access time

$$t_e = h\,t_{cache} + (1-h)\,t_{memory}$$

h = hit ratio

BENCHMARKS USING ON-CHIP CACHE

600 ns/270 ns

Table 1.
Benchmark times (dynamic memory). These are absolute times, in seconds, for each processor executing each benchmark. (N = no cache, C = cache.)

		MHz	Benchmark E	F	H	I	K
INTEL	80286	(10)	4.89	13.63	6.59	11.20	19.39
	80386	(16)	3.57	5.16	3.63	6.86	6.20
MOTOROLA	68000	(8)	13.73	14.61	8.79	12.08	16.59
	68020 N	(16)	8.02	5.55	3.84	5.65	4.78
	68020 C	(16)	3.84	2.47	2.14	2.75	3.02
National	32032	(10)	12.52	13.07	6.21	8.57	13.07
AT&T	32100 N	(18)	16.81	6.84	5.05	8.57	9.17
	32100 C	(18)	6.75	4.29	2.74	3.63	4.45

Table 2.
Benchmark times (static memory). These are absolute times, in seconds, for each processor executing each program. (N = no cache, C = cache.)

	MHz	Benchmark E	F	H	I	K	
80286	(10)	3.18	6.81	3.46	5.98	10.27	
80386	(16)	1.92	2.53	1.53	3.18	3.68	190 ns
68000	(8)	8.79	9.67	5.60	7.69	11.37	
68020 N	(16)	3.74	2.74	1.87	2.69	2.64	230 ns
68020 C	(16)	2.52	1.98	1.26	1.92	2.25	
32032	(10)	11.04	10.05	8.83	6.59	11.65	260 ns
32100 N	(18)	9.28	4.72	2.52	4.45	4.84	220 ns
32100 C	(18)	6.04	3.07	1.81	2.97	3.46	

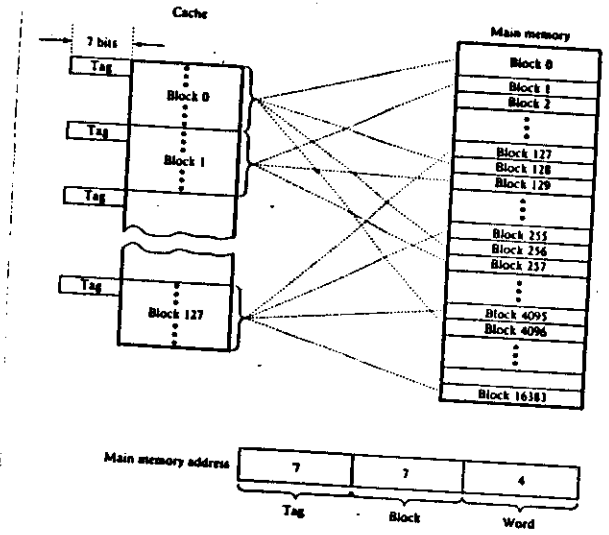
Table 3.
Mean of the test times for each processor with each memory, and the processor comparison ratio derived from the means. (Times are in seconds; N = no cache, C = cache.)

	MHz	Dynamic memory Mean	Ratio	Static memory Mean	Ratio
80286	(10)	11.14	3.92	5.94	2.98
80386	(16)	5.08	1.79	2.57	1.29
68000	(8)	13.16	4.63	8.62	4.33
68020 N	(16)	5.57	1.96	2.74	1.38
68020 C	(16)	2.84	1.00	1.99	1.00
32032	(10)	10.69	3.76	8.83	4.44
32100 N	(18)	9.69	3.41	5.16	2.59
32100 C	(18)	4.37	1.54	3.47	1.74

Benchmarks
- E : character-string search
- F : bit test, set and reset
- H : linked-list insertion
- I : quicksort
- K : bit-matrix transposition

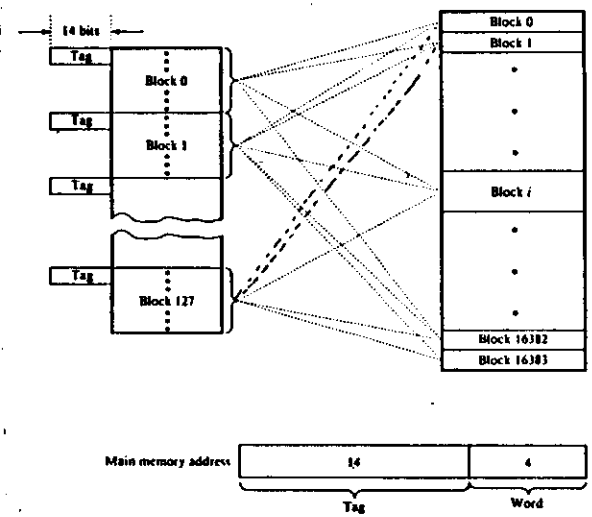
Cache placement policy: DIRECT MAPPING.

$BMM_i \rightarrow BC_i$ (modulo 128)



- It is the simplest organization
- Block i of the memory maps into cache block i modulo 128 (generally modulo the block capacity of the cache)
- The tag field contains the high order bits of the main memory of the cached block
- One only tag comparator is required

Cache placement policy: FULLY ASSOCIATIVE

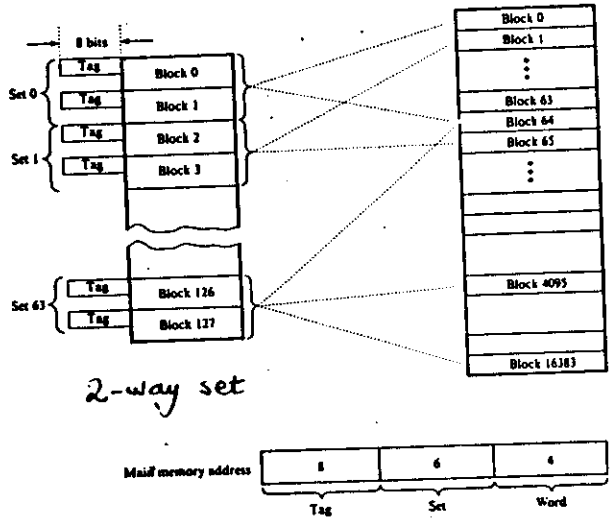


- It is the best but most expensive cache organization
- A main memory block may be in any cache block
- The number of tag comparators required is equal to the number of the cache blocks (e.g. 128)

Cache placement policy: SET ASSOCIATIVE

- A compromize between DIRECT & FULL ASSOCIATIVE
- The cache is divided into S sets of E = M/S blocks each

E-way set associative



2-way set

- E tag comparators are required
- A main memory block i may be in any of the cache blocks of set i modulo S
- It is the most commonly used organization

2-, 4-way associative

- 2- to 16 way set associative ≃ full associative (experimental results)

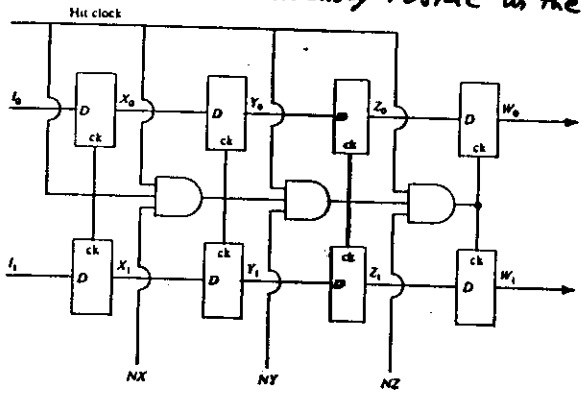
CACHE BLOCK REPLACEMENT POLICIES

- DECIDE which cache block to overwrite when the cache is full and a miss occurs

  Random
  FIFO
  LRU : Least Recently Used

  $$\frac{\text{Miss Ratio}_{FIFO}}{\text{Miss Ratio}_{LRU}} \simeq 1.12$$

- An implementation of the LRU policy using a set of D flip-flops to maintain the status of the blocks which currently reside in the cache



$NX = (X_i \oplus I_i) \cdot (X_0 \oplus I_0)$
$NY = (Y_i \oplus I_i) \cdot (Y_0 \oplus I_0)$
$NZ = (Z_i \oplus I_i) \cdot (Z_0 \oplus I_0)$

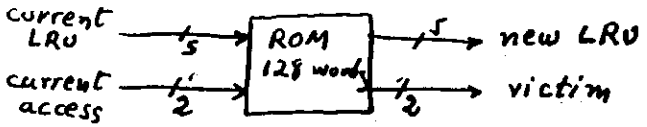
Figure 2.30 An implementation of the LRU algorithm.

- LRU implementation is generally hard, but

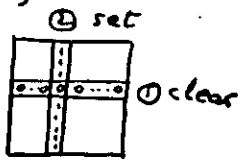
  2-way associative : trivial

  4-way associative

  4! = 24 ... 5 bits



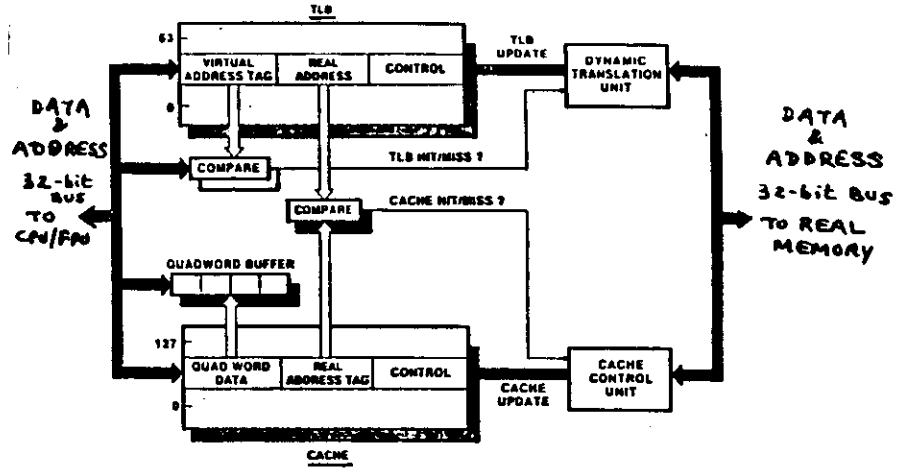
on access:



LRU is the row with all 1's

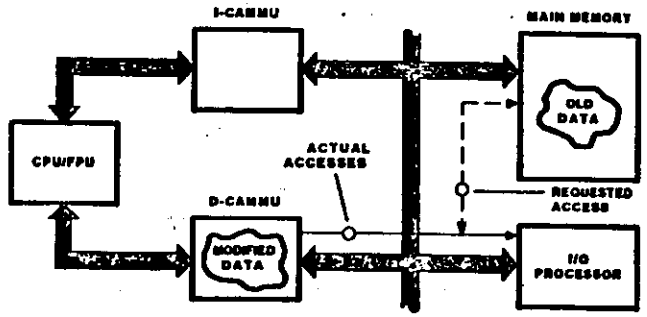
ADDRESS & DATA CACHES OF THE FAIRCHILD CLIPPER

VIRTUAL ADDRESS TRANSLATION & CACHE ACCESS



CACHE COHERENCE ON CLIPPER

BUS WATCH mechanism

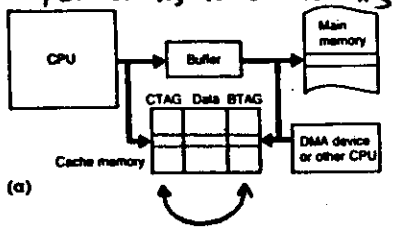


- it fulfills read requests by other bus masters from the cache instead of memory
- it ensures that writes by other bus masters update the cache as well as main memory

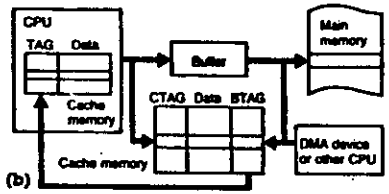
CACHE COHERENCE - BUS WATCHER

When a bus master, other than the CPU, modifies the main memory, the cached copy of that modified location must be invalidated

A bus watcher usually duplicates the CPU's tag store CTAG in a bus tag store (BTAG), so it can separate out write operations to locations stored in the cache



CACHE COHERENCE WITH ON-CHIP CACHE (National NS32532)

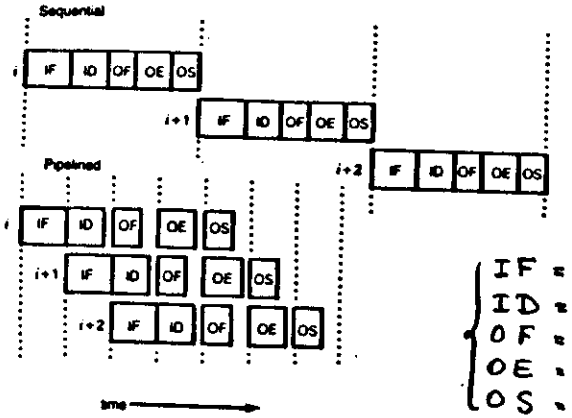


Use a separate port directly into the CPU tag store only for invalidation, so that invalidations can occur in one clock cycle

INSTRUCTION PIPELINING

- In its simplest form is a 2-stage Instruction fetch/Execute overlapping
- The effective speed is increased by a factor equal to the number of stages of the pipeline (peak)
- 2 TO 5 stages
- However, Data Dependencies and Branch instructions reduce the average instruction rate to below peak
  - Data dependencies and Branches create Bubbles in the Pipeline
  - There are techniques which remove some of the bubbles
    - data forwarding/chaining/bypass
    - optimized delayed branch (RISC)
  - The deeper the pipeline, the more difficult is to fill the bubbles.

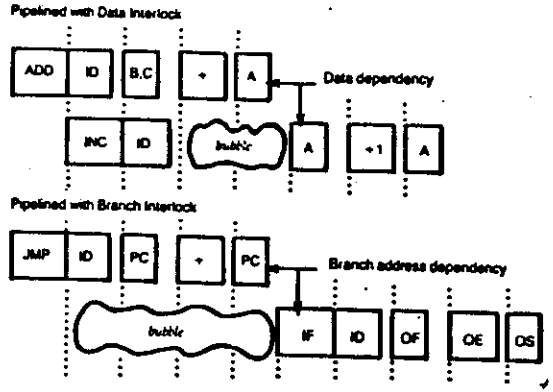
INSTRUCTION PIPELINING



IF = Instruction Fetch
ID = Instruction Decode
OF = Operand Fetch
OE = Operand Execution
OS = Operand Store

5-stage pipeline

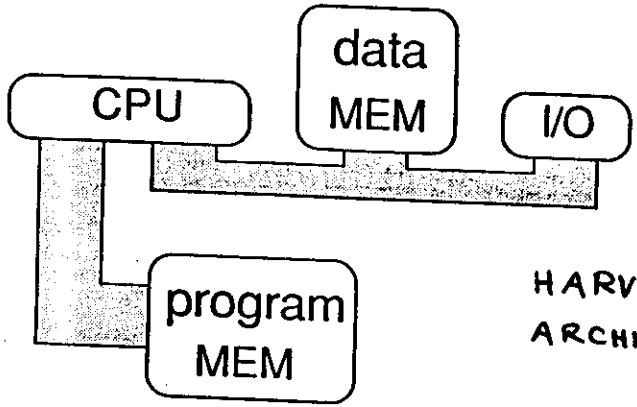
BUBBLES INTO PIPELINES



DELAYED BRANCH

Address	Normal branch	Delayed branch	Optimized delayed branch
100	LOAD X,A	LOAD X,A	LOAD X,A
101	ADD 1,A	ADD 1,A	JUMP 105
102	JUMP 105	JUMP 105	ADD 1,A
103	ADD A,B	NO-OP	ADD A,B
104	SUB C,B	ADD A,B	SUB C,B
105	STORE A,Z	SUB C,B	STORE A,Z
106		STORE A,Z	
107			

Separate data and program memory

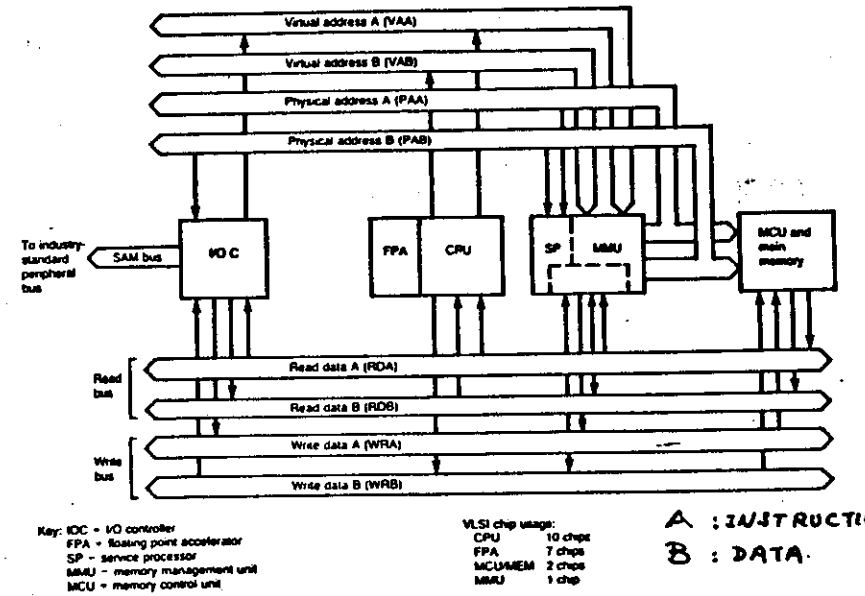


HARVARD ARCHITECTURE

-Allows parallel reference of data words and program instructions

-Difficult to move a program from disk to program memory

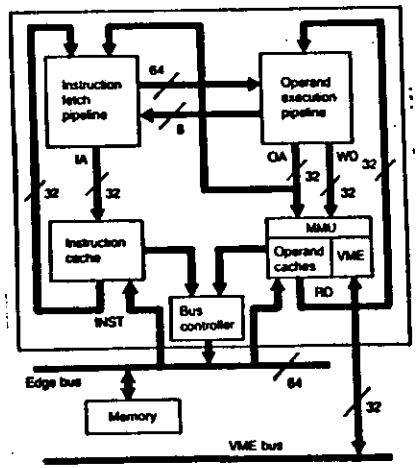
EDGE1 proprietory separate buses processor
(modified Harvard architecture)



A : INSTRUCTION
B : DATA

CMOS gate arrays

M 68010 compatibility



The EDGE1 processor architecture
a 4.5 MIPS machine

REGISTERS IN PROCESSOR

- ADVANTAGES
  - High speed data access
  - No external references leave bus free
- DISADVANTAGES
  - More complex instruction set
  - Compiler writing gets more difficult
  - Context switching gets slower because the registers must be saved
- Solutions
  - Overlapped register windows
  - Multi register sets
- Weighted against

CACHE memory

Instructionset Tuning

- Important for special pupose processors, critical functions can be supported by hardware
- Complex instructionset (CISC) in general purpose processors:
  - -Multiple addressing modes
    - -Bits, arrays, structures, stacks
  - -Floating point
  - -String
  - -Stackframes
- Why CISC?
  - -Functions performed by hardware are faster than software
  - -Compilerwriting gets easier

General purpose microprocessors (CISC)

-Operating System support
- -Memory management, segmented virtual memory
- -Security, OS/user priviliged instructions, no accidental overwriting of segments

-Compiler support
- -Addressing modes, stacks,arrays, structures
- -Uniform instruction set

-DMA support
- -Allows parallel I/O and processor actions
- -Arbitration needed

-System developement support
- -Debugging at hardware level
- -Hardware trace option
- -Hardware breakpoints

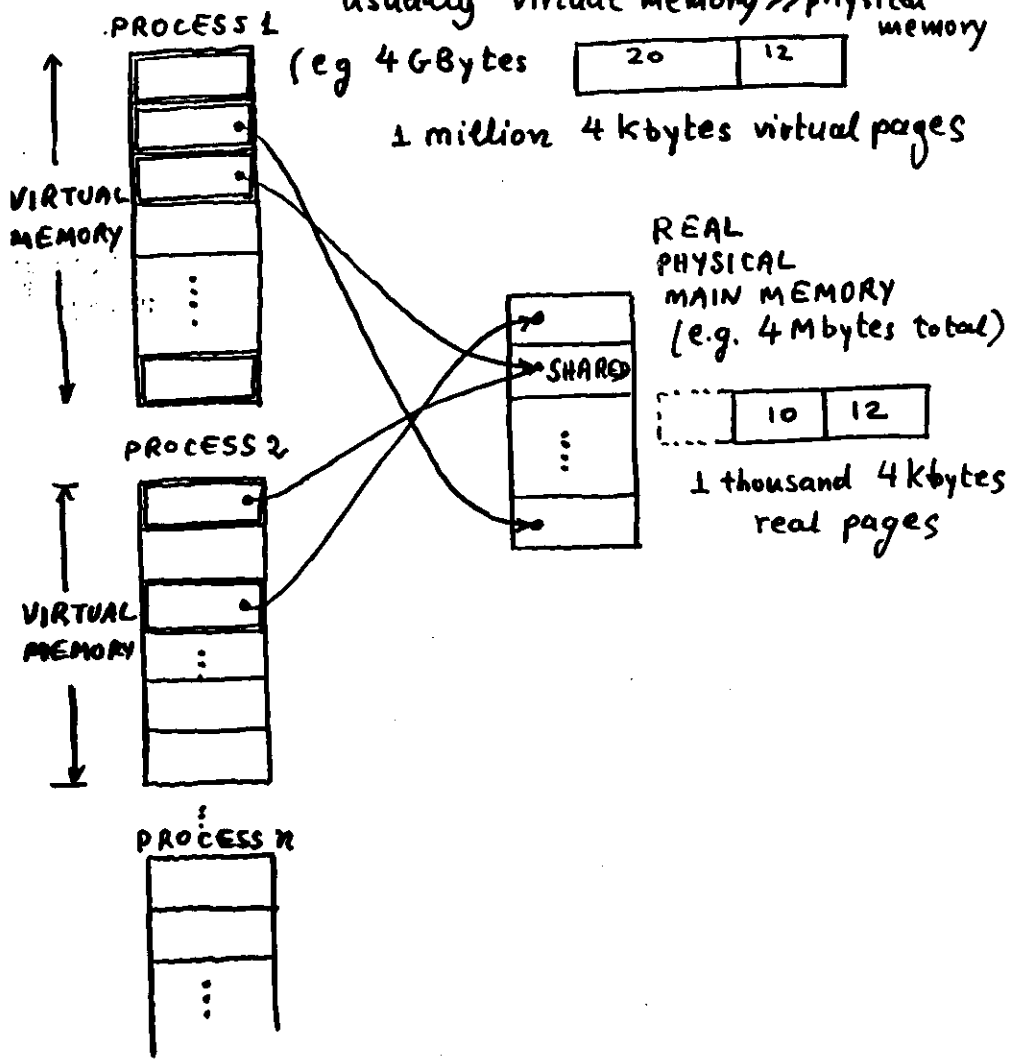
-Speed
  -1 to 5 MIPS
  -Cashe, full, address, instruction
  -Pipelining, parallellisme
  -Internal registers (16 words)

-Compatibility
  -Upward compatibility to simplify system design
  -NS16016,16032, 32032
  -MC68000, 68010, 68020
  -I8086, 80186, 80286
  -I8080, Z80, Z800, Z8000, Z80000

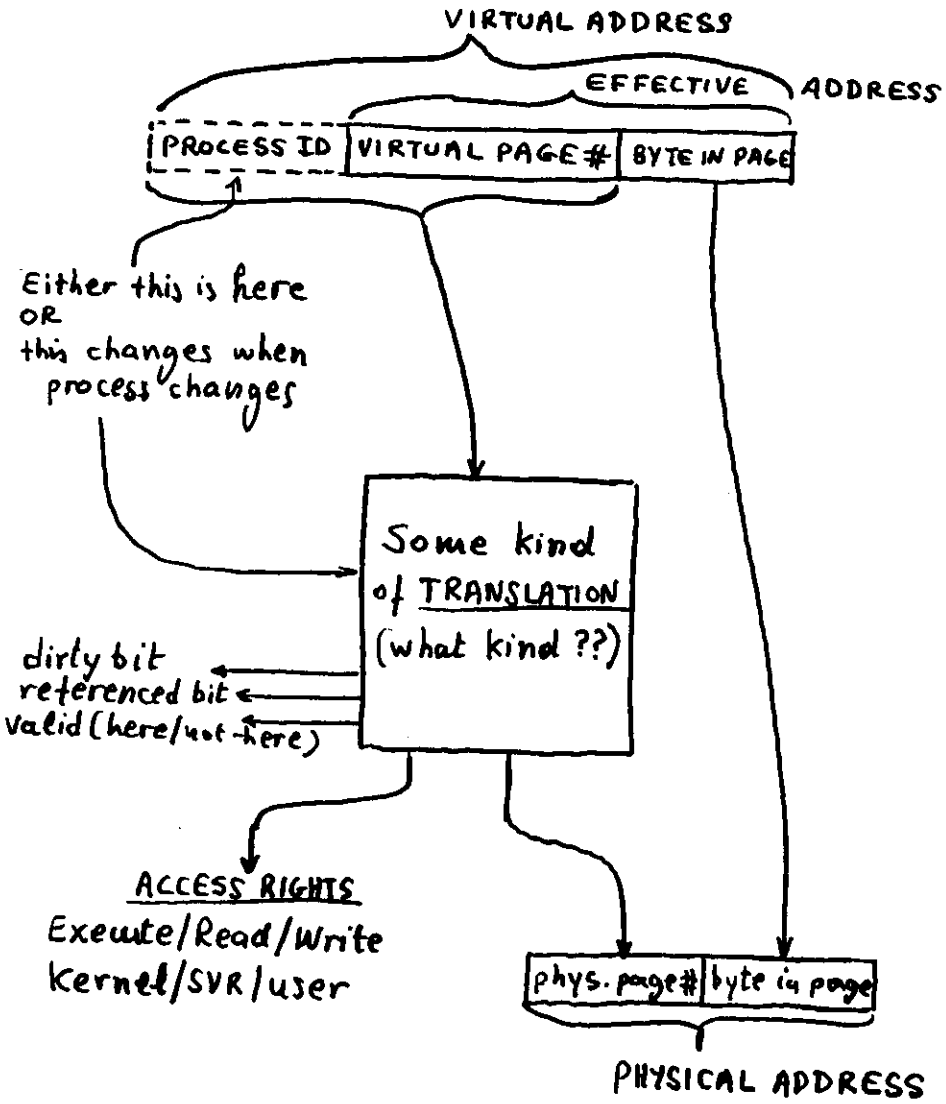
-Coprocessors
  -Space limitations forces implementation of some function on seperate chips
  -Allows flexible hardware extension
  -Memory Management Units (MMU)
  -Floating Point Units (FPU), MC68881 FPU, full IEEE FPU implementation

VIRTUAL MEMORY

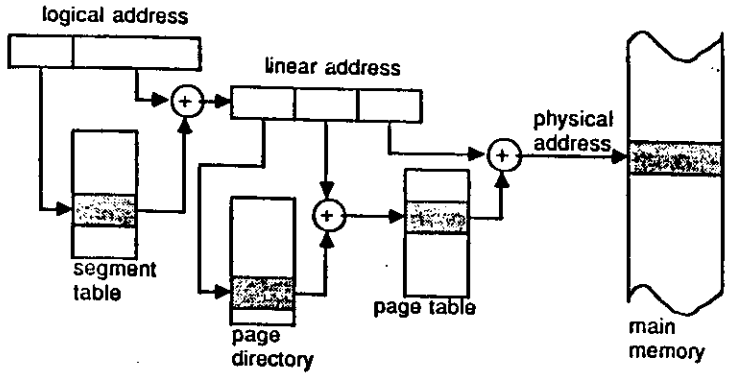
BASIC IDEA : Each process has its own memory space
Usually virtual memory >> physical memory



VIRTUAL ADDRESS : USUAL SCHEME



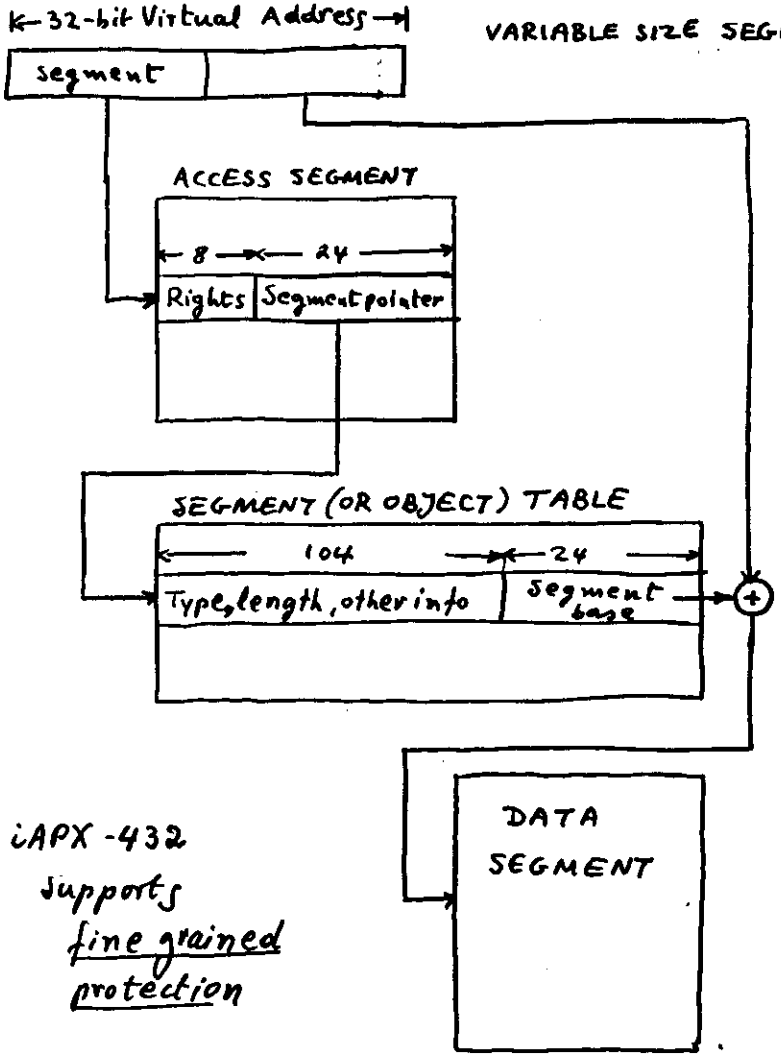
Memory management
(Intel 80386)



-Segmentation allows seperate address spaces for processes
-Segmentation prevents accidental over-writing of memory used by another proces

-Virtual memory space allows a very large address space
-The virtual address space is divided in pages
-Pages not in main memory are in background memory (disk)

iAPX-432 CAPABILITY-BASED ADDRESSING SCHEME



- iAPX-432 supports fine grained protection
- iAPX-432 supports object-oriented environments (Smalltalk ...)

The Intel iAPX-432 processor

- represents the most complete approach to integrate the needs of an entire software environment into silicon
- main characteristics
  - dense instruction decoding
  - object-oriented support mechanism (creation & protection of an object)
  - packet-switched bus protocol
  - OS support
  - multiprocessing support
  - fault-tolerant operation
- Three major chips
  - instruction decode unit
  - microexecution unit
  - interface processor
- no registers or stack cache, instead
  - memory and stack operands
  - bit encoded instructions
  - data typing
  - symmetric addressing
  - 1-, 2-, 3- operand instructions
- 16 Mbytes/sec bus
- 222 instructions
  6-321 bits/instruction
- NMOS technology

- Micro Programming !!

  - CPU is controlled by an interpreter in the μ-store. Instructions in the μ-store are executed directly in hardware.
  - Separation of conventional machine level and hardware implementation.
  - Upward compatibility can be realised in μ-program.
  - Modification of the hardware implementation only affects the μ-program (interpreter)
  - Moving functions from software to the μ-program results in a faster computer.
  - High instruction code density. (Small program size)

- Microprogramming still the correct way to go??

  - The difference in access times of μ-store and main memory has decreased. (was factor 10, now 5)
  - Not all implemented instruction are used frequently.
  - Additional instructions may slow down existing ones. (n+1 effect)
  - μ-programs become increasingly longer and more complex to maintain upward compatibility.
  - High level instructions requiring many μ-instuctions do not necessarily match a given high level language.

- ALTERNATIVE : RISC approach

RISC DESIGN PHILOSOPHY

1. Analyse target applications to determine which operations are used most frequently.
2. Optimize the data path design to execute these instructions as quickly as possible.
3. Include other instructions only if they fit into previously developed datapath, are relatively frequent, and their inclusion will not slow the execution of the more frequent instructions.
4. Apply a similar strategy to the design of other processor resources. Include a resource only if it is justified by its frequency of use, and its inclusion will not slow other, more frequently used, resources
5. Push as much complexity as reasonable from runtime hardware into the compile-time software

- The RISC designer is free to make tradeoffs across boundaries of
  - architecture / implementation
  - hardware / software
  - compile-time / run-time

FEATURES COMMONCY SEEN IN RISCs

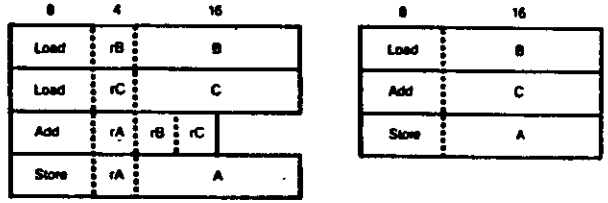
1. Single cycle execution of most instructions
2. Load/store instruction set (register-to-register) (compare with m-to-m, m-to-r)
3. Hardwired instruction decoding
4. Relatively few instructions and address modes,
5. Fixed instruction format for simple decoding. (see comparison of RISC I, VAX, iAPX-432)
6. Complexity pushed into optimizing compiler, (see comparison with m-to-m)
7. Highly pipelined datapath for much concurrency
8. Large register set (windowed or not windowed)
9. Many levels of memory hierarchy,
10. Instruction set designed for a specific application class

REGISTER/MEMORY EXECUTION MODELS

EXAMPLE : A ← B + C

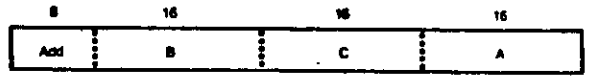
ARCHITECTURE METRICS:

I : total size of executed instructions
D : total size of executed data
M : total memory traffic



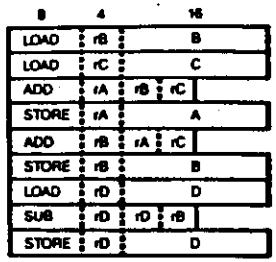
I = 104b; D = 96b; M = 200b (Register-to-Register)

I = 72b; D = 96b; M = 168b (Memory-to-Register)



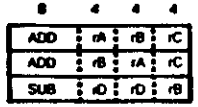
I = 56b; D = 96b; M = 152b (Memory-to-Memory)

OPTIMIZING COMPILER EFFECT

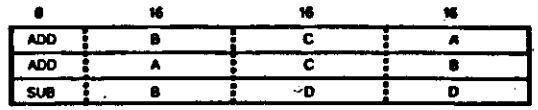


Reuse of Operands
I = 228b; D = 192b; M = 420b
(Register-to-Register)

A ← B + C
B ← A + C
D ← D − B



Compiler allocates Operands in Registers
I = 80b; D = 0b; M = 80b
(Register-to-Register)



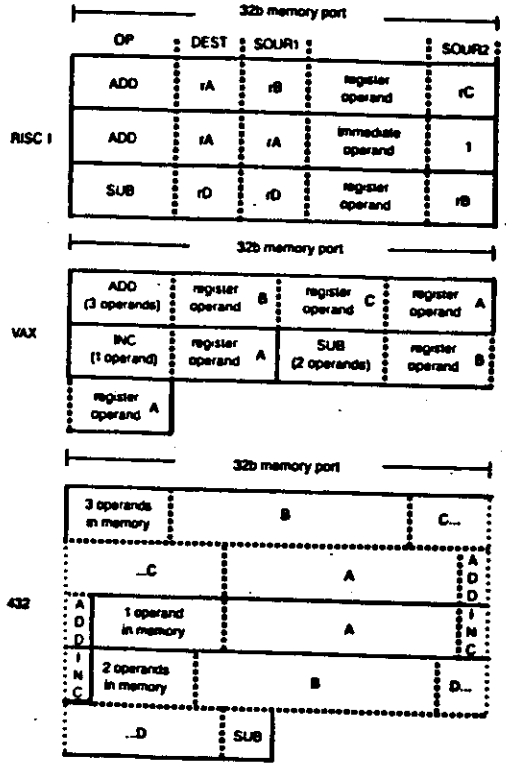
I = 168b; D = 288b; M = 456b
(Memory-to-Memory)

COMPARISON OF THREE MACHINES

(RISC I, VAX, LAPX-432)

FOR THE EXECUTION SEQUENCE

A ← B+C
A ← A+1
D ← D-B



CONCLUSION :

Although Variable-sized Instructions improve the architectural metrics, they also make instruction decoding more expensive

MIPS (Stanford University)

Microprocessor without Interlocked Pipe Stages

True 32 bits architecture

2 µm single metal nMOS VLSI implementation.

16 general purpose registers

6 pipeline stages ?

cycle time 250 ns (4 Mhz)

Delayed branching scheme

Two level instruction set architecture ← ASSEMBLY MACHINE
Instruction reordering
- exploiting the delayed branch instructions
- removing depencies between instructions in pipeline.
- packs instructions. (code density > 1 instruction/word)

Compiler performs a cost driven inline code expansion.

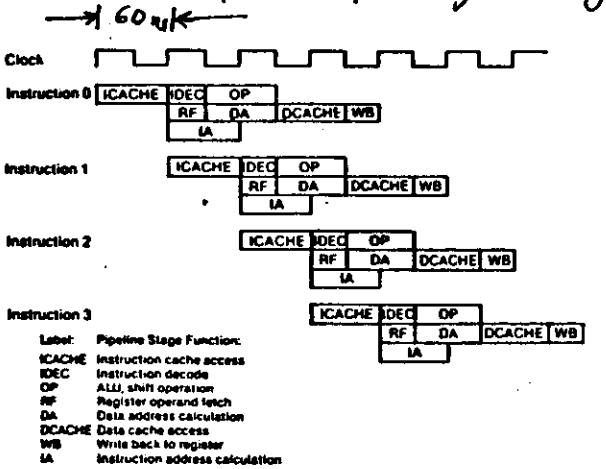
OS protection: user- and priviliged mode

User protection: segmented/paged virtual memory

system calls are traps to the OS.

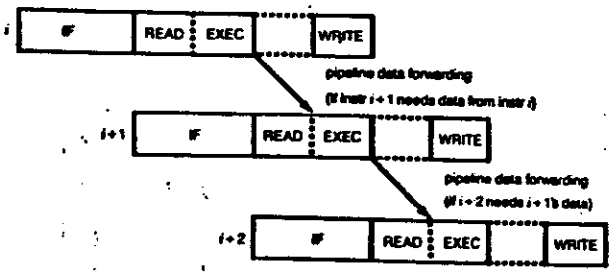
INSTRUCTION PIPELINE OF MIPS

5-stages ? : It is difficult to say how many stages because there are half- and full-cycle stages



INSTRUCTION PIPELINE OF RISC II

3-stages



RISC II (Berkeley)

True 32 bits computer

8 frames each containing 16 general purpose registers. REGISTER WINDOWS

8 Mhz clock (500 ns cycle time)

4 µm single metal nMOS VLSI chip

3 pipeline stages.

pipeline data forwarding to prevent pipeline instruction dependencies

fast procedure calling mechanism using register frames

OS protection: user- and priviliged mode

virtual memory support: paging

WINDOWS OF REGISTERS:

- A new window of registers is automatically allocated to each procedure call
- Windows overlap so that some registers are simultaneously part of two windows.
- The ovelapping part is used for parameter passing

RISC II MULTI-WINDOW REGISTER FILE

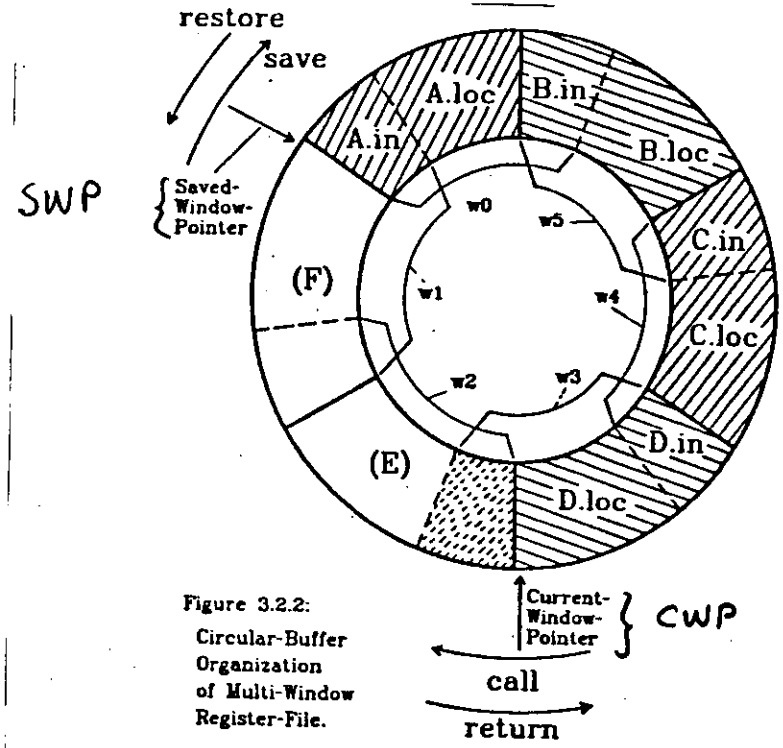


Figure 3.2.2:
Circular-Buffer Organization of Multi-Window Register-File.

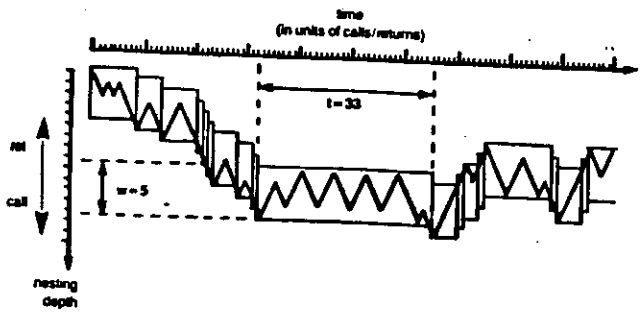


Table 3. RISC processors datapath features.

Processor	Pipeline Stages	Register Set	Delayed Branch	Execution Units
Accel	3	48 32-bit register windows with overlap of 16, 512 windows	Yes, with delay of 2	Separate integer and floating point units
ARM	3	32 32-bit, 25 for user	No, compiler rescheduling	1 plus fetch incrementer
AMD29300	2	Variable	Yes, plus nonsequencer	2: execute and fetch/decode
CAP	?	16 per processor	Send both options, do according to data mask	Scalar unit plus vector data and address units
Clipper	3	16 user, 16 special, 16 supervisor, all 32-bit	?	2: integer and floating point, all on chip
CRISP	3	32 32-bit, as a stack cache	Branch folding	2: prefetch and decode, and execution units
Dragon	?	?	Static branch prediction	2: instruction fetch, execute
FAIM-1	2	Stack	Yes	2: evaluation and switching processors
MIPS	5 (see Figure 1)	32 general purpose, 1 PC, 2 arithmetic, no windows	Yes, with reordering	2: CPU and system coprocessor. Separate data and address units
Pyramid	3	528 32-bit in 16 windows of 64	Yes	Separate instruction fetch and execution units
Ridge 32	4	16 32-bit with overlap windows	Branch predictions	2: instruction fetch, execute
ROMP	3	16 32-bit, 10 system, 4-port register file	Yes, branch with execute instruction	1
Spectrum	5	32 32-bit general purpose	Yes, with reorganization	1 CPU plus coprocessors
Transputer	?	6 32-bit	?	2: execute, I/O
Whetstone	3	4	?	3: arithmetic, address, I/O
MIPS-X	5	32 32-bit general purpose	Yes, with squash instruction, delay of 2	Separate execution and PC
SOAR	3	72 32-bit dual port with 32 register windows, using overlap of 8	Yes, with delay of 1 cycle. Must do type checking.	1
SPUR	4	32 32-bit registers in each window, using overlap of 6 and 10 global	Yes, with cancel compare instruction	1, but instruction and data fetches concurrently
CDC GaAs	6	16 32-bit general purpose	Yes, with reorganization and delay of 2	1
McD GaAs	4	16 32-bit general purpose, 16 32-bit special	Yes, with reorganization and delay of 1	1
RCA GaAs	5 + 2 waits	16 with variable size windows or background loading	Yes, with ignore instruction and delay of 2	Separate execution and PC units

Processor	Number of Instructions	Decoder	Instruction Length
Accel	142 integer, 126 floating point	Microcoded	90% are 16 bits, 10% are 32 or 80 bits
ARM	?	Programmable logic array	All 32 bits
AMD29000	Variable	Hardwired or microcoded	32 bits
CAP	33 general, 7 scientific, 9 SIMD support	Hardwired; vector units are either 16 or 32 bits	16 or 32 bits
Clipper	101 general, 61 macroinstructions	Hardwired with 2048-word macroinstruction ROM	Multiples of 16 bits
CRISP	33	Decoder unit	16, 64, and 96 bits
Dragon	approx. 150	Programmable logic array	8, 16, 24, or 40 bits
FAIM-1	64	Finite state machine	?
MIPS	?	Hardwired	All 32 bits
Pyramid	90	Microcoded	32, 64, or 96 bits
Ridge 32	170	Microcoded	16, 32, and 48 bits
ROMP	112	Hardwired with 256 words of microcode	16 or 32 packed to 32 bits
Spectrum	140	PLA with millicode	All 32 bits
Transputer	16	Programmable logic array	All 8 bits
Whetstone	18 basic, 181 total	Basic instructions have hardware decode	16 bits
MIPS-X	< 32	Hardwired, distributed	All 32 bits
SOAR	20	Hardwired	8, 16, and 24 bits
SPUR	28 general, 10 Lisp support, 25 floating point	Hardwired CPU and FPU	All 32 bits
CDC GaAs	29 CPU, 31 FCOP, 6 MMU	Hardwired	Multiple; depends on execution unit
McD GaAs	< 64	Hardwired	All 32 bits
RCA GaAs	< 64	Hardwired	32 and 64 bits

Processor	Cycle Time/Clock Rate	Instruction Rate (MIPS)
Accel	100 ns	3.2 MIPS
ARM	8 MHz	3-4 MIPS
AMD29000	125 ns	4-5 MIPS
CAP	I: 10 MHz, II: 25 MHz	12.5 MIPS peak, scalar unit
Clipper	33 MHz	5 MIPS
CRISP	16 MHz	> 10 MIPS
Dragon	10 MHz	5 MIPS per CPU
FAIM-1	?	?
MIPS	16.6 MHz	8 MIPS
Pyramid	125 ns	2-4 MIPS
Ridge 32	125 ns	1-4 MIPS
ROMP	170 ns	2 MIPS
Spectrum	30 MHz	10.8 MIPS
Transputer	50 ns	10 MIPS
Whetstone	50 ns	5-13.3 MIPS
MIPS-X	20 MHz	> 10 MIPS
SOAR	400 ns	See text
SPUR	150 ns	?
CDC GaAs	5 ns	99 MIPS
McD GaAs	10 ns	100 MIPS
RCA GaAs	200 MHz	200 MIPS peak

Table 1. Description of RISC processors.

Processor	Manufacturer	Configuration	Technology	Application
Accel	Celerity Computing	Single-board CPU and workstation	Uses NCR 32000 MOS chip set	Multiuser scientific and general purpose with Unix support
ARM	Acorn Computers	Microprocessor and single-board computer	2 to 3 μ MOS, 25K transistors	Workstation for HLL programming and real time AI
AMD29300	Advanced Micro Devices	Chip set to construct custom computer	Bipolar LSI with ECL internal and TTL interface	Construct target architecture without custom VLSI
CAP	IT&T Advanced Technology Center	Slave multiprocessor array	1.2 to 3 μ CMOS, 120K to 600K transistors	Signal processing, image processing, and scientific calculations
Clipper	Fairchild	Single-board computer	3 chips of 2 μ CMOS, total of 836K transistors	Scientific programming in a Unix environment
CRISP	AT&T	Microprocessor	1.75 μ CMOS, 172K transistors	General purpose
Dragon	Xerox PARC	Multiprocessor workstation	2 μ CMOS	Symbolic processing, AI
FAIM-1	Schlumberger, Palo Alto	Multiprocessor workstation	Custom VLSI CMOS	Symbolic processing, AI
MIPS	MIPS Computer Systems	Microprocessor, chip set, and minicomputer	2 μ CMOS, 100K transistors	General purpose programming with Unix support
Pyramid 90X	Pyramid Technology	Superminicomputer	Schottky TTL	Scientific programming and graphics with Unix support
Ridge 32	Ridge Computers	Workstation and superminicomputer	STTL and MOS VLSI in multiple chips	Scientific programming and graphics with Unix support
ROMP	IBM	Microprocessor chip set	2 chips of 1.8 μ NMOS with a total of 111K transistors	Scientific and graphics workstation with Unix support
Spectrum (HP Precision Architecture)	Hewlett-Packard	CPU family	1.5 μ NMOS, 115K transistors	Scientific, business, and instrumentation computing with Unix support
Transputer	INMOS	Microprocessor family	2 μ NMOS, 250K transistors	Multiprocessor
Whetstone I, Whetstone II	Integrated Digital Products	Single-board computer, one-chip CPU	Bipolar VLSI with ECL internal and TTL interface	Plug-in replacement general purpose computer
MIPS-X	Stanford University	Microprocessor chip set	2 μ CMOS	Updated MIPS, multiprocessing
SOAR	University of California, Berkeley	Microprocessor and processor board	NMOS, 25K transistors	Smalltalk-80 programming system workstation
SPUR	University of California, Berkeley	Multiprocessor workstation	3 chips in 2 μ CMOS	Parallel processing research in Lisp environment
CDC GaAs	Control Data	Microprocessor chip set	3 chips of 10K gates each in HIIL GaAs	Embedded computers for signal processing
McD GaAs	McDonnell Douglas	Microprocessor chip set	E-JFET GaAs, 41K transistors in 2 chips	Embedded computers for signal processing
RCA GaAs	RCA and Purdue University	Microprocessor chip set	ED-MESFET GaAs	Embedded computers for signal processing

RISC vs. CISC CONTROVERCY

1. What differentiates a RISC from a CISC?
   - Many of the features now seen in RISCs have been extensively used in CISCs.
   - Some of the features, such as pipelide datapath, caching and register windowing, are often viewed as attributes of a RISC design
   - RISC has a coherent design philosophy
   - RISC were originally targeted for a specific application, CISC are designed for a broad range of applications. Now RISC —application→ CISC
   - Instruction set size and complexity is the major feature of RISCy designs.

2. How does one make reasonable and useful performance measurements for RISC vs. CISC comparisons?
   - Popular performance measurements, such as MIPS, are of questionable value when comparing RISC performance with CISC performance
   - Typically, the effects of Operating system overhead, compiler optimization and multiple register sets are not properly considered.
   - Benchmarks related to number of application transactions per second are more meaningful.

CONCLUSIONS

- VLSI technology bridges the gap between CISC — RISC
- CISCs are better suited for
  - general purpose applications
  - for uniprocessor-based computers
- RISCs are always to be out first under a new technology
- RISCs are suitable as building blocks for
  - high performance workstations
  - parallel tightly coupled systems

FASTER CIRCUIT TECHNOGY

- THE LEAST RISKY APPROACH
  RETAINS OLD PROGRAMMING AND ARCHITECTURE

HOWEVER,

- TAKES ADVANTAGE ONLY OF THE SPEED NOT OF THE DENSITY (VLSI)
- SPEED ALONE CANNOT SATISFY OUR COMPUTATIONAL NEEDS

VECTORIZING & OPTIMIZING COMPILERS

- SOUND TECHNIQUE PROVEN IN THE FIELD
- VECTOR MACHINES (CYBER-205 CRAY-XMP FUJITSU VP-200 (DATA)
- PIPELINING : THE BASIC PRINCIPLE

HOWEVER,

- SCALAR OPERATIONS DISTURB THE PIPELINE
- VECTORIZATION IS NOT SUCCESSFUL TO EVERY PROBLEM
- THE LEVEL OF PIPELINING HAS REACHED ITS LIMIT (NO MORE STAGES).
- NO LONGER A COST/EFFECTIVE APPROACH

PARALLEL ALGORITHMS & PARALLEL LANGUAGES

PARALLEL PROCESSORS

- PARALLELISM IS THE RIGHT APPROACH TO SOLVE OUR PROBLEMS IN-TIME
- PROBLEMS PROVIDE SUFFICIENT PARALLELISM
- VLSI HAS REMOVED THE COST BARRIER
- MULTIPROCESSOR SYSTEMS ALREADY VERY SUCCESSFUL IN SEVERAL CASES

MIMD
- ~~HEP~~
- COSMIC CUBE, NCUBE
- ULTRA

SIMD
- ILLIAC IV
- ICL DAP
- CONNECTION MACHINE
- ARRAY PROCESSORS

HOWEVER,

- NO AGREEMENT ON A SINGLE GENERAL PURPOSE DESIGN
- OVER 2000 CONCEIVABLE DESIGNS. MANY QUESTIONS TO ANSWER.
- SERIOUS PROBLEMS TO BE SOLVED
  - PARTITIONING
  - SCHEDULING
  - SYNCHRONIZATION
  - COMMUNICATION

APPLICATIONS & ALGORITHMS
WHICH PROVIDE SUFFICIENT PARALLELISM

- LARGE SCIENTIFIC & ENGINEERING CALC.s
  - FLOW DYNAMICS
  - PARTICLE BEHAVIOR
  - WEATHER CODES
  - PARTIAL DIFF. EQNS
  - SEISMIC PROCESSING
- VLSI DESIGN AUTOMATION
  - SIMULATION
  - PLACEMENT, ROUTING, WIRING
  - LOGIC SYNTHESIS - SILICON COMPILATION
  - TESTGEN, FAULT SIMULATION
  - TIMING ANALYSIS
- DATA BASES
- NEAR-TERM A.I.
  - EXPERT/PRODUCTION SYSTEMS
  - PARALEL PROLOG/LOGIC PROGR
  - PARALLEL SEARCH OF STATE SPACE
- LONG-TERM A.I.
  - PERCEPTION/PLANNING SYSTEMS
  - VISUAL PERCEPTION/RECOGNITION
  - NATURAL LANGUAGE PROCESSING
  - ROBOTICS
  - LEARNING

PARALLEL PROCESSORS

CHARACTERISTICS - OBJECTIVES

- A LARGE NUMBER OF FEW DIFFERENT TYPES OF COMPUTING ELEMENTS COMMUNICATING OVER AN INTERCONNECTION NETWORK
- CONCEPTUALLY EXPANDABLE AT A COST NOT MUCH GREATER THAN LINEAR AND ACHIEVING A SPEED-UP NOT MUCH LOWER THAN LINEAR
- USED TO SOLVE A SINGLE PROBLEM AT A TIME

SOME FUNDAMENTAL QUESTIONS

- COLLECTION OF SERIAL PROCESSORS
  - How MANY ? (10, 100, 10^3, 10^6)
  - How BIG ? (1 bit → main frame)
  - WHAT KIND ? (INSTRUCTION SET ?)
  - How MUCH MEMORY ?
  - How MUCH I/O ?
- WHICH CAN COMMUNICATE ...
  - INTERCONNECTION NETWORK ?
  - MEANS OF COMMUNICATION ?
  - MEANS OF SYNCHRONIZATION ?
- AND COOPERATE ...
  - SIMD / MIMD ?
  - GRAIN SIZE ?
  - OPERATING SYSTEM ?
- TO SOLVE BIG PROBLEMS FAST
  - WHICH PROBLEMS ? - APPLICATIONS
  - COMPUTATIONAL MODEL ?
  - PROGRAMMER'S VIEW ?
    - HOW IS PROBLEM DECOMPOSED ?
    - HOW TO SPECIFY CONCURRENT EXECUTION ?
    - WHO DOES IT !
    - LANGUAGES ?
    - FEATURES ?
    - COMPILERS ?
  - WHAT ABOUT RAS (RELIABILITY AVAILABILITY SERVICEABILITY)

THE PARTITIONING PROBLEM

"PARTITION A PROGRAM INTO TASKS, EACH TASK REPRESENTED BY A NODE IN A GRAPH"

- OBJECTIVES : MAXIMIZE PARALLELISM & MINIMIZE OVERHEAD
  - ↓
  - SCHEDULING
  - SYNCHRONIZATION
  - COMMUNICATION
- FACT :

  THE FINER THE GRAIN OF PARTITIONING
  THE GREATER THE PARALLELISM EXPLOITED
  BUT
  THE HIGHER THE OVERHEAD OF SYNC. & COMM.

- PARTITIONING RESOLVES INTO

  PARALLELISM DETECTION
  &
  CLUSTERING INTO NODES

  BY USERS (VIA LANGUAGES) - OCCAM
  BY COMPILERS - PARAFRASE

- A NODE MAY BE =
  - A SINGLE INSTRUCTION
  - A VECTOR INSTRUCTION
  - A SUBROUTINE / PROCEDURE
  - A PROCESS

THE SCHEDULING PROBLEM

"ASSIGN EACH NODE TO ONE OR MORE PROCESSING UNITS FOR EXECUTION"

- SCHEDULING MAY BE
  - STATIC
    - BY THE USER VIA LANGUAGE OR
    - BY THE COMPILER

    LOWER OVERHEAD
    LOWER UTILIZATION OF MACHINE
  - DYNAMIC BY THE MACHINE AT RUN TIME

    HIGHER OVERHEAD
    HIGHER UTILIZATION OF MACHINE
    - CENTRALIZED SCHEDULING

      ITS OVERHEAD MAY BECOME THE BOTTLENECK
    - DISTRIBUTED SCHEDULING (SELF-SCHEDULING)

      BETTER LOAD-BALANCING

THE SYNCHRONIZATION PROBLEM

"ASSURE AN ORDER OF EXECUTION THAT LEADS TO CORRECT RESULTS"

- THE PROBLEM ARISES WHEN ACCESSING SHARED DATA.
- HARDWARE SOLUTIONS (FOR SHARED VARIABLES)

  IBM • TEST & SET INSTRUCTION
  HEP • FULL/EMPTY BIT FOR EACH MEMORY LOCATION
  ULTRA • FETCH & ADD INSTRUCTION FOR EACH MEMORY MODULE

  CEDAR
  - COUNTER METHOD
  - BIT-MAP

  FINER SYNCHRONIZATION
  HIGHER CONCURRENCY AMONG NODES

- SOFTWARE SOLUTION
  - MESSAGE-PASSING BETWEEN NODE PROCESSES

    NO SHARED DATA

    SLOWER METHOD

THE COMMUNICATION PROBLEM

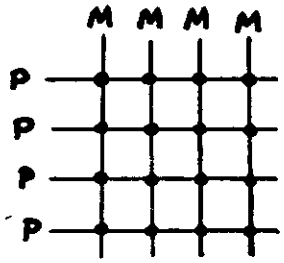
"MATCH PROCESSORS—MEMORY SPEED"

ISSUES INVOLVED

- TOPOLOGY
- COMPLEXITY / COST
- GENERALITY
- SWITCHING METHOD
  - CIRCUIT SWITCHING
  - PACKET SWITCHING
- RECONFIGURATION

IT IS COMING INTO THE SCENE

NETWORK TOPOLOGIES

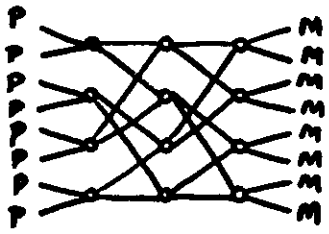


CROSSBAR (S-1)

ANY P TALKS TO ANY M AT ANY TIME

Complexity = $O(n^2)$ Delay = $O(1)$

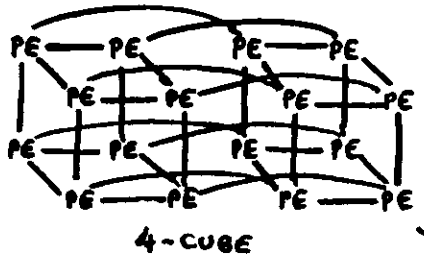
THE BEST COMMUNICATION BUT AND THE MOST EXPENSIVE



SHUFFLE (Ω) (ULTRA, CEDAR)

ANY P CAN TALK TO ANY M MOST OF THE TIME

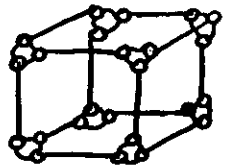
Complexity = $O(n \cdot \log n)$ Delay = $O(\log n)$



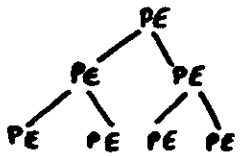
'Log-n' CUBE (COSMIC CUBE)

EACH PE CONNECTED TO LogN PE'S

Complexity = $O(n \log n)$ Delay = $O(\log n)$

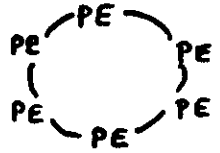


CUBE CONNECTED CYCLES

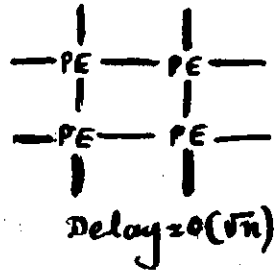


TREE

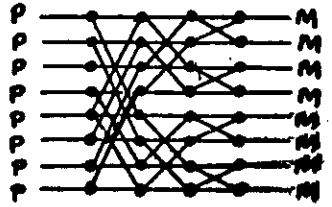
Delay = $O(\log n)$



RING / BUS



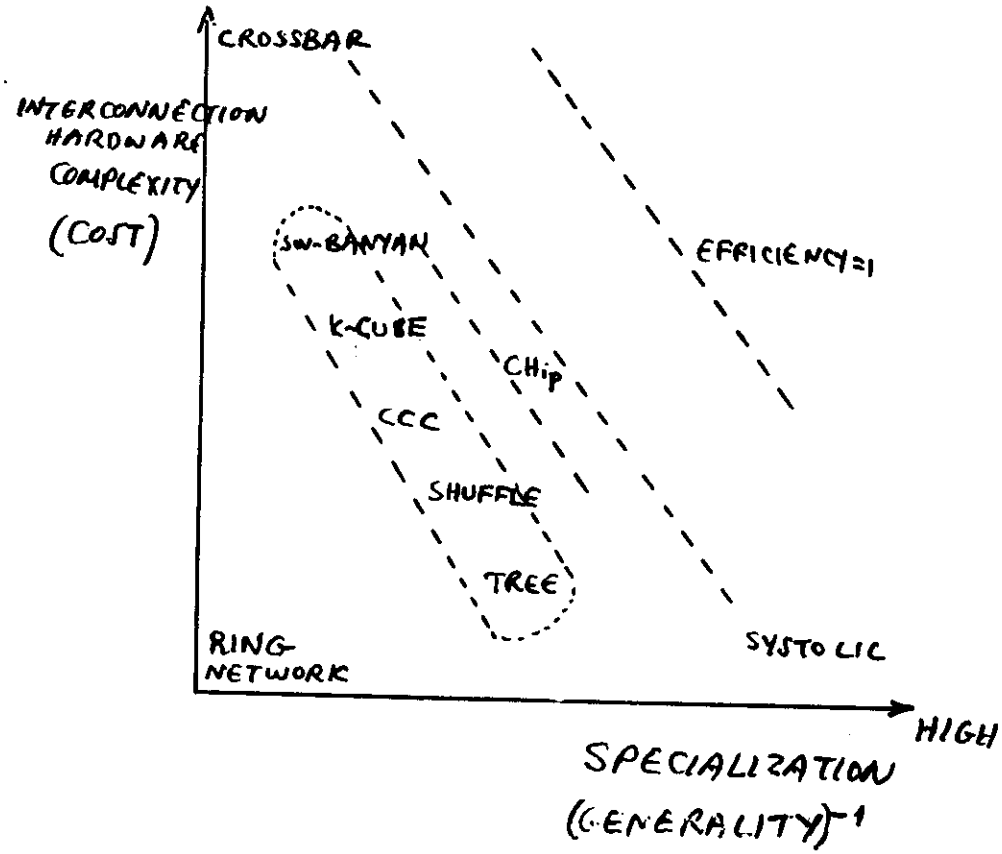
MESH



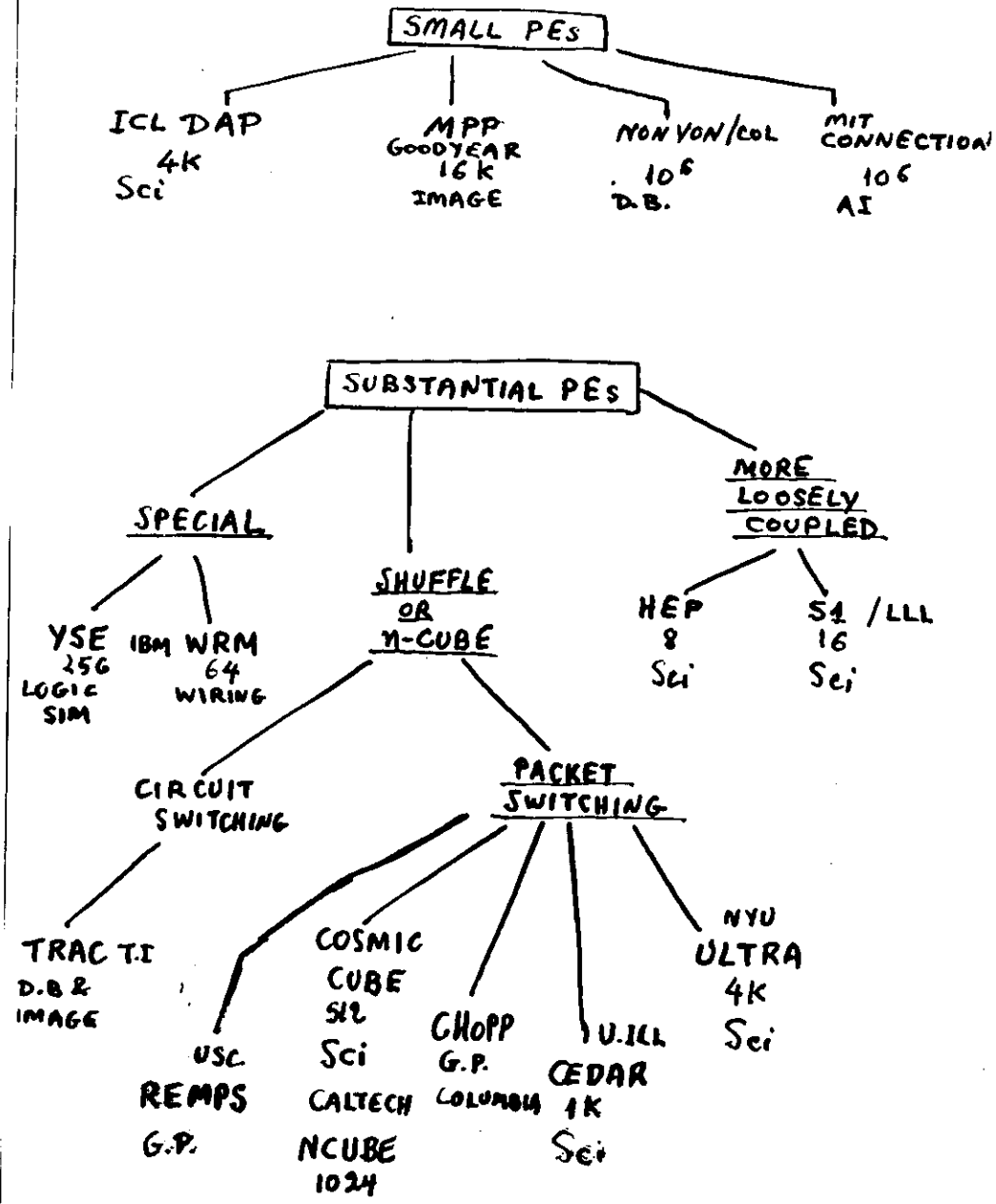
SW-BANYAN

Delay = $O(\log n)$

NETWORK COMPLEXITY VS GENERALITY



SPECTRUM OF MULTIPROCESSORS



CONVENTIONAL PARALLEL PROCESSOR ARCHITECTURES

- SIMD CLASS (REGULAR PARALLELISM)

  ASSOCIATIVE PROCESSORS (WILL NOT COVER)

  ARRAY PROCESSORS

- MIMD CLASS (ANY PARALLELISM ?)

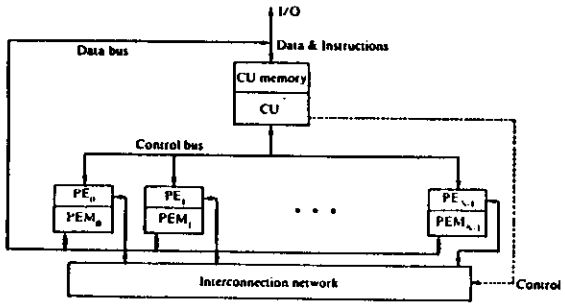
- SIMD/MIMD COMBINATION

SIMD OVERVIEW

- A set of PEs that operate in lock-step mode executing the same instruction stream each on different data streams
- The instruction stream is broadcast to all PE's by one control unit
- The data streams either are preloaded into local to PE's memories, or are in common shared memory modules
- An interconnection network is used to route DATA (& INTERMEDIATE RESULTS) among the PE's (data routing)
- Some PEs may be inactive during the execution of an instruction (masking)
- Data routing and masking are the key-factors that influence the performance of the array processors
- Basic configurations:
  - ILLIAC IV type
  - BSP type

ILLIAC IV type configuration:

NO GLOBAL MEMORY

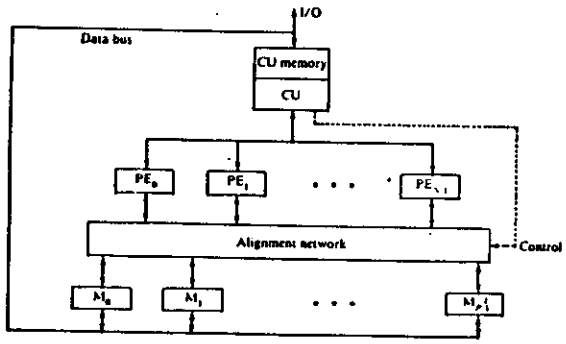


(a) Configuration I (Illiac IV)

- A single program control unit (CU)
- N processing elements (PEs)
- PE = processor + local memory
- Inter-PE interconnection network
- The CU broadcasts the same instruction
- CU has:
  - A global index register I
  - A distributed Masking Register M<0:N-1>
  - A distributed Status Reg. S<0:N-1>
- Each PEi has a local index reg. Ii in order to modify the address broadcast by the CU and access thus different vector elements.

BSP type configuration

GLOBAL MEMORY



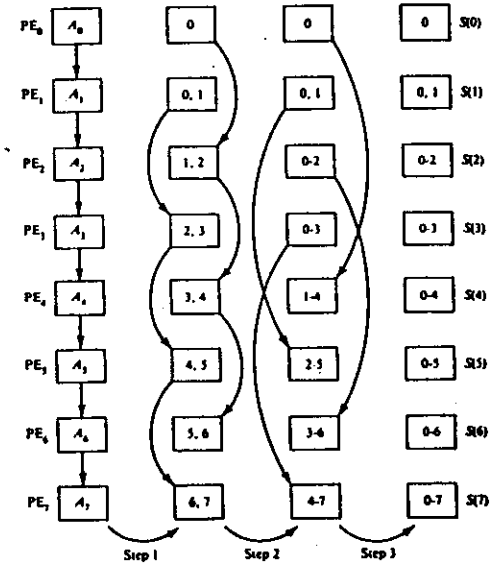
(b) Configuration II (BSP)

Figure 5.1 Architectural configurations of SIMD array processors.

- A single control unit (CU)
- N processing elements (PEs)
- No local memory to PEs
- Instead M shared memory modules
- Alignment interconnection network between PEs ↔ memory modules
- It allows conflict-free accesses of the shared memory modules by as many PEs as possible.

VECTOR PROCESSING & DATA ROUTING USING ARRAY PROCESSORS

- The physical length of a vector is determined by the number of PEs
- The CU performs segmentation of long vectors into vector loops, sets the global Base address, and the offset increment.
- Distributing vector elements is crucial to the efficient utilization of an array processor.



- Example:

compute $S_k = \sum_{i=0}^{k} A_i \quad k = 0,1,2,\ldots,7$

all partial sums

DESIGN OPTIONS OF INTERCONNECTION NETWORKS FOR ARRAY PROCESSORS

- Operation mode < [Synchronous] / Asynchronous
- Control strategy < [Centralized] / Distributed
- Switching method < [Circuit switching] / Packet switching

(Control strategy, Switching method: ← CONNECTION MACHINE)

- Network Topology < [Static] / [Dynamic]

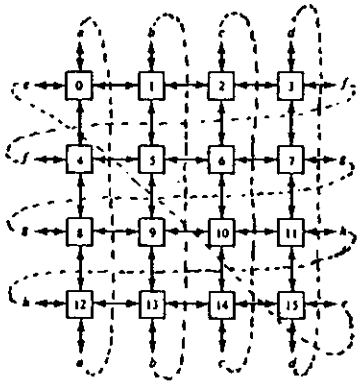
[×] : usual choice for SIMD machines however, see the CONNECTION MACHINE

- Popular Interconnection Networks for SIMDs
  - the MESH-connected network (ILLIAC IV)
  - the N-CUBE network (Connection machine)

The MESH-connected network topology (ILLIAC)

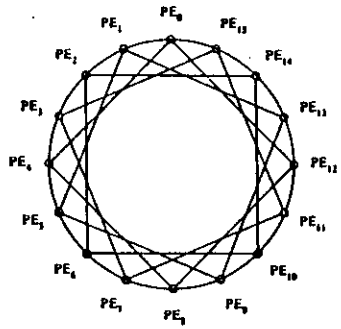
- each PE communicates with its neighbours mod N
- N is usually a perfect square
- network efficiency = $O(\sqrt{N})$, Delay = $O(\sqrt{N})$
- The same routing function is applied to all PEs (R_{+1}, R_{-1}, R_{+r}, R_{-r})

N=16



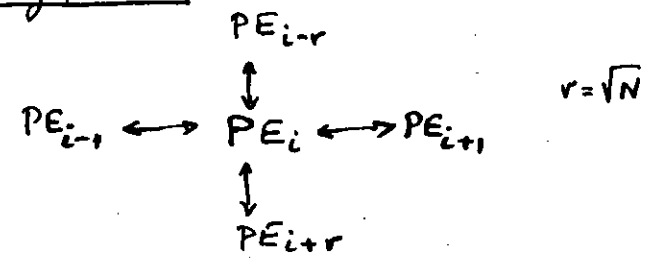
(a) The mesh connections

Redrawn



(b) The mesh redrawn

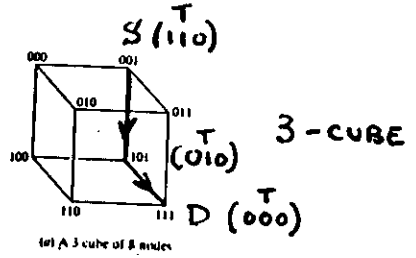
Routing function:



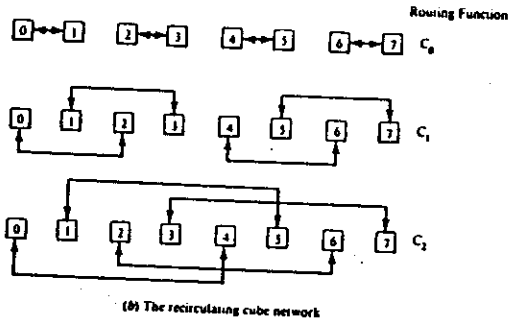
$r = \sqrt{N}$

THE RECIRCULATING N-CUBE (HYPERCUBE) - CONNECTION MACHINE

- An N-dimentional cube connecting 2^N PEs
- Each PE is directly connected to N neighbours
- Each PE corresponds to an N-bit number
- Routing tag T = Source # ⊕ Destination #
- Delay = $O(\log n)$, Network efficiency = $O(n \log n)$



(a) A 3-cube of 8 nodes

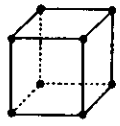


(b) The recirculating cube network

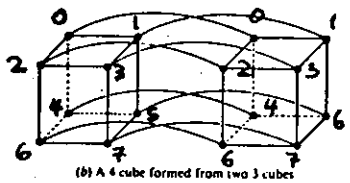
S = 001
D = 111
T = S ⊕ D = 110

- Deadlock free routing:
  Follow path that diminishes the T dimension

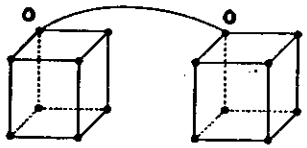
RECURSIVE DESIGN OF THE N-CUBE



(a) A 3 cube



(b) A 4 cube formed from two 3 cubes



(c) The 4 cube showing only one of eight fourth-dimension connections

Figure 5.19 The construction of $(n + 1)$-cube from two n-cubes. (Courtesy of Thomas, 1981.)

An $(n+1)$-cube is constructed from two n-cubes by linking together their corresponding vertices by 2^n extra edges.

ICL-DAP DISTRIBUTED ARRAY PROCESSOR

- 64x64 MESH ARRAY OF SINGLE-BIT PEs
- 4096-BIT STORE PER ONE PE
- DAP INTEGRATED INTO HOST's MEMORY
- VECTOR MODE ON 128 VERTICAL WORDS
- SCALAR MODE ON 64 HORIZONTAL WORDS
- 12-30 MFLOPS 32-bit.
- PROGRAMMED IN DAP-FORTRAN

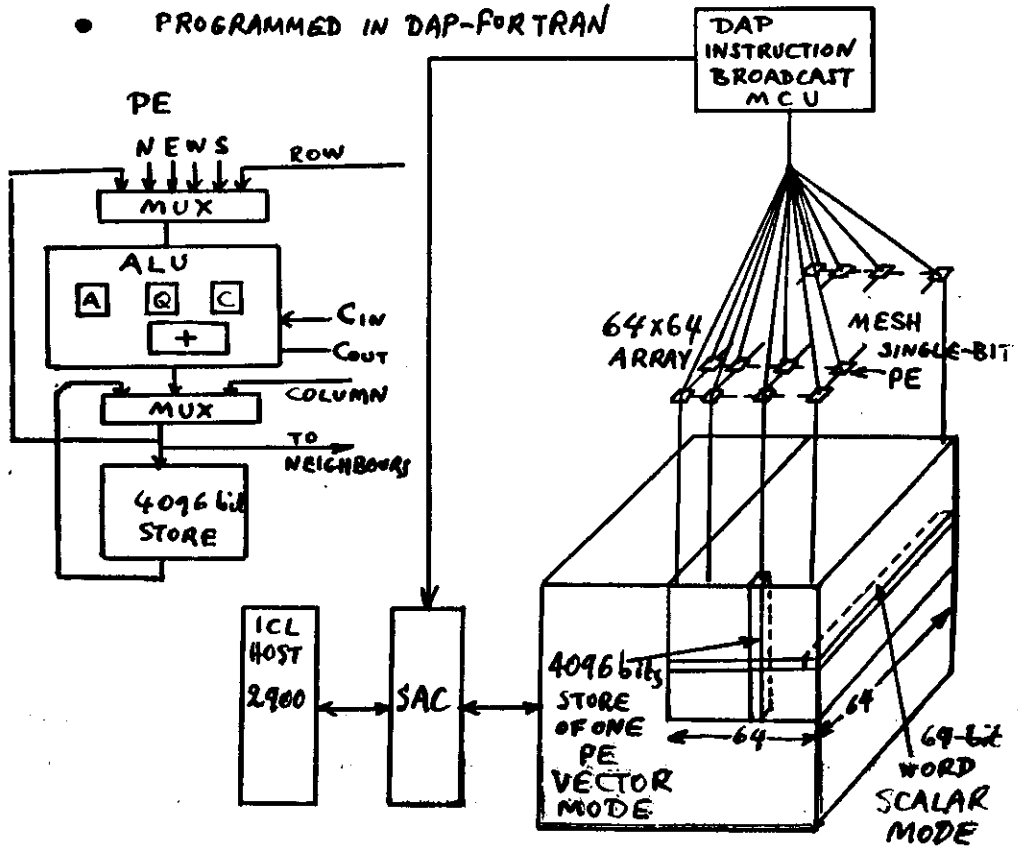
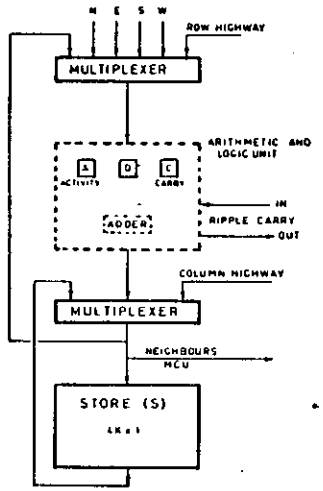


Fig. 13

1-bit processor

3 registers : Activity, Q, Carry

A is used for masking



- DAP was used is several scientific areas
  - Monte Carlo
  - QCD
  - Molecular dynamics
- A new version has been announced

THE CONNECTION MACHINE

GOALS:

- Design of a thinking machine
- Make optimal use of current & future VLSI
- Change the traditional partitioning between memory vs. processing
- Use many (16) simple PEs on one chip

MACHINE MODEL:

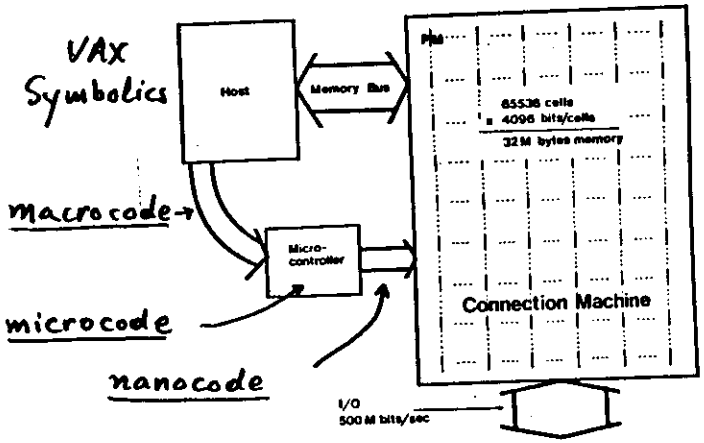
- Fine grain machine
- SIMD machine with one-bit PEs
- Communication paths one-bit wide
- $65536 = 2^{16}$ PEs (4×16384 PEs)
- Connected via a 12-CUBE network (4096 NODES) with packet-switching
- Also a 1-bit wide global OR network.

CHIP ARCHITECTURE:

- One chip contains:
  - 16 1-bit PEs
  - one control unit
  - one routing unit
- On chip the 16 PEs are connected in a 4x4 grid pattern
- Each PE communicates with a 4096 bit static RAM

BLOCK DIAGRAM OF THE CONNECTION MACHINE

- It appears to the host as a smart memory.
- Machine cycle (3 microcontroller states) ≈ 750 nsecs
- already delivered
- $3000 per MIP



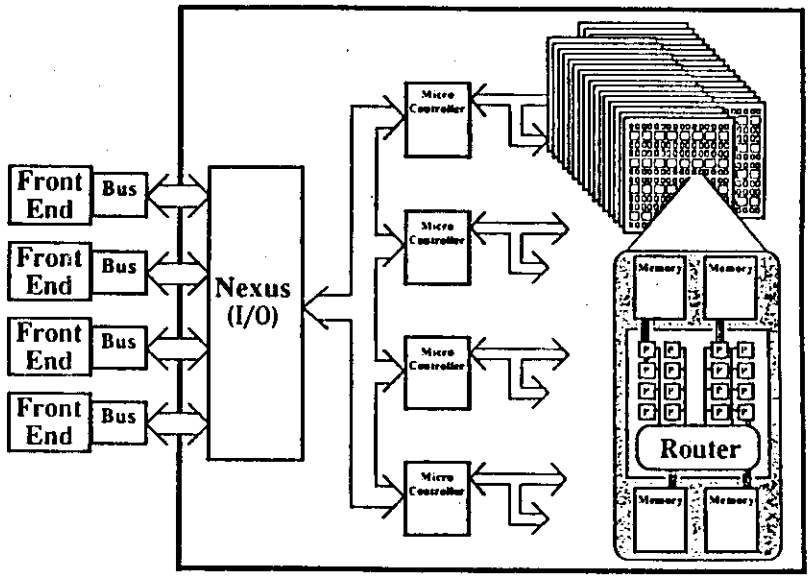
PERFORMANCE:

peak rate ≈ 2000 MIPS (32-bit)

communication bandwidth ≈ 3.2×10^7 b/s – 1.0×10^9 b/s

Connection machine algorithms
use O(N) processors
to solve problems of size N
typically in O(logN) time

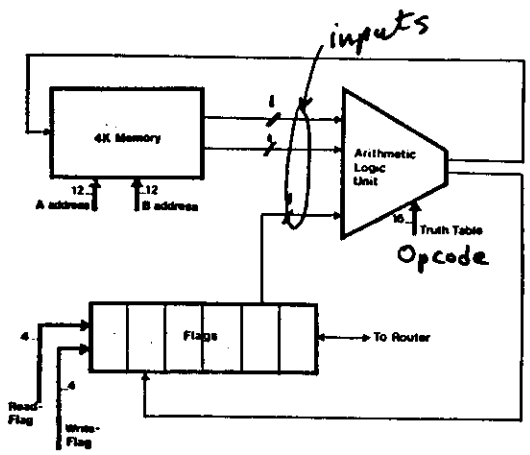
ANOTHER VIEW OF THE CONNECTION MACHINE



- The 2^{16} PEs are in fact partitioned into 4 groups of 2^{14} (16.384) PEs, each group with its own microcontroller

The processing element of the CONNECTION machine

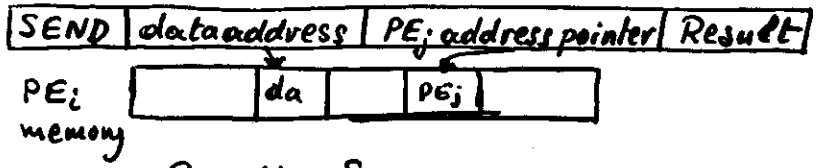
- 1-bit ALU
- performs all 256 3-input functions
- specified via 2x8-bits truth table
- 16 flags (8 general purpose, 8 special flags)



CONNECTION MACHINE PROGRAMMING CONSIDERATIONS

- The control structure of a program running on the Connection machine system is executed by the front-end in the usual way.
- The program can perform computations in the usual serial way and also issue commands to the PEs array.
- The PEs array executes commands in SIMD mode. Most SIMD instructions are executed conditionally (condition flag)
- The set of conditioned PEs is the context in which instructions are executed
- Contexts may be saved in memory and later restored.
- Also it is possible to form the union, intersection, or complement of contexts using respectively, the OR, AND and NOT operations issued to the PEs array.
- Compared to other SIMD machines, the Connection machine offers:
  - (a) general, pointer-based communication

    SEND instruction

    Result ∈ {sum, max, min, AND, OR}

  - (b) Virtual processors

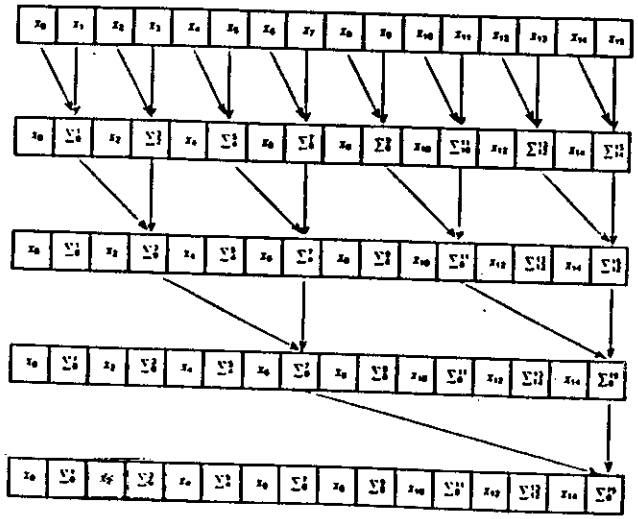
    Programs are abstracted from the details of the hardware and in particular, the number of PEs the machine has

Connection machine algorithms

(1) Sum of array of numbers $O(\log N)$

```
for j := 1 to log2 n do
  for all k in parallel do
    if ((k + 1) mod 2^j) = 0 then
      x[k] := x[k − 2^(j−1)] + x[k]
    fi
  od
od
```

time = 200 μsec for 65536 elements

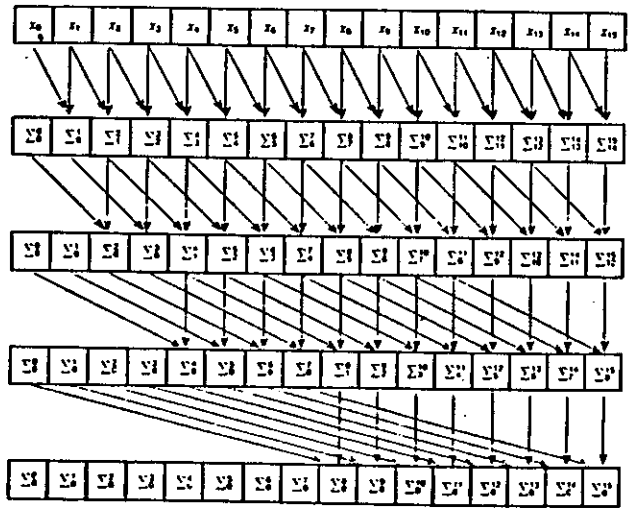


Connection machine algorithms

(2) All partial sums of an array $O(\log N)$

```
for j := 1 to log2 n do
  for all k in parallel do
    if k ≥ 2^j then
      x[k] := x[k − 2^(j−1)] + x[k]
    fi
  od
od
```

time = 200 μsec for 65536 elements



Connection machine algorithms

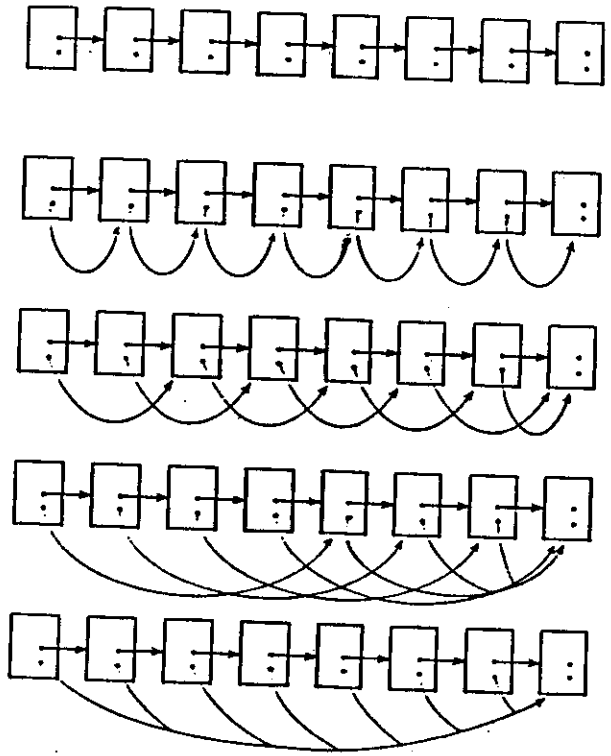
Find the end of a linked list

- Let each PE holds a cell of the linked list
- For each PE_k there is a pointer next(k) that points to the next PE in the list
- Basic idea:
  All PEs in the list get repeatedly a copy of the pointer next(next(k))
- Initially a PE points to a PE one cell apart, then to a PE two cells apart, then to four cells apart and so on until it points to the last PE in the list.
- Program

```
for all k in parallel do
  chum[k] := next[k]
  while chum[k] ≠ null and chum[chum[k]] ≠ null do
    chum[k] := chum[chum[k]]
  od
od
```

- time . $O(\log n)$ n = list length
- The same algorithm can be used to find the end of many lists in parallel in time = $O(\log n_{max})$

Illustration of the algorithm for finding the end of a linked list



Connection machine algorithms

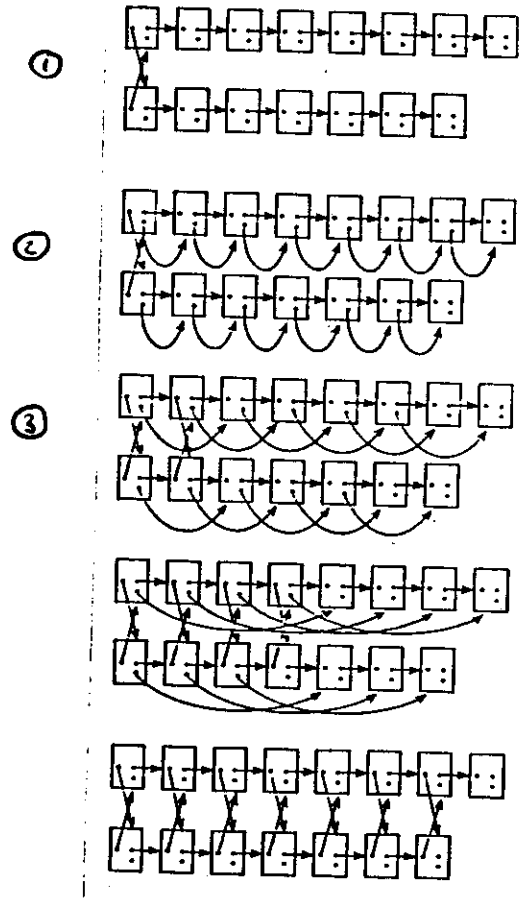
Matching up elements of two linked lists

- given two linked lists assign to each list cell a pointer to its "friend" in the other list

```
for all k in parallel do
  friend[k] := null
od
friend[list1] := list2          } → ①
friend[list2] := list1          }
for all k in parallel
  chum[k] := next[k]            ——→ ②
  while chum[k] ≠ null do
    if friend[k] ≠ null then
      friend[chum[k]] := chum[friend[k]] } → ③
    chum[k] := chum[chum[k]]
    fi
  od
od
```

- One can process many lists (or pairs of lists) in parallel
- It is possible to match up two lists of unequal length

Matching up elements of two linked lists



ARRAY PROCESSORS FOR LATTICE QCD CALCULATIONS

	#PEs		Peak rate
• APE	16 PEs Weitek		1000 MFLOPS
• COLUMBIA UNIV	256 PEs	80286/287	4000 MFLOPS
• IBM GF11	512 PEs	Weitek	11520 MFLOPS

MULTIPLE PROCESSORS/MIMD

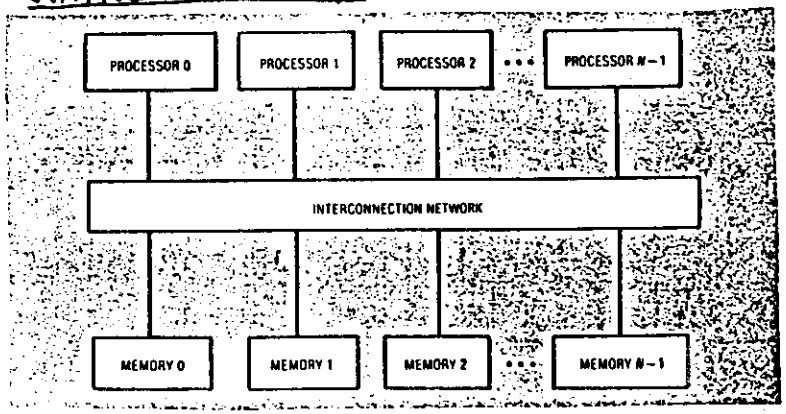
- BUILDING BLOCKS: PROCESSORS (μP's today)
  MEMORIES
  NETWORK SWITCHES
- APPLICABLE TO WIDER RANGE OF APPICATIONS
- AT THE EXPENCE OF INCREASED
  - HARDWARE COMPLEXITY
  - SOFTWARE SUPPORT
  - PROGRAMMING EFFORT
  - ALGORITHM EFFORT

- SPEED-UP FACTOR $= \dfrac{1}{\frac{1}{n}P + H + S}$

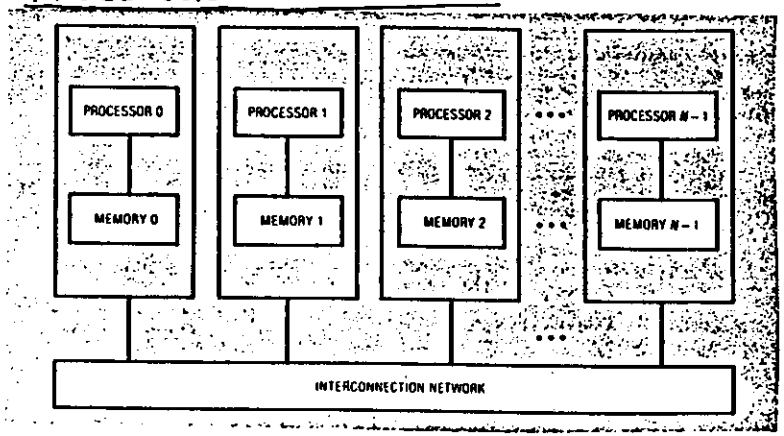
  P+S = UNIT OF WORK
- FACTORS THAT PREVENT <u>LINEAR SPEED-UP</u>
  - ALGORITHM
  - SYNCHRONIZATION
  - OVERHEAD
  - CONTENTION
  - INPUT/OUTPUT

BASIC MIMD CONFIGURATIONS

SHARED MEMORY — TIGHT COUPLING



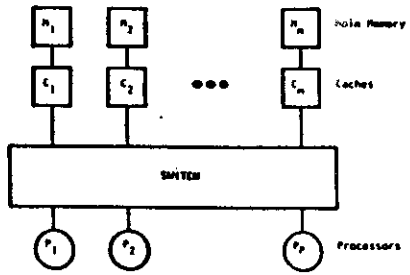
PROCESSOR-TO-PROCESSOR — LOOSE COUPLING



MESSAGE PASSING SYSTEMS

USE OF CACHE IN MIMD machines

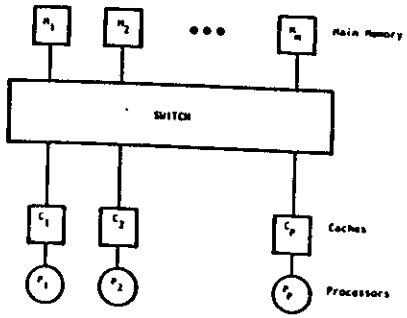
(a) CACHE IN FRONT OF MEMORY



Makes sense when the connect time via the network is much less than the memory access time (circuit switching)

(6) CACHE IN FRONT OF PROCESSOR

SEVERE CACHE COHERENCE PROBLEMS



A separate high speed bus is used to maintain cache coherence

SPECTRUM OF MIMD MACHINES

DESIGN PARAMETERS

Size of Processors

FROM : Largest Pipelined Vector Processor
(Cray X-MP2, Cray II, ETA)
TO : Single chip μPs

Number of Processors

FROM : Few Processors (4 - 30)
TO : Few hundreds (512 - 1024)

Interconnection Networks

FROM : Shared Bus
THROUGH : Optimal Logarithmic Networks
TO : Trees

MIMD machines

COMMERCIALLY AVAILABLE

Machine	#Processors	Network	MFLOPS peak
CRAY-X-MP-2	2		420
CRAY-X-MP-4	2		940
CRAY 2	4		2000
ALLIANT FX/8	8	BUS	44
SEQUENT	4-30	BUS	2.9-21
INTEL i-PSC	4-128	n-CUBE	12.
NCUBE	4-1000	n-CUBE	500
FPS T-Series	256		17000
IBM 3090/VF-400	4		432
HEP	4	Banyan	
ENCORE	2-20	u	1.5-15

UNIVERSITY/EXPERIMENTAL MACHINES

COSMIC CUBE	64	n-CUBE
ULTRA NYU	512	Ωmega
RP3	512	Ωmega
SUPERNODE		

DESIGN OPTIONS OF INTERCONNECTION NETWORKS FOR MIMD MACHINES

- Operation mode
  - Synchronous
  - [x] Asynchronous
- Control strategy
  - Centralized
  - [x] Distributed
- Switching method
  - [x] Circuit Switching
  - [x] Packet switching
- Network topology
  - [x] Static
  - [x] Dynamic

[x] Usual choice for MIMD machines

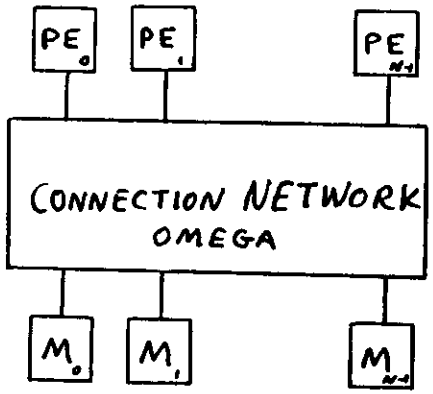
Popular networks

- Shared bus
- N-cube
- Multistage networks
  - Ωmega
  - Banyan

NYU ULTRACOMPUTER

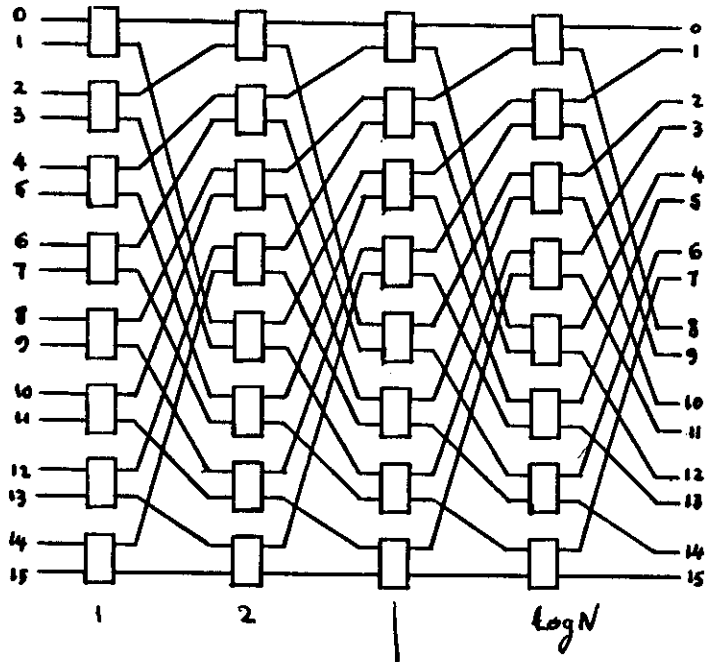
- 4096 PEs USING 1990 TECHNOLOGY
- SHARED MEMORY MODULES
- LAWRIE-TYPE (OMEGA) NETWORK
- APPROXIMATES Schwartz's PARACOMPUTER MODEL
- Repadd PROCESS CYNCHRONIZATION PRIMITIVE

Encouraging Simulation results

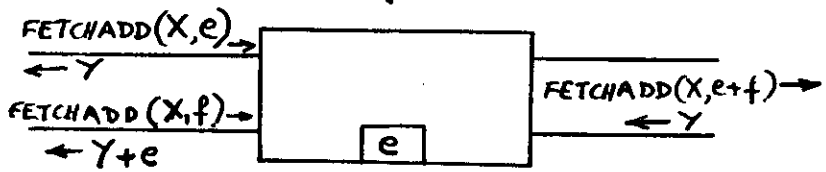


- RP3 IS A MACHINE SIMILAR TO ULTRA BUILT BY IBM
  512 801 RISC processors

LAWRIE (OMEGA) NETWORK FOR 16 PEs



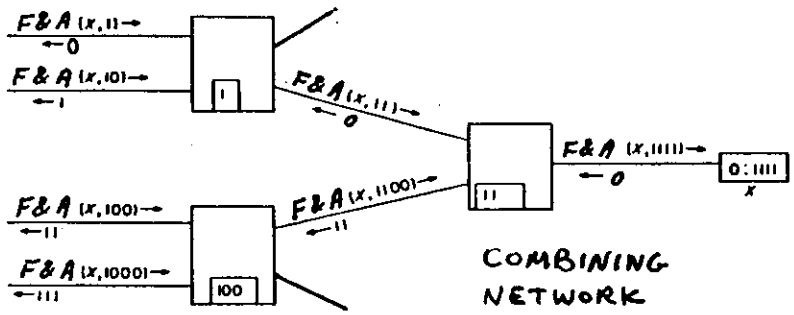
INTERPROCESSOR SYNCHRONIZATION : Fetch-Add



SIMULTANEOUS FETCHADD OPERATIONS

FETCH & ADD SYNCHRONIZATION PRIMITIVE

$S := F\&A(X,e)$ S becomes X
X becomes $X+e$



For several $S_i = F\&A(X,e_i)$

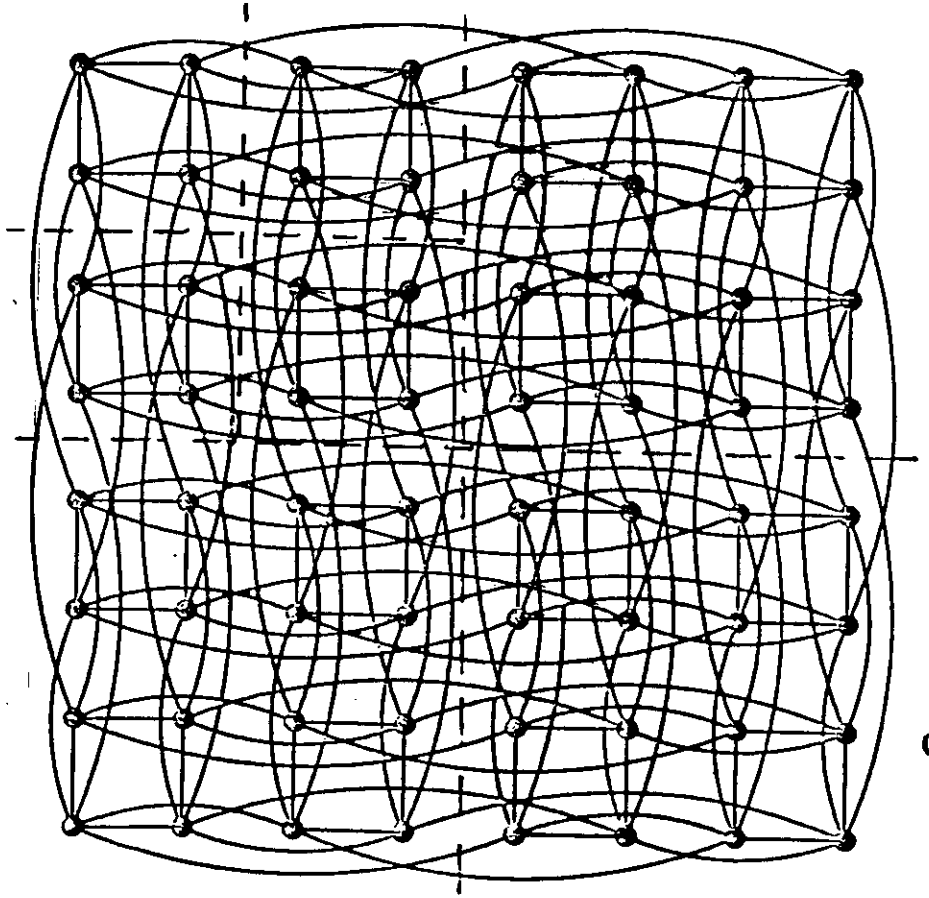
$$S_1 \leftarrow X$$
$$S_2 \leftarrow X + e_1$$
$$S_3 \leftarrow X + e_1 + e_2$$
$$\vdots$$
$$S_n \leftarrow X + (e_1 + e_2, \dots, e_{n-1})$$
$$X \leftarrow X + \sum_{i=1}^{n} e_i$$

This <u>combination</u> is implemented in the Interconnection network:

Each switch box of the IN combines Two incoming F&A that refer to the same location in memory
HOT SPOTS

COSMIC CUBE (2^N $N=2,4,6$)

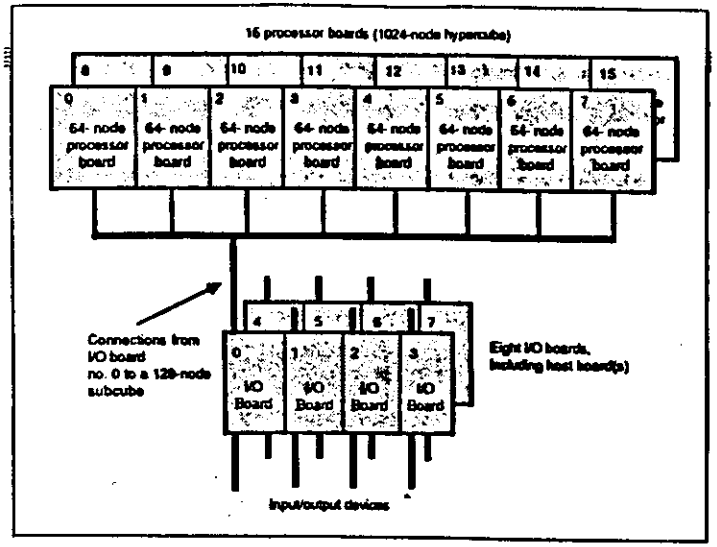
- a <u>message passing</u> system
- N-cube connection
- PE 8086/8087



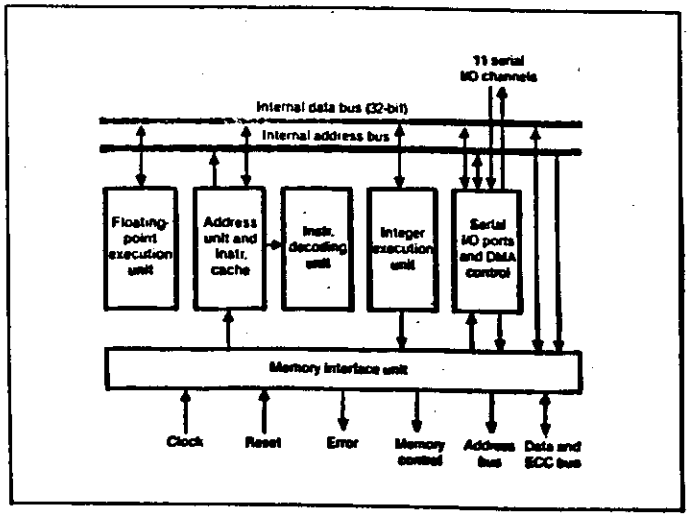
- 128 kbytes local memory
- Bidirectional 10 Mbit/sec links
- Node processes are <u>explicitly</u> distributed to the node by the programmer
- Commercial version by Intel iPSC 80286/287

NCUBE/10 SYSTEM

- up to 1024 PEs VAX-like custom chip
- peak rate 500 MFLOPS



0.5 MFLOP PE



CONCLUSIONS

- Conventional parallel processor machines is a reality

  SIMD : DAP / CONNECTION

  MIMD : shared bus systems
  n-cubes

- Multistage interconnected MIMDs still a challenge to VLSI (ULTRA, RP3)

- Programming MIMDs is major problem

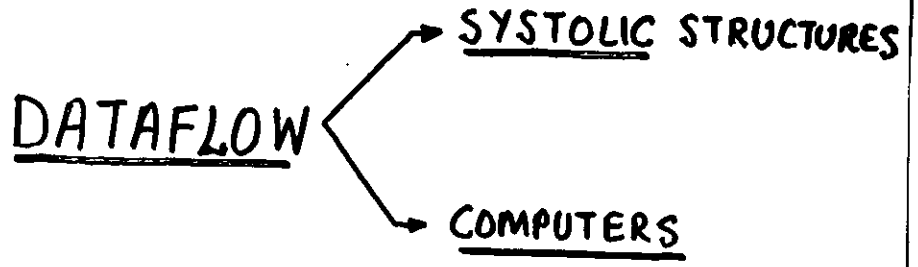
- Much easier to program SIMDs

- Can we map MIMDs → SIMDs ?

  - The burden of efficient programming is on the user
  - User must plan processing and communication
  - User must think about parallelism and rewrite sequential programs to make them run in parallel
  - No dynamic scheduling & load balancing
  - No parallelizing compilers

NOVEL COMPUTER ARCHITECTURES

SPECTRUM OF NOVEL MACHINE ARCHITECTURES



REDUCTION MACHINES

LOGIC & INFERENCE MACHINES

WHY NOVEL ARCHITECTURES

- EXPLOIT PARALLELISM IN A BETTER WAY
- UNVEIL ALL INHERENT PARALLELISM IN A MORE NATURAL WAY
- EMPLOY NEW PROGRAMMING LANGUAGES THAT INCREASE PROGRAMMER'S PRODUCTIVITY

"TO BREAK THE VON-NEUMANN BOTTLENECK"

- EXPAND THE AREAS OF APPLICATION
  - DATA BASES
  - COMBINATORIAL SEARCH < DECISION / OPTIMIZATION
  - A.I.

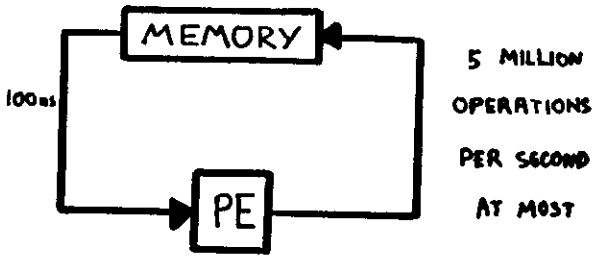
DATAFLOW SYSTOLIC STRUCTURES

- HARDWIRED OR MINIMALLY PROGRAMMED
  - "SPECIALIZED" PROCESSORS WITH
  - "FIXED" INTERCONNECT THAT
  - DIRECTLY EXECUTE A SINGLE ALGORITHM
- SYSTOLIC ARCHITECTURE BY KUNG/CMU
  A GENERAL METHODOLOGY FOR MAPPING HIGH LEVEL COMPUTATIONS INTO HARDWARE STRUCTURES
- SATISFIES VLSI CRITERIA
  - SIMPLE & REGULAR DESIGN
    BASED ON FEW TYPES OF FUNCTIONAL UNITS
  - FUNCTIONAL UNITS EASILY TESSELLATED
    USING LOCAL REGULAR INTERCONNECTIONS
  - CONCURRENCY ACHIEVED BY PIPELINING
    DATA THROUGH THE STRUCTURE,
    BALANCING COMPUTATION WITH I/O

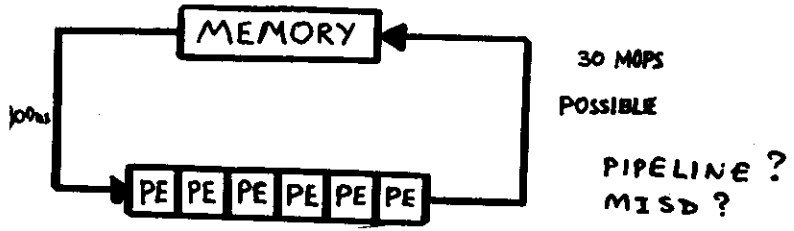
BASIC IDEA OF SYSTOLIC ARCHITECTURE

FOR COMPUTE-BOUND TASKS

INSTEAD OF:



WE HAVE:



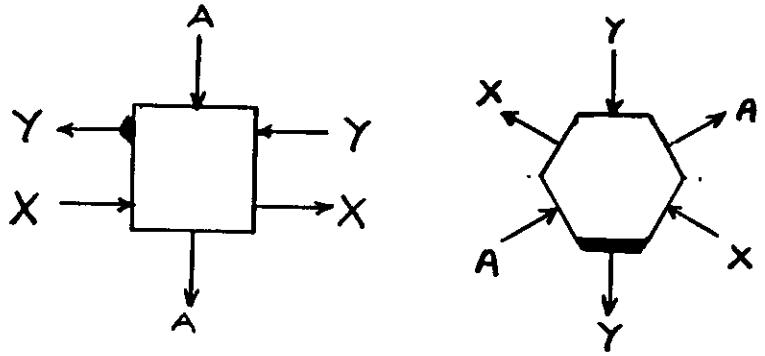
THE SYSTOLIC ARRAY

- THE VON NEWMANN BOTTLENECK IS RESOLVED BY MAKING MULTIPLE USE OF EACH x FETCHED FROM THE MEMORY
- MODULAR EXPANSIBILITY
- REGULAR DESIGN

APPICATIONS OF SYSTOLIC ARRAYS

- SIGNAL & IMAGE PROCESSING
  - FIR, IIR FILTERING & 1-D CONVOLUTION
  - 2-D CONVOLUTION & CORRELATION
  - INTERPOLATION
  - 1-D & 2-D MEDIAN FILTERING
- MATRIX ARITHMETIC
  - MATRIX-VECTOR MULTIPLICATION
  - MATRIX-MATRIC MULTIPLICATION
  - MATRIX TRIAGULARIZATION
  - LU DECOMPOSITION
  - QR DECOMAOSITION (EIGENVALUE)
- NON-NUMERIC APPLICATIONS
  - DATA STRUCTURES - STACK & QUEUE SEARCHING
    PRIORITY QUEUE SORTING
  - GRAPH ALGORITHMS - TRANSITIVE CLOSURE,
    MINIMUM SPANNING TREES
  - LANGUAGE RECOGNITION - STRING MATCHING
    REGULAR EXPRESSIONS
  - ENCODERS (POLYNOMIAL DIVISION)
  - RELATIONAL DATA-BASE OPERATIONS

BASIC SYSTOLIC CELL



INNER PRODUCT STEP FUNCTIONAL UNIT

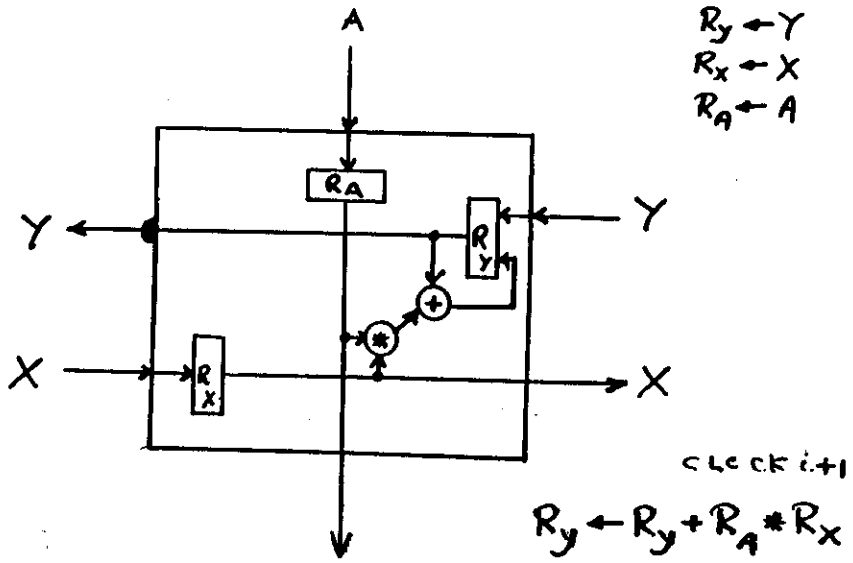
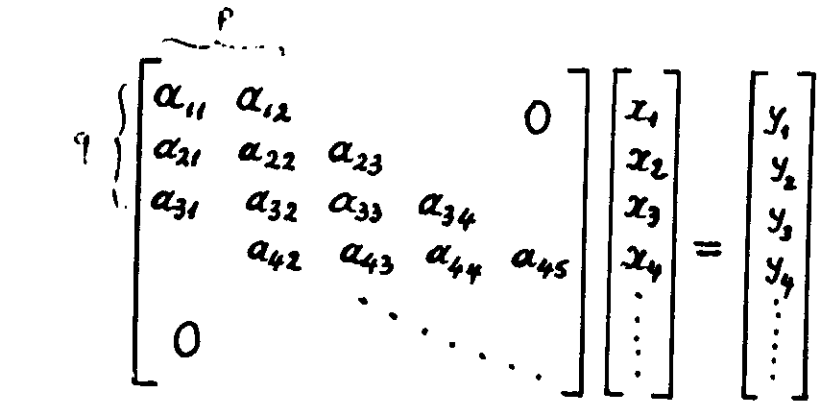
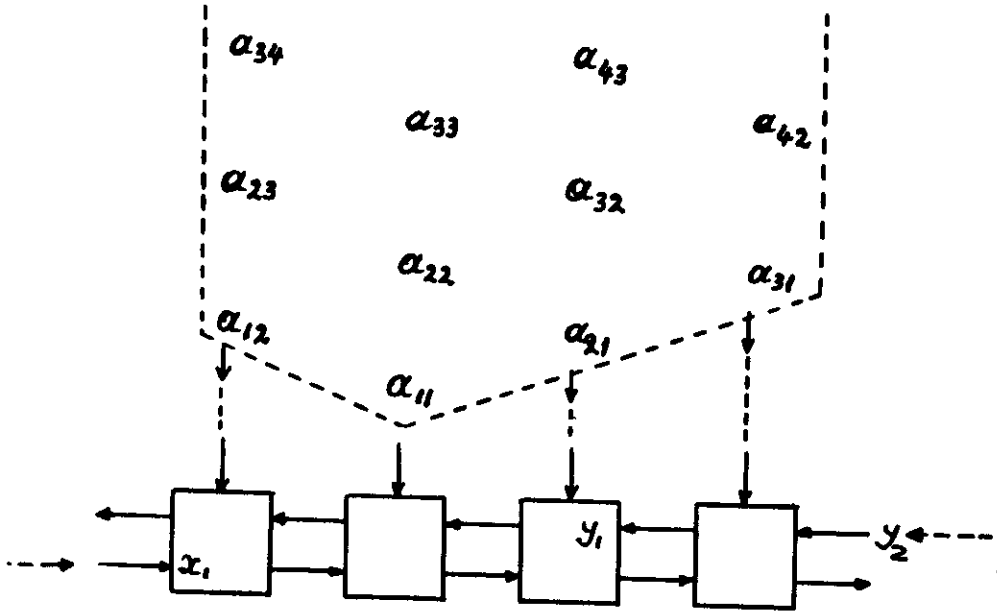


Fig. 3.

BASIC TYPES OF SYSTOLIC STRUCTURES

- ONE-DIMENTIONAL LINEAR ARRAYS
- TWO-DIMENTIONAL ARRAYS
- TREE STRUCTURES



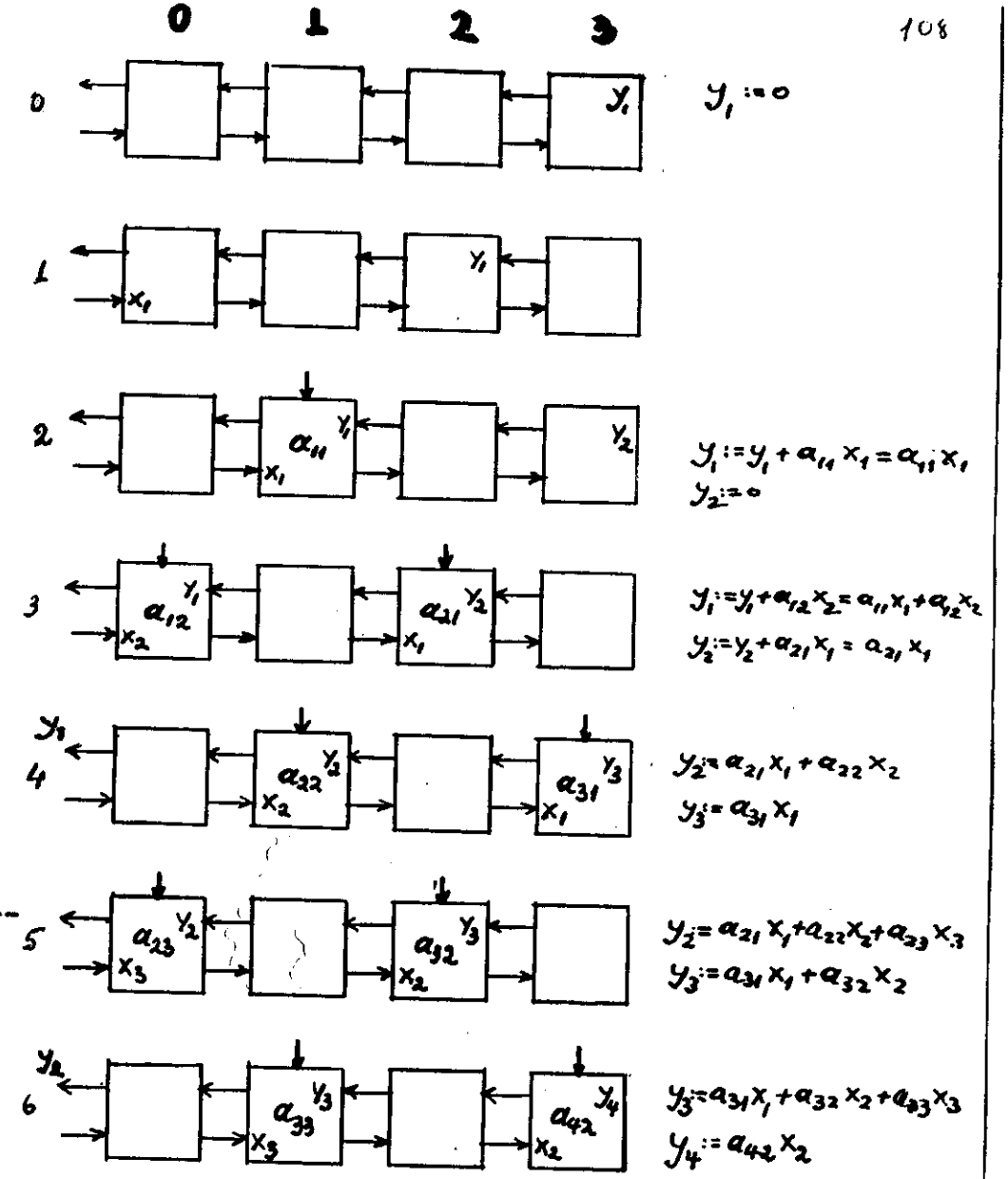


inner product: $y_i^{(k+1)} = y_i^{(k)} + a_{ik} x_k$

$w = p + q - 1$ bandwidth = # systolic cells

$c(n) \Leftrightarrow c(n+1)$

systolic ... pipelining ...



The first seven steps of the matrix-vector systolic multiplication.

TWO-DIMENTIONAL SYSTOLIC TOPOLOGIES

SQUARE

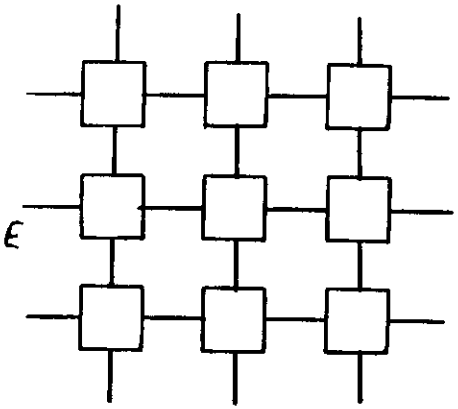
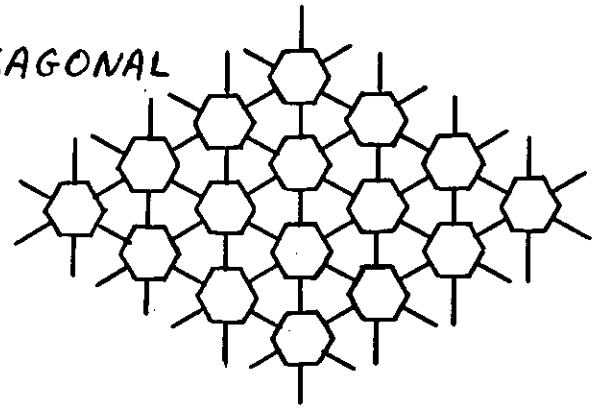


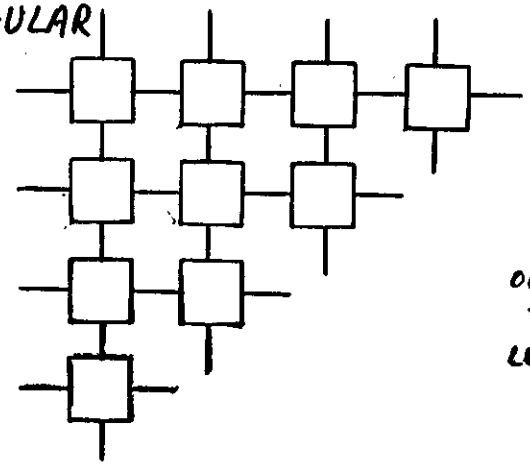
IMAGE PROCESSING
PATTERN MATCHING
RELATIONAL OPERATIONS
GRAPH ALGORITHMS

HEXAGONAL



MATRIX ARITH
TRANSITIVE CLOSURE
PATTERN MATCH
DFT

TRIANGULAR

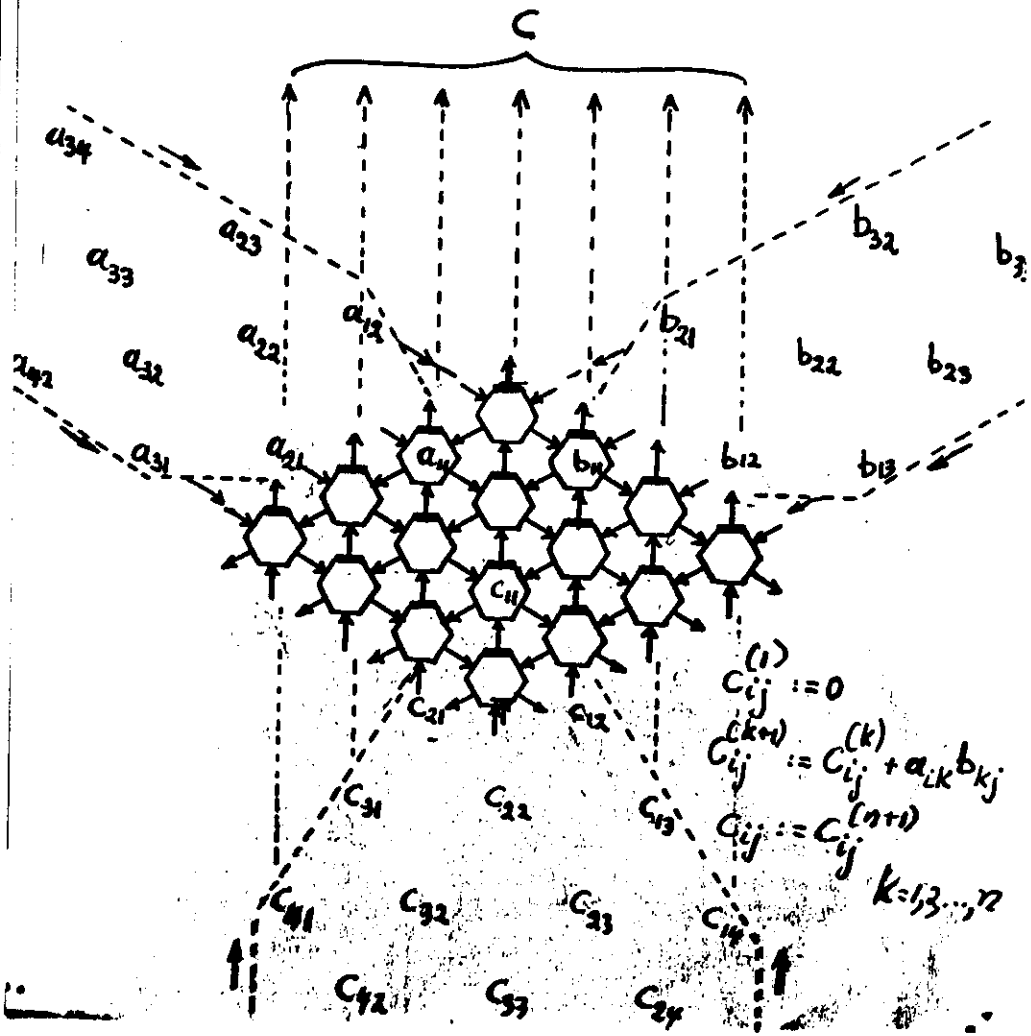


ORTHOGONAL TRIANGULARIZATION
LEAST SQUARE FIT

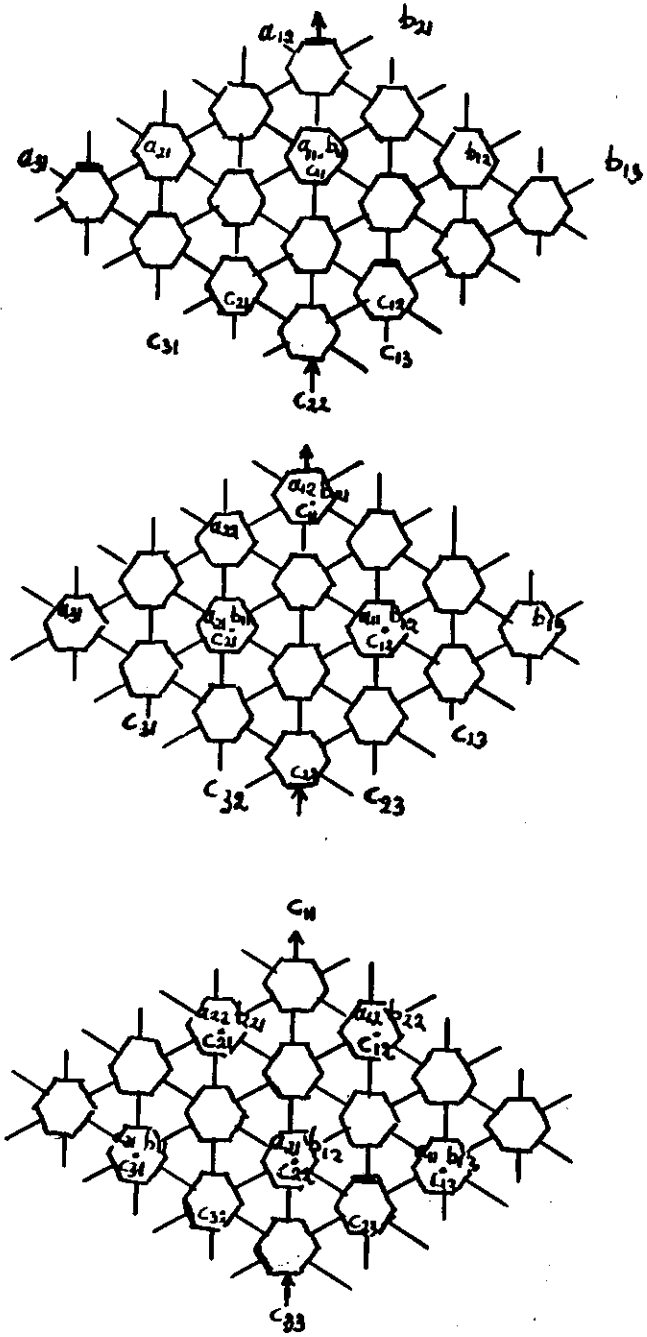
MATRIX-MATRIX MULTIPLICATION

$$\begin{bmatrix} a_{11} & a_{12} & & 0 \\ a_{21} & a_{22} & a_{23} & \\ a_{31} & a_{32} & a_{33} & a_{34} \\ & a_{42} & & \ddots \\ 0 & & & \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} & b_{13} & & 0 \\ b_{21} & b_{22} & b_{23} & & \\ & b_{32} & b_{33} & b_{34} & b_{35} \\ & & b_{43} & & \\ 0 & & & & \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} & c_{14} & 0 \\ c_{21} & c_{22} & c_{23} & c_{24} & \\ c_{31} & c_{32} & c_{33} & c_{34} & \\ c_{41} & c_{42} & & \ddots & \\ 0 & & & & \end{bmatrix}$$

A B C



THE FIRST TREE MULT. STEPS

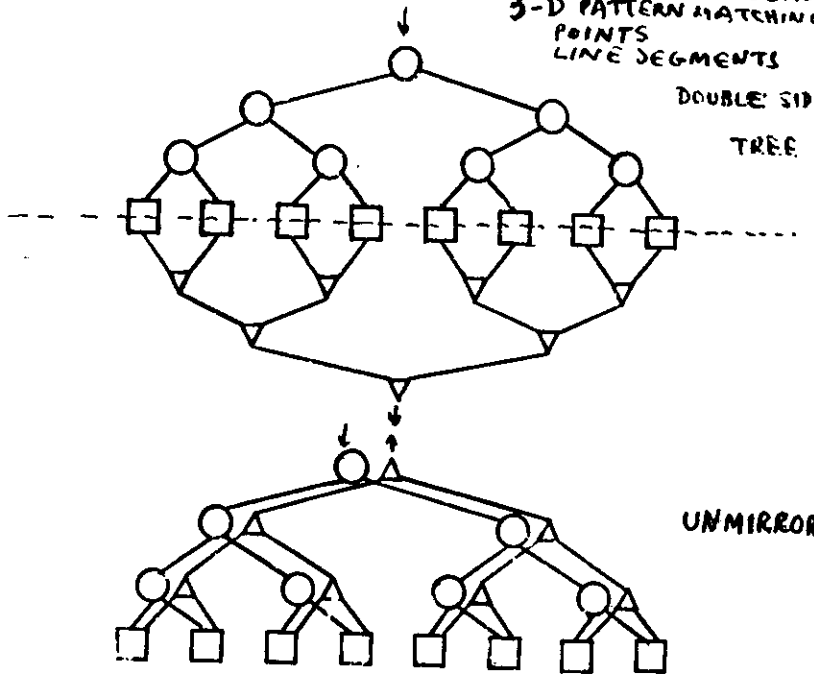


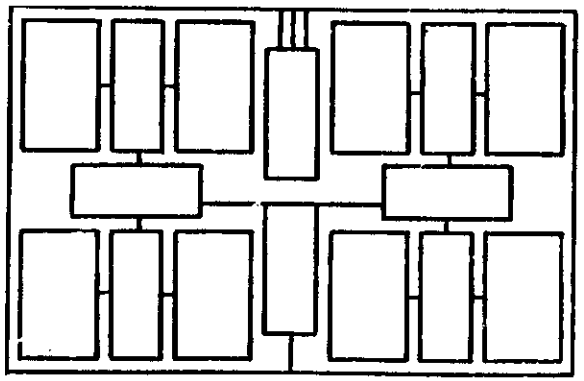
SYSTOLIC TREES

RELATIONAL OPERATIONS } SELECT, JOIN, PROJECT, UNION, INTERSECT

FOR SEARCH

3-D PATTERN MATCHING.
POINTS
LINE SEGMENTS





VLSI LAYOUT

SYSTOLIC ARRAY IMPLEMENTATIONS

a) FULL CUSTOM DEVICES (SINGLE-PURPOSE)
- INNER PRODUCT CELL
- MULT-ACC CHIP

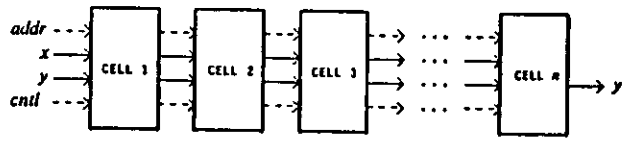
2) INTERMEDIATE RANGE OF FLEXIBILITY
- ESL SYSTOLIC PROCESSOR
- ESL SYSTOLIC CHIPSET FOR F.P. MATRIX CALC.
- WARP PROCESSOR (LINK)

3) GENERAL PURPOSE
- NOSC SYSTOLIC ARRAY TESTBED (FROM G.P μP's)
- PSC : PROGRAMMABLE SYSTOLIC CHIP
- CHiP
- TRANSPUTER

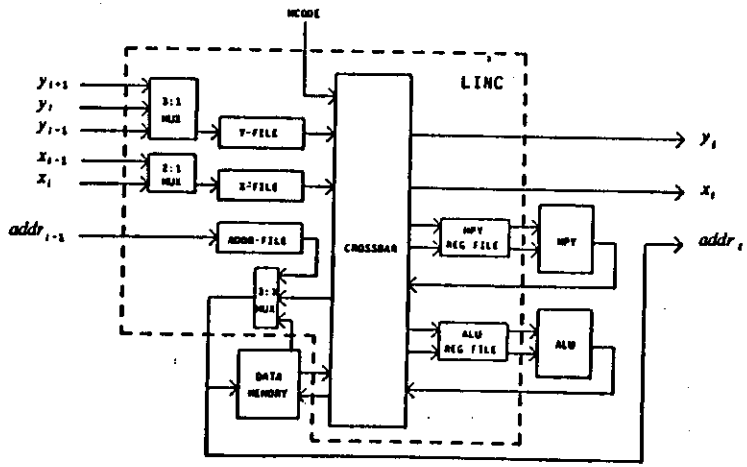
THE CMU WARP PROCESSOR (CMU, 1987)

PROTOTYPE OF 10 CELLS

200 nsec CYCLE TIME ⇒ 10 MOPS/CELL
1024-point FFT / 600 μsec



ONE DIMENSIONAL SYSTOLIC ARRAY

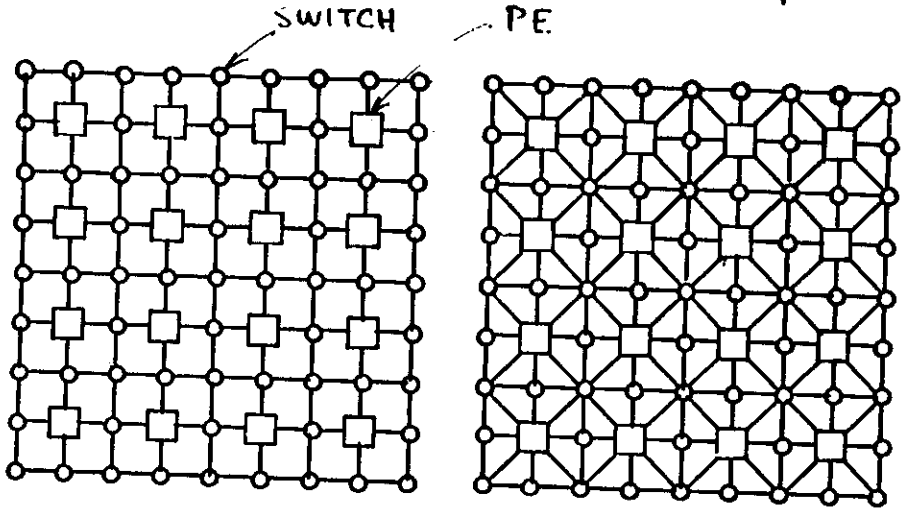


THE WARP CELL

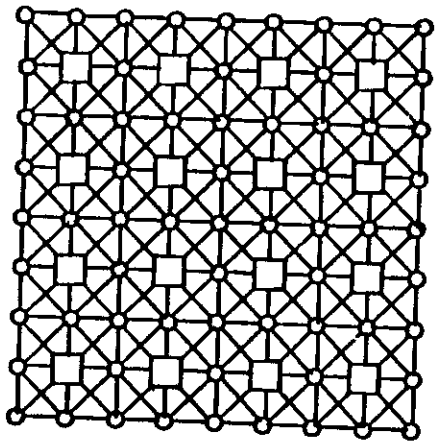
[LINC chip + WEITEK 32-bit MULT
WEITEK 32-bit ALU

CHiP CONFIGURABLE HIGHLY PARALLEL COMPUTER

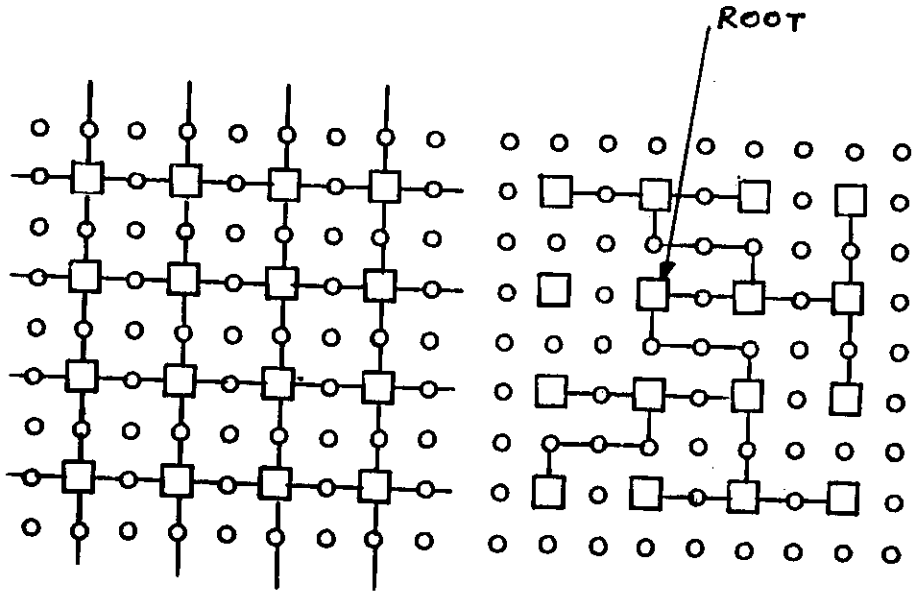
- A SWITCHABLE SYSTOLIC ARRAY
  VLSI 2^8 to 2^{16} PEs HOMOGENEOUS μPROCESSORS



SWITCH LATTICE STRUCTURES



- SWITCH-LATTICE RECONFIGURABLE
- FAULT TOLERANCE
- WAFER-LEVEL FABRICATION



MESH PATTERN

BINARY TREE

CONCLUSIONS

SYSTOLIC DATAFLOW STRUCTURES ARE SUCCESSFULLY USED IN SEVERAL COMPUTE-BOUND APPLICATIONS

SIGNAL PROCESSING, IMAGE PROCESSING } MAN-MACHINE INTERFACE

DATA BASE MACHINES

WORK ON

SYSTOLIC ALGORITHMS

LARGE SCALE DESIGNS

AUTOMATIC DESIGN

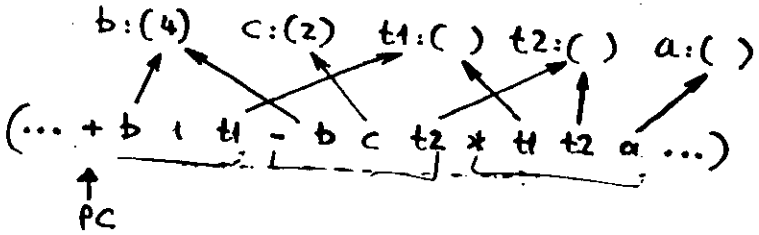
INTEGRATION INTO COMPLETE SYSTEMS

DATAFLOW COMPUTING

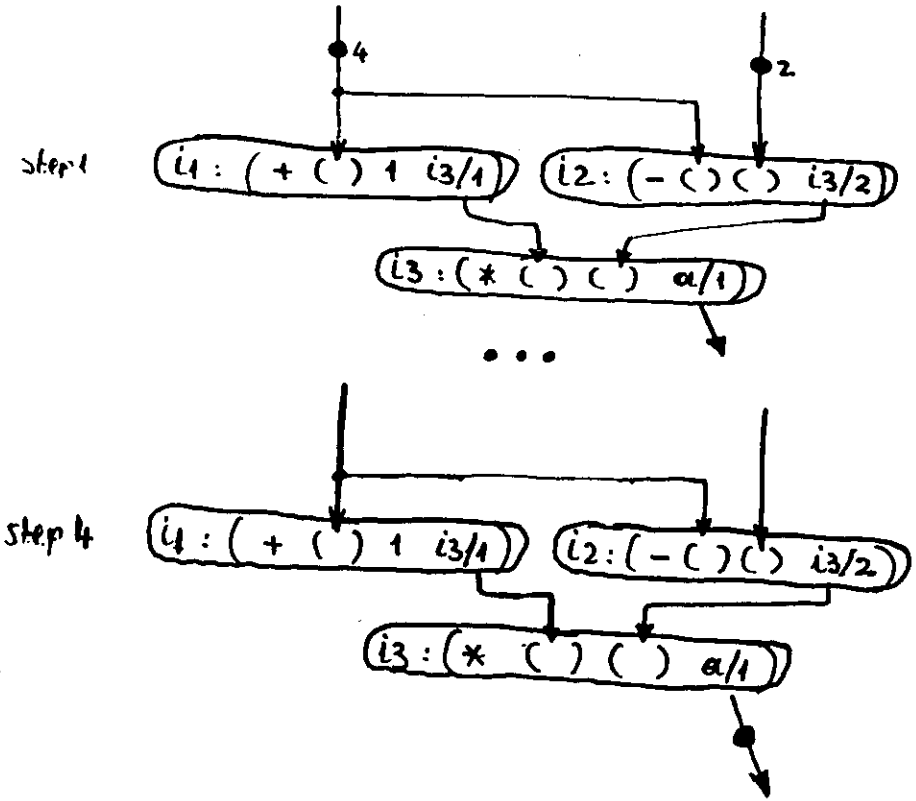
- STARTED BACK IN 1968 (DENNIS, MIT)
- THEORY BY KARP-MILLER - 1966
- FIRST PROTOTYPE WORKING 1979
  THE TOULOUSE LAU SYSTEM BY SYRE
- DESIGN CENTERS
  - USA : MIT, IRVINE, LLL, UTAH
  - EUROPE : MANCHESTER, CERT-ONERA
  - JAPAN : NTT, ETL
- BASED ON THE DATA-DRIVEN MODEL OF COMPOTATION
  - THE AVAILABILITY OF OPERANDS TRIGGERS THE EXECUTION OF THE OPERATION
  - NO PROGRAM COUNTER
- REPRESENTED BY DATAFLOW GRAPHS
  - NODES : OPERATORS
  - ARCS : PATHS OF TOKENS BETWEEN NODES
  - TOKENS : CARRY DATA VALUES

$a = (b+1) * (b-c)$

CONTROL-DRIVEN

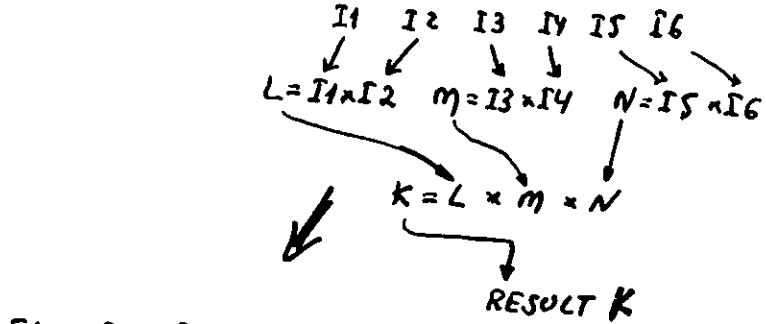


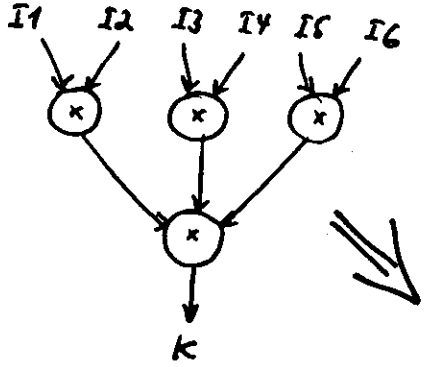
DATA-DRIVEN

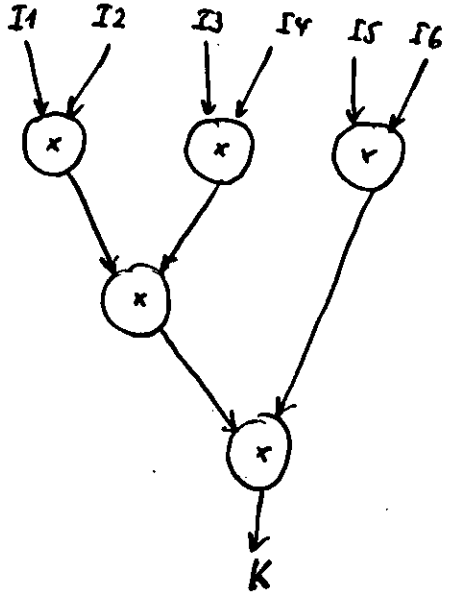


PROGRAMS AS GRAPHS

$L = I1 * I2$
$M = I3 * I4$
$N = I5 * I6$
$K = L * M * N$





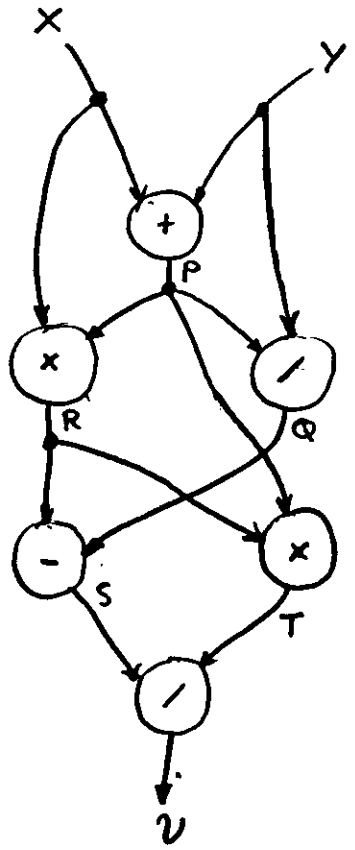


1. P = X + Y
2. Q = P / Y
3. R = X × P
4. S = R - Q
5. T = P × P
6. U = S / T

Possible computations sequenses

(1, 2, 3, 4, 5, 6)
(1, 3, 2, 5, 4, 6)
(1, 3, 5, 2, 4, 6)
(1, 2, 3, 5, 4, 6)
(1, 3, 2, 4, 5, 6)

(1, [2;3], [4;5], 6)



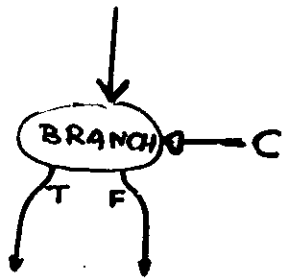
THE DATAFLOW GRAPH REVEALS THE DATA DEPENDENCE OF THE COMPUTATION

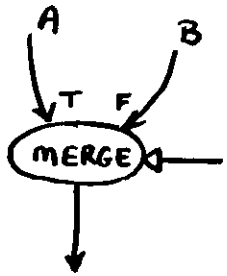
ARCS REPRESENT THE DATA DEPENDENCES BETWEEN NODES

ALL PARALLELISM IS EXPOSED

CONDITIONALS

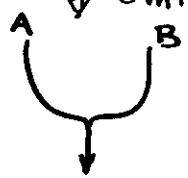
if ... then ... else ... fi

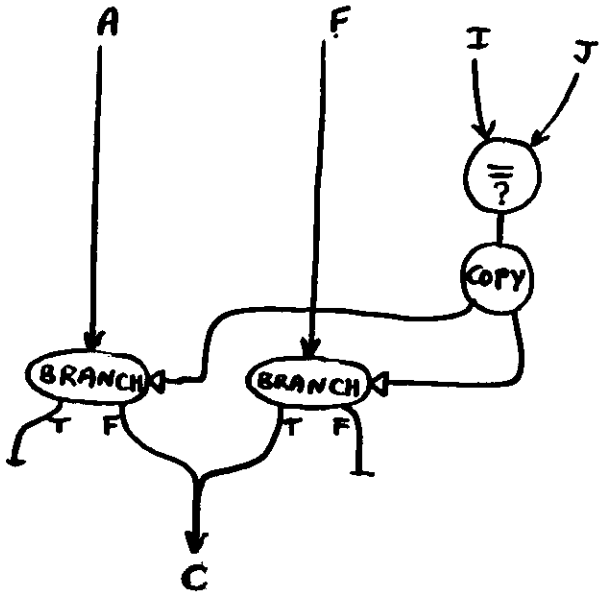




sometimes ⇓ omitted

C = A
IF (I.EQ.J) C = F



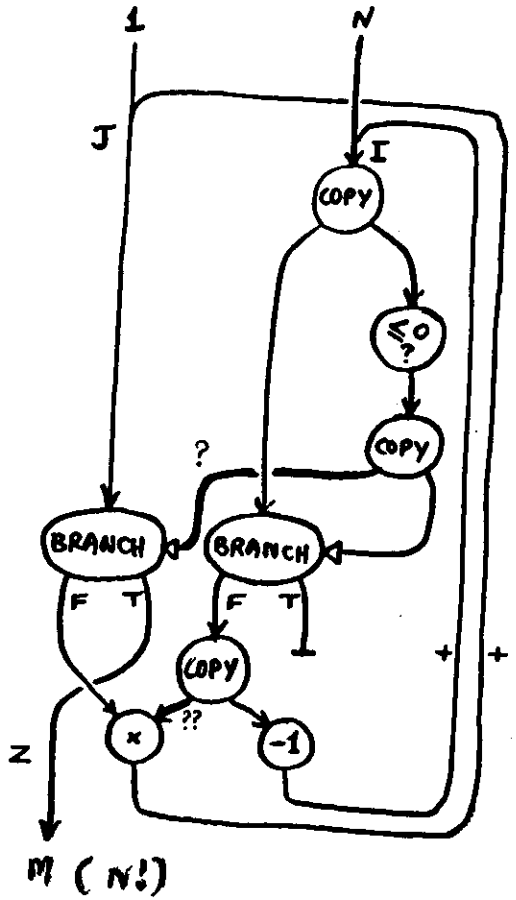


: EAGER OR NOT EVALUATION OF THE then & else BODIES

LOOPS

```
      I=N
      J=1
1ϕ    IF (I.LE.ϕ) GO TO 99
      J=I×J
      I=I-1
      GO TO 1ϕ
99    M=J
```

N!



ARCS SHOULD BEHAVE AS FIFO QUEUES

?, ?? ARCS

MORE THAN ONE TOKEN MAY OCCUPY THESE ARCS

⇓ LOOP UNRAVELLING

STATIC

V.S.

DYNAMIC

\+ COLOURING

z

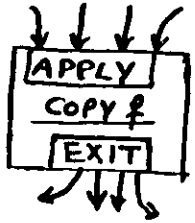
FUNCTIONS - REENTRANT CODE

SOLUTIONS

1) CREATE MANY COPIES & PLANT THEM IN-LINE AT THE APPROPRIATE PLACES

- WASTE OF MEMORY
- INHIBITS RECURSION (INFINITELY EXPANDED FLOWGRAPHS)

2) DYNAMIC CODE-COPYING VIA AN APPLY INSTRUCTION



NO CODE SHARING

3) DYNAMIC TAGGED TOKENS - COLOURING

- TOKENS ARE TAGGED (COLOURED) AS THEY ENTER INTO & EXIT FROM THE REENTRANT AREAS

TOKEN MATCHING

- ONLY TOKENS OF THE SAME COLOUR CAN GROUP TOGETHER TO FIRE A NODE.
- TOKEN TAGGING CAN BE USED TO DISTINGUISH DATA BELONGING TO DIFFERENT CYCLES AROUND A LOOP.

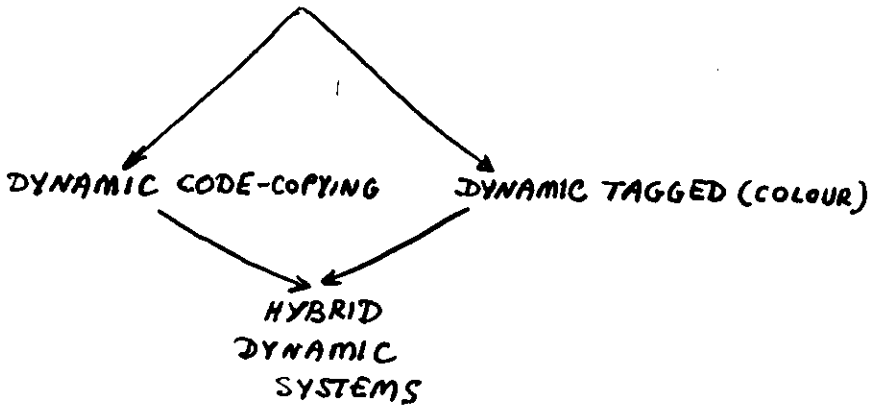
FIRING RULE :

STATIC DATA FLOW

A NODE FIRES WHEN TOKENS ARE PRESENT ON EACH INPUT ARC AND THERE IS NO TOKEN PRESENT ON ANY OUTPUT ARC

DYNAMIC DATA FLOW

A NODE FIRES WHEN TOKENS ARE PRESENT ON EACH INPUT ARC



* WHEN A NODE FIRES THE INPUT TOKENS ARE REMOVED & A RESULT TOKEN IS PLACED ON THE OUTPUT ARC.

- DATA FLOW GRAPH
  - NODES — OPERATIONS
  - ARCS - CARRY TOKENS
- MACHINE INSTRUCTION
  - OP CODE
  - ROOM FOR OPERAND VALUES
  - DESTINATION ADDRESS(ES) OF RESULT
- BY VALUE FLOW OF DATA
  - FUNDAMENTAL ALSO TO PURELY FUNCTIONAL/APPLICATIVE MODELS
- DATA FLOW LANGUAGES

DATA FLOW LANGUAGES

FEATURES

FREEDOM FROM SIDE EFFECTS (LIKE PURE LISP)
NO GLOBAL VARIABLES BUT CALL BY VALUE

LOCALITY OF EFFECT (DEFINITE SCOPE
NO UNNECESSARY FAR-REACHING DEPENDENCIES

EQUIVALENCE OF INSTRUCTION SCHEDULING CONSTRAINTS WITH DATA DEPENDENCIES

DATA FLOW GRAPH CONTAINS ALL INFORMATION NEEDED TO EXECUTE A PROGRAM

SINGLE ASSIGNMENT RULE

A VARIABLE IS ASSIGNED ONLY ONCE

UNUSUAL NOTATION FOR ITERATIONS

NEW J = J + 1 OR J = OLD J + 1

LACK OF HISTORY SENSITIVITY

NO MEMORY - NO STATE VARIABLES

LANGUAGES FUNCTIONAL TYPE LANGUAGES

VAL

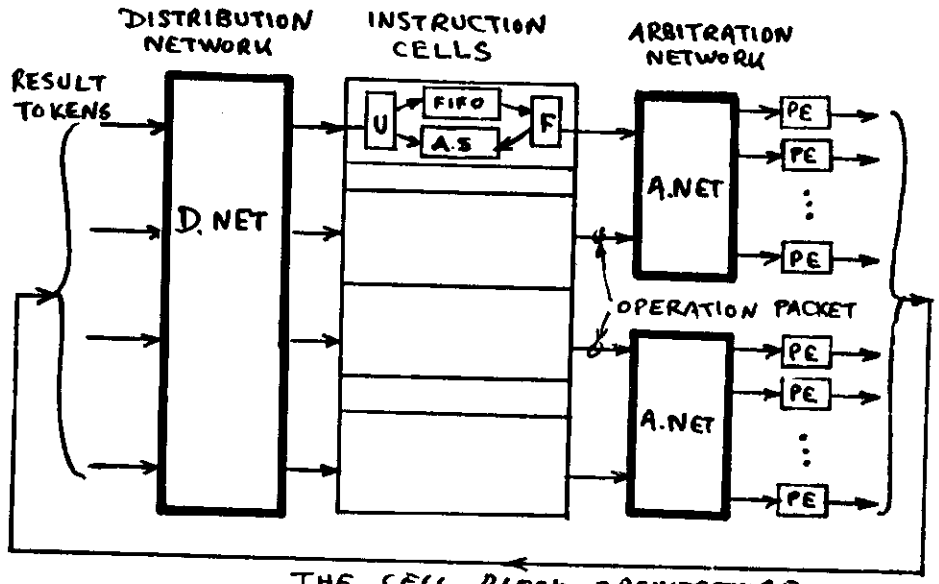
ID

LAPSE

LAU

SISAL (LLL, DEC, UM, COLORADO)

THE DENNIS-MIT MACHINE

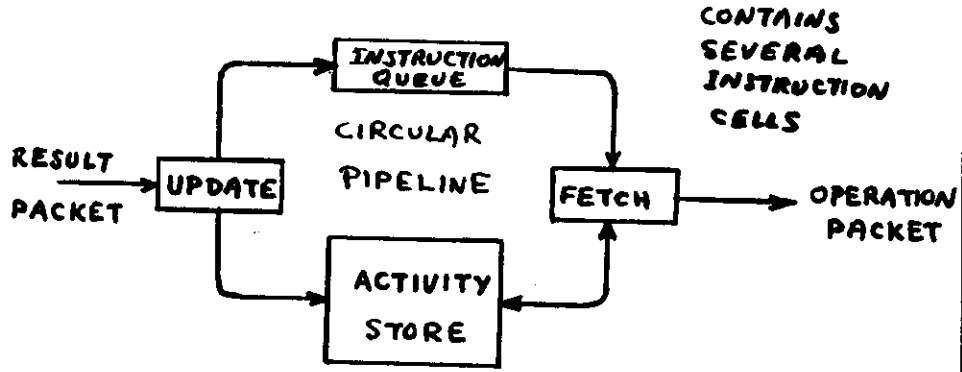


THE CELL BLOCK ARCHITECTURE

- STATIC DATA FLOW < DATA TOKENS, CONTROL TOKENS
- STARTED IN 1968
- VAL LANGUAGE
- PACKET COMM ORGANIZATION WITH TOKEN STORAGE
- HARDWARE + VAL COMPILER UNDER DEVELOPMENT
- PLANS FOR A 1000 MIPS MACHINE WITH LLL
  - 512 PEs
  - 1000 PACKET ROUTERS (2x2)
  - VLSI CHIPS

DENNIS DATAFLOW MACHINE

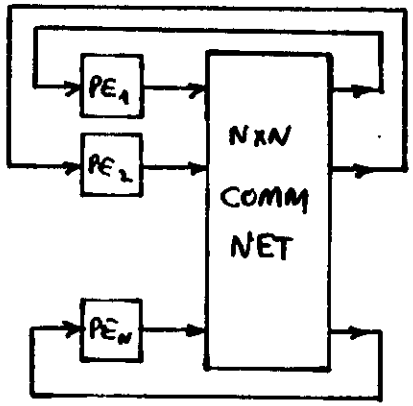
THE INSTRUCTION CELL BLOCK



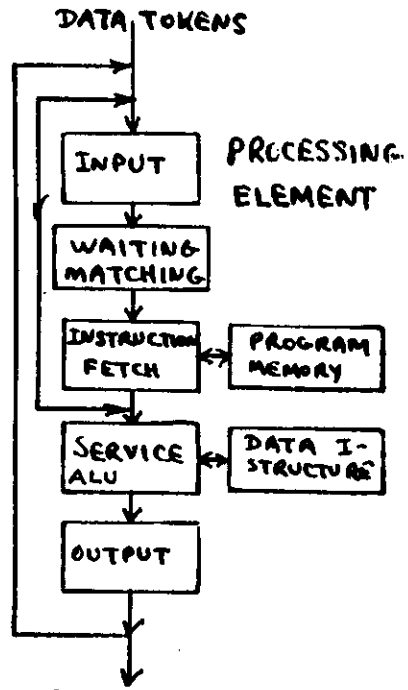
OPERATION

1. The UPDATE unit receives a RESULT packet
2. It updates the instruction cell into the ACTIVITY store
3. It checks if that instruction cell has received all its inputs and if YES
4. It sends into the INSTR. Q the cell's address.
5. The FETCH unit retrieves instructions according to INST. Q. addresses, and
6. Sends them to the PROCESSING UNIT via the ARBITRATION NET.

THE IRVINE-MIT MACHINE OF ARVIND 1980



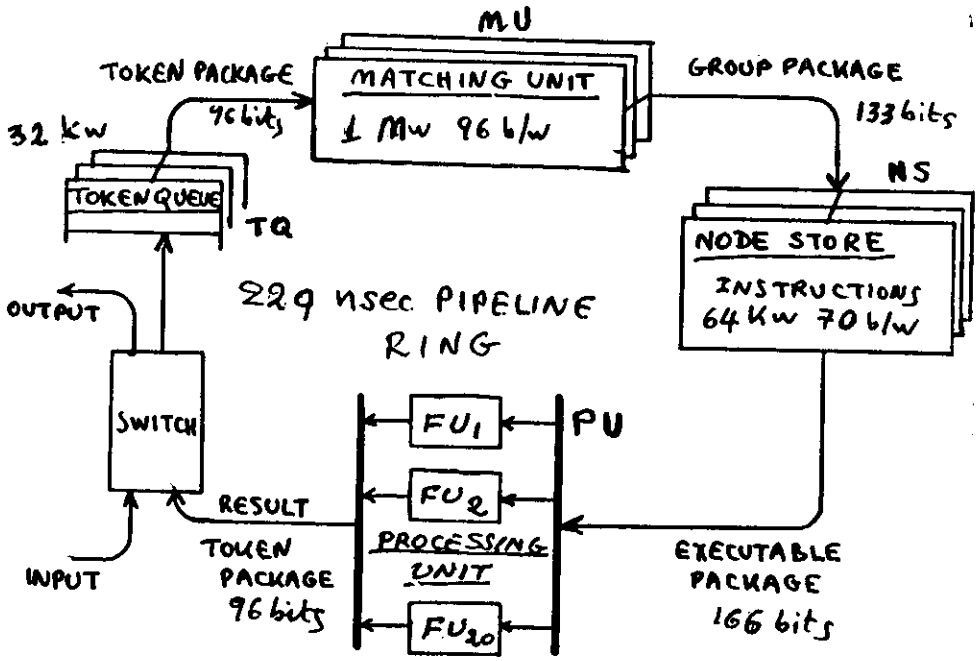
MEMORY IS DISTRIBUTED INTO THE PES



- DYNAMIC DATA FLOW / COLORING
- PACKET COMM ORGANIGATION WITH TOKEN MATCHING
- ID LANGUAGE — U INTERPRETER
- I-STRUCTURE (LENIENT CONS) AVOIDS STRUCTURE COPYING
- A 64PEs MACHINE IS UNDER CONSTRUCTION

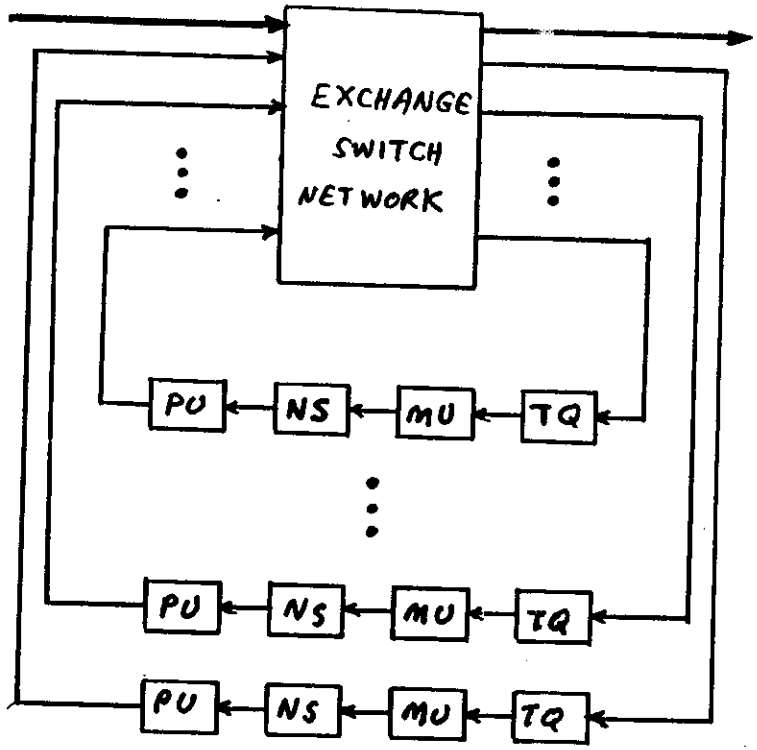
THE U INTERPRETER DYNAMICALLY COLORS LOOP INTANCES & FUNCTION APPLICATION

UNIV. OF MANCHESTER DATA FLOW MACHINE 1976



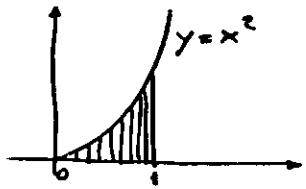
- DYNAMIC DATA FLOW / COLOURING
- PACKET COMM ORGANIZATION WITH TOKEN MATCHING
- LANGUAGE LAPSE / SISAL
- PROTOTYPE COMPLETED IN 1981
- UP TO 20 FU'S PER PROCESSING UNIT
- 0.27 MIPS / FU
- 10 M packets/sec max rate
- MULTI-RING MACHINE PROPOSED

MULTI-RING STRUCTURE OF THE MANCHESTER DATAFLOW MACHINE



PROGRAMMING IN SISAL

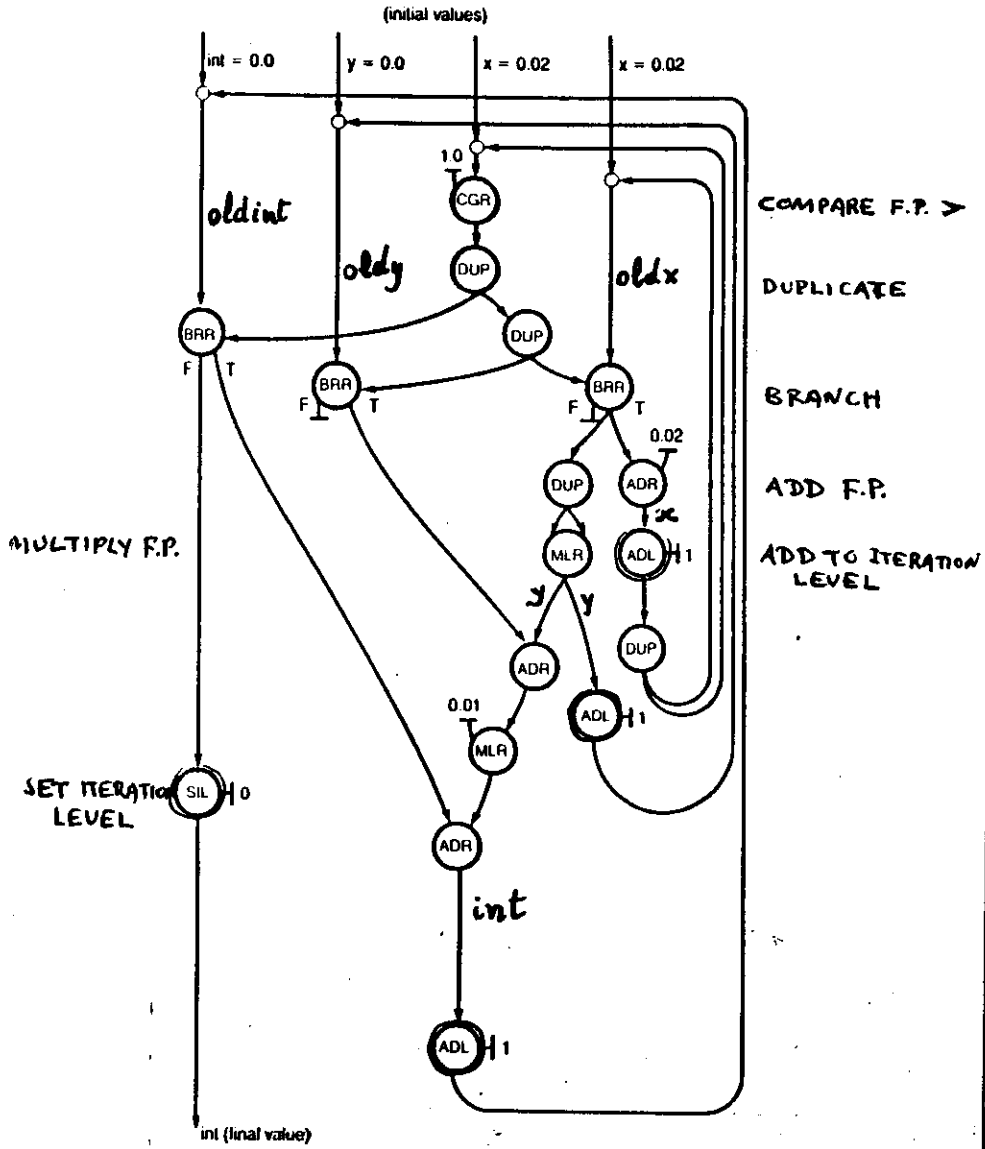
```
function Integrate (returns real)
for initial
        int := 0.0 ;
        y   := 0.0 ;
        x   := 0.02
while
        x < 1.0
repeat
        int := 0.01 * (oldy + y) + oldint
        y   := oldx * oldx ;
        x   := oldx + 0,02
returns
        value of sum int
end for
end function
```



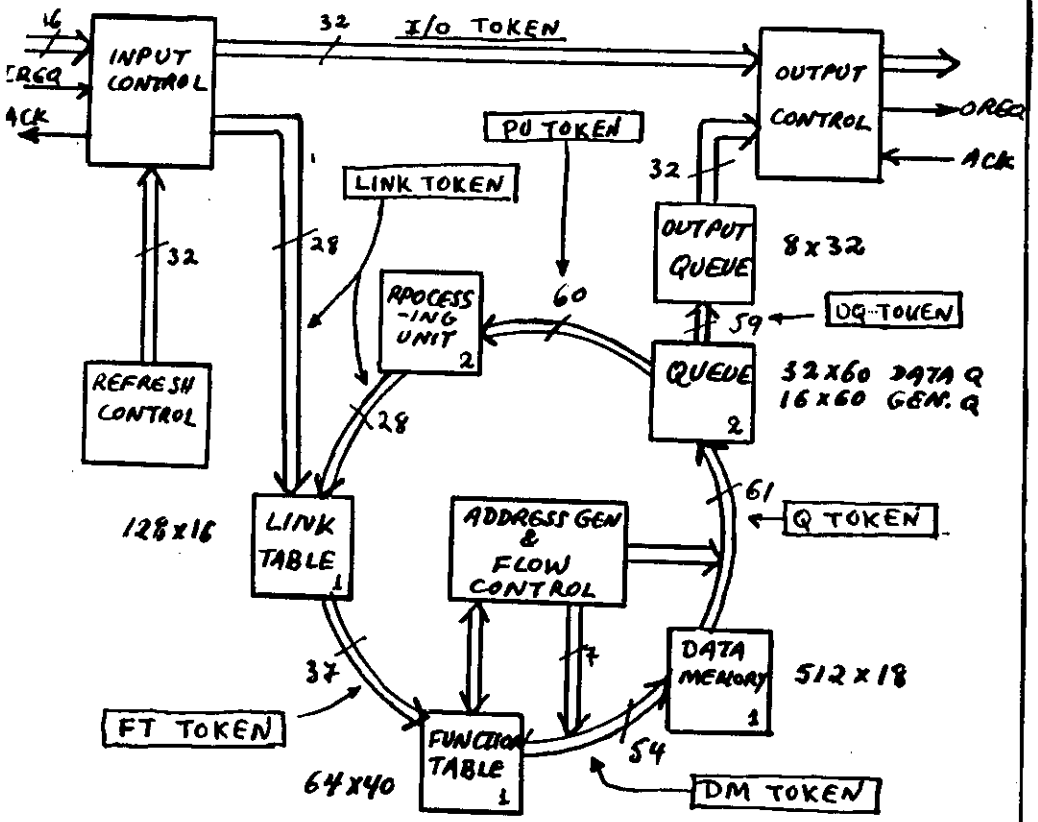
A SISAL program is compiled into the template assembler language TASS

TASS is finally converted to machine code

DATAFLOW GRAPH OF THE INTEGRATION function



THE NEC μPD7281 DATAFLOW CHIP



TOKEN DATA FLOW ARCHITECTURE

PIPELINED RING STRUCTURE - SEVEN CYCLE TIMES

POWERFUL INSTRUCTION SET FOR IMAGE PROCESSING

17x17 BIT MULTIPLIER (200 nsec)

HIGH SPEED I/O

MULTI-PROCESSOR (IN CASCADE) CONFIGURATION

APPLICATIONS: SIGNAL & IMAGE PROCESSING

μPD7281 INSTRUCTION SET

PU INSTRUCTIONS	AG/FC INSTRUCTIONS
ADD/SUB	QUEUE 1ST OPRND
MULT	R/W CYCLIC
INC/DEC	R/W DM
SHR/SHL	CONVOLVE
CMP	DIVIDE
MASK OR/AND	DISTRIBUTE
ACC	SAVE
OR/AND/XOR/NOT	CUT
	PICKUP
	COUNT

(LINK TABLE) ⟹ ARCS OF THE PROGRAM GRAPH

(FUNCTION TABLE) ⟹ NODES OF THE PROGRAM GRAPH

THE PROGRAM IS INITIALLY LOADED INTO LT & FT VIA SPECIAL INPUT TOKENS

DM STORES THE 1ST OPERAND UNTIL THE 2nd ARRIVES

PERFORMANCE BENCHMARKS		1U	3U
ROTATION	512x512 IMAGE	1.5s	0.6s
½ SHRINK	4	80 ms	30ms
SMOOTH	4	1.1 s	0.4s
3x3 CONVOL	4	3 sec	1.15s
cos(x)	33-bit fixed point	40 μs	15 μs
64-stage FIR	17-bit	30 μs	18 μs

CONCLUSIONS (DATA FLOW)

1) SIMPLE PRINCIPLE
BUT
DIFFICULT TO REALIZE

2) TOO EARLY WITH PRESENT
TECHNOLOGY ?

3) CERTAINLY CAN BE
USED AT A
HIGHER LEVEL
OF PARALLELISM

REDUCTION COMPUTING

- STARTED BACK IN 1971 (BERKLING, GMD)
- THEORY BY ALONZO CHURCH - 1951
  LAMBDA CALCULUS
- MANY DESIGN CENTERS
  USA : MIT, N.CAROLINA, UTAH
  EUROPE : KENT, NEWCASTLE, CAMBRIDGE, GMD
  IMPERIAL COLLEGE
  JAPAN : NEC
- BASED ON THE DEMAND-DRIVEN
  MODEL OF COMPUTATION

  THE REQUIREMENT FOR AN OPERAND
  TRIGGERS THE OPERATION WHICH
  WILL PRODUCE IT EVENTUALLY
- REPRESENTED BY NESTED EXPRESSIONS
- FUNCTIONAL PROGRAMMING

  KEY ISSUE : REFERENTIAL TRANSPARENCY

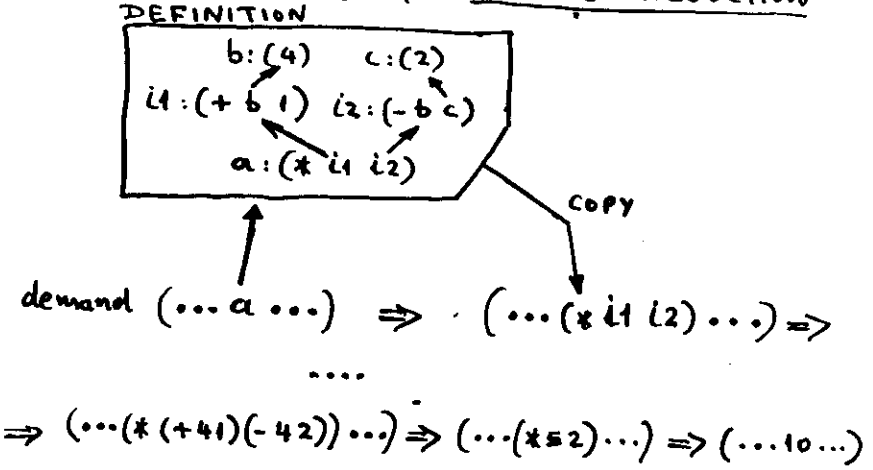
  THE VALUE OF AN EXPRESSION DEPENDS
  ONLY ON ITS TEXTUAL CONTEXT,
  NOT ON COMPUTATIONAL HISTORY.

- PROGRAM = SET OF FUNCTIONS
- REDUCTION COMPUTATION =

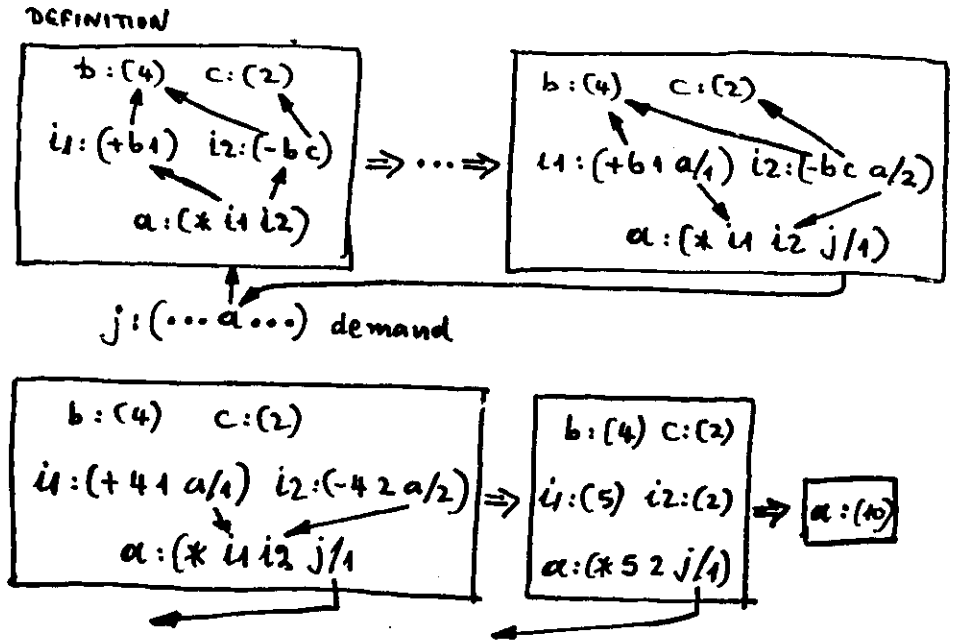
  APPLICATION OF FUNCTIONS
  UPON THEIR ARGUMENTS

  CALLED: BETA-CONVERSION
- REDUCTION METHODS
  - STRING REDUCTION
  - GRAPH REDUCTION
- REDUCTION ORDER OF ARGUMENTS
  - NORMAR ORDER AFTER SUBSTITUTION
  - APPLICATIVE ORDER BEFORE SUBSTITUTION
- REDUCTION ORDER : ANY
  - MAKES NO DIFFERENCE TO THE VALUE
    ALTHOUGH IT MAY AFFECT TERMINATION
    (CHURCH-ROSSER THEOREM)
  - IN PRACTICE :
    INNERMOST ORDER
    OUTERMOST ORDER

DEMAND-DRIVEN (a) STRING REDUCTION



GRAPH REDUCTION



REDUCTION ORDER

- **INNERMOST ORDER**

$$((A*C)-(B*D),(A*D)+(B*C))$$

AN INNERMOST COMPUTATION IS NEVER SELECTED UNTIL ITS ARGUMENTS ARE AVAILABLE ⇒ DATA DRIVEN?

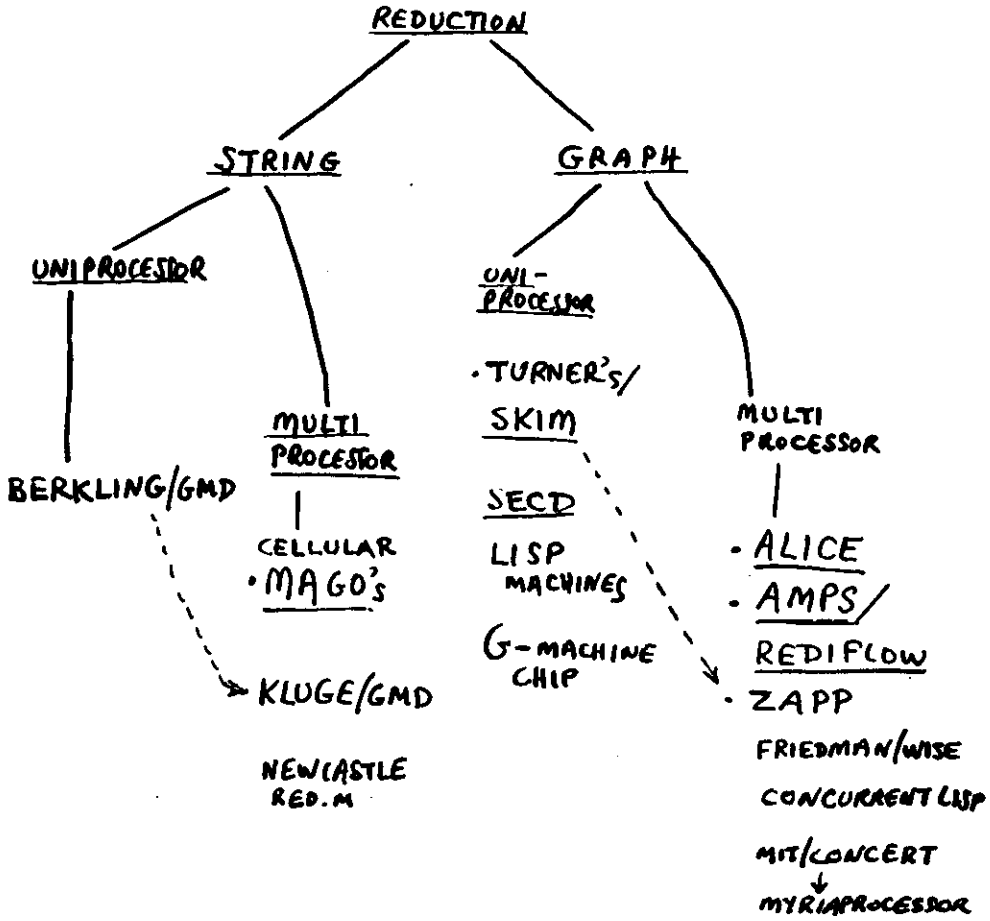
- **OUTERMOST ORDER**

$$((A*C)-(B*D),(A*D)-(B*C))$$

AN OUTERMOST COMPUTATION IS SELECTED WHEN ITS OUTPUT IS NEEDED BY ANOTHER ALREADY SELECTED COMPUTATION ⇒ DEMAND DRIVEN

THE ARGUMENTS ARE COERCED INTO THE REQUIRED FORM BY SPAWNING, IF NECESSARY, NEW REDUCTIONS AND WAITING FOR THEM TO RETURN WITH A VALUE

SPECTRUM OF REDUCTION MACHINES

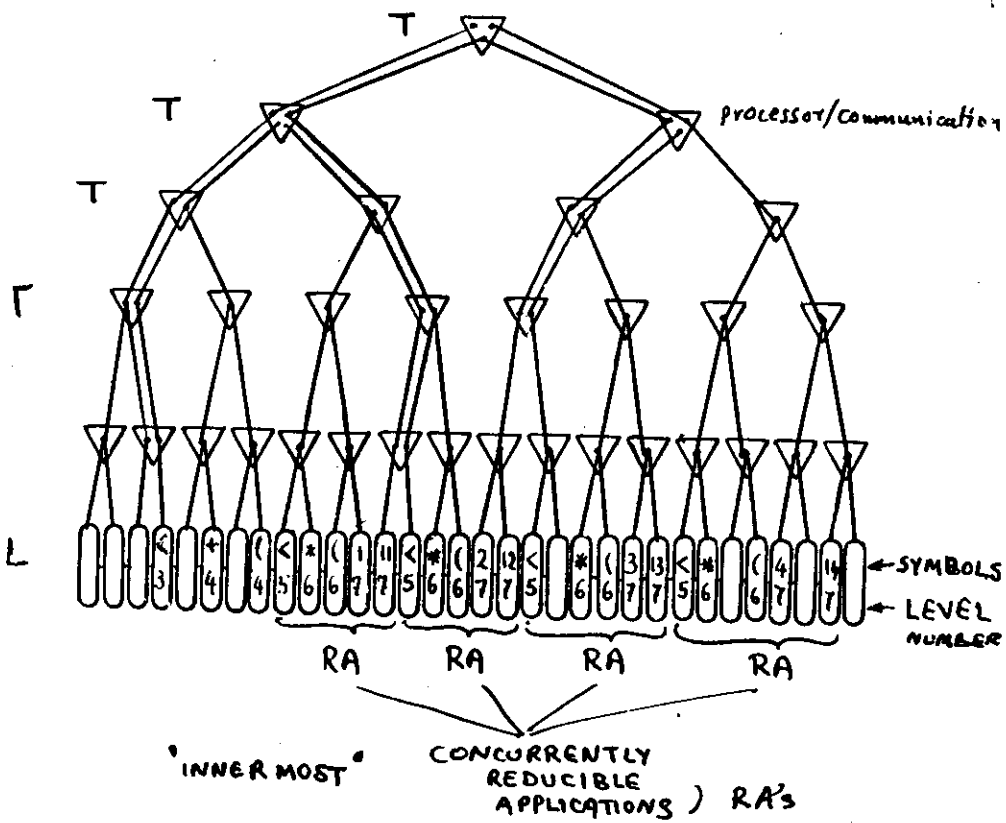


FUNCTIONAL LANGUAGES

LISP / INTERLISP / COMMON LISP / ...
ML
HOPE
SASL / KRC / MIRANDA
FFP
DATAFLOW (VAL, ID, SISAL, ...)

MAGO's REDUCTION MACHINE

TREE MACHINE

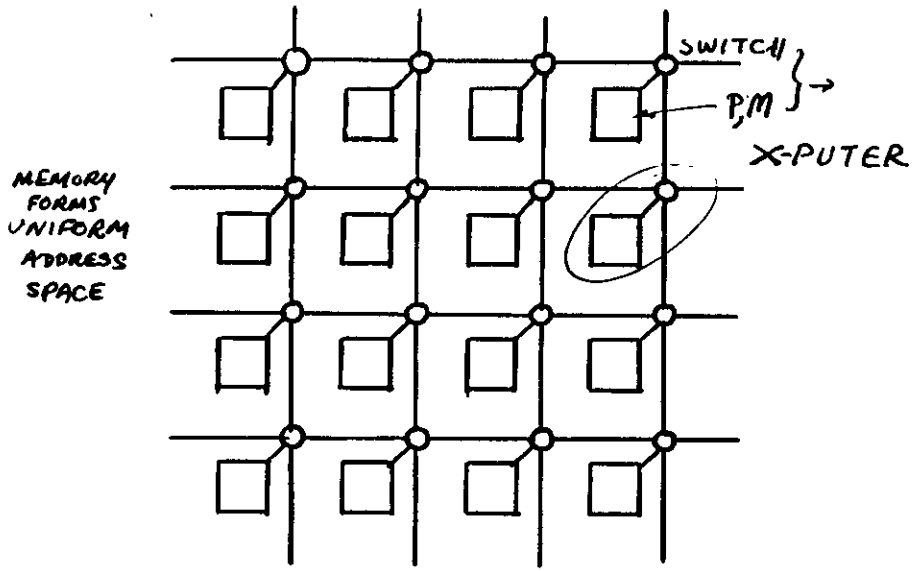


- LARGE NUMBER OF CELLS (VLSI chips) T & L
- STRING REDUCTION WITH CONCURRENT RAs
- BACKUS' FFP LANGUAGE, ALSO MACHINE LANGUAGE
- THROUGHPUT : $O(N/\log N)$ N = # of L chells
- T & L CELLS ARE SMALL F.S. MACHINES
- EACH L CELLS HOLDS ONE SYMBOL

OVERALL ORGANIZATION & OPERATION OF MAGO'S MACHINE

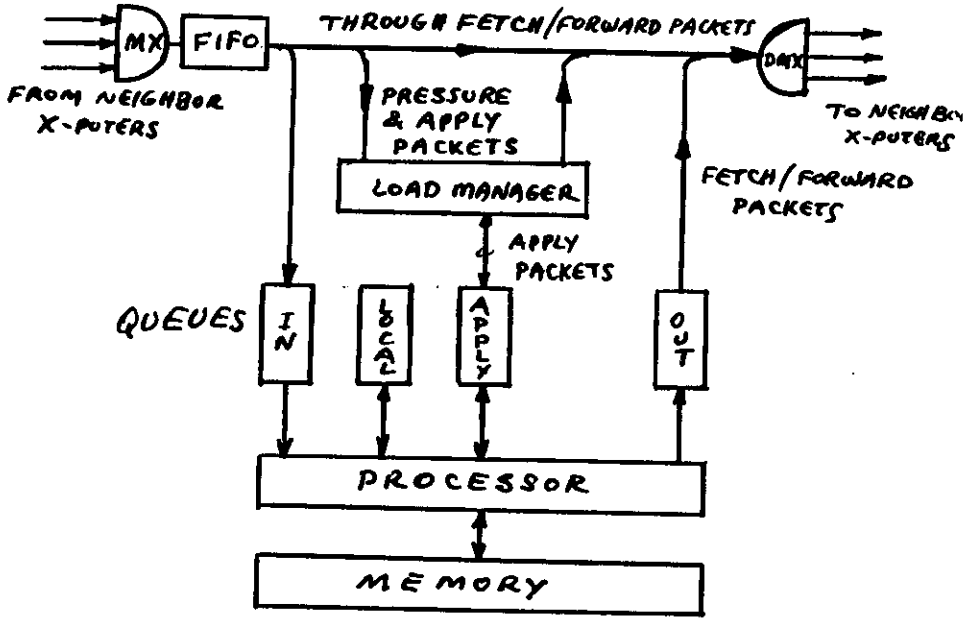
- FFP EXPRESSION IS INITIALLY LOADED INTO L ARRAY (LEFT TO RIGHT, ONE SYMBOL PER L CELL)
- FFP EXPRESSION IS REDUCED IN INNERMOST ORDER IN A SERIES OF DOWNSWEEP AND UPSWEEP CYCLES
- FFP EXARESSION AND THE MACHINE TREE ARE AUTOMATICALLY AND DYNAMICALLY DECOMPOSED INTO RA's AND SUB-TREE AREAS RESPECTIVELY
- THE WHOLE OPERATION IS CONTROLLED VIA MICROPROGRAMS DISTRIBUTED TO THE TREE NODES BY THE ROOT.
- DOWNSWEEP & UPSWEEP CYCLES MAY OVERLAP IN TIME AND INVOLVE MOVEMENT OF DATA ALONG AND ACROSS SUB-TREE AREAS
- THREE MODES OF OPERATION
  - MODE I : T & L COMPUTE, L COMMUNICATE
  - MODE II : L COMPUTE, T COMMUNICATE
  - MODE III : L COMMUNICATE DIRECTLY (STORAGE MANAGEMENT)

THE REDIFLOW MACHINE - UTAH



- COMBINES REDUCTION WITH DATAFLOW
- GRANULARITY OF PARALLELISM AT THE FUNCTION LEVEL
- THIS EXPLOITS LOCALITY OF INFO & REDUCES COMMUNICATION
- DATAFLOW IS EMPLOYED AT HIGHER LEVEL
- PACKET SWITCHING COMMUNICATION
- LOAD BALANCING MECHANISM
  BASED ON THE NOTION OF PRESSURE
- SIMULATION BENCHMARKS

PACKET FLOW WITHIN A REDIFLOW X-PUTER



FETCH PACKET — FETCH ADDRESS / DESTINATION ADDRESS

FORWARD PACKET — VALUE FETCHED / DESTINATION ADDRESS

APPLY PACKET — $\langle f : x \rangle$

PRESSURE PACKET — PROPAGATED PRESSURE, PP

$$PP = \text{if } IP < \text{threshold then } 0 \text{ else } \min[1+EP, \text{ceiling}]$$

INTERNAL PRESSURE

$$IP = \text{queue length} + k \frac{1}{1 - \text{fraction of memory used}}$$

EXTERNAL PRESSURE

$$EP = \min[\text{PP's of neighbors}]$$

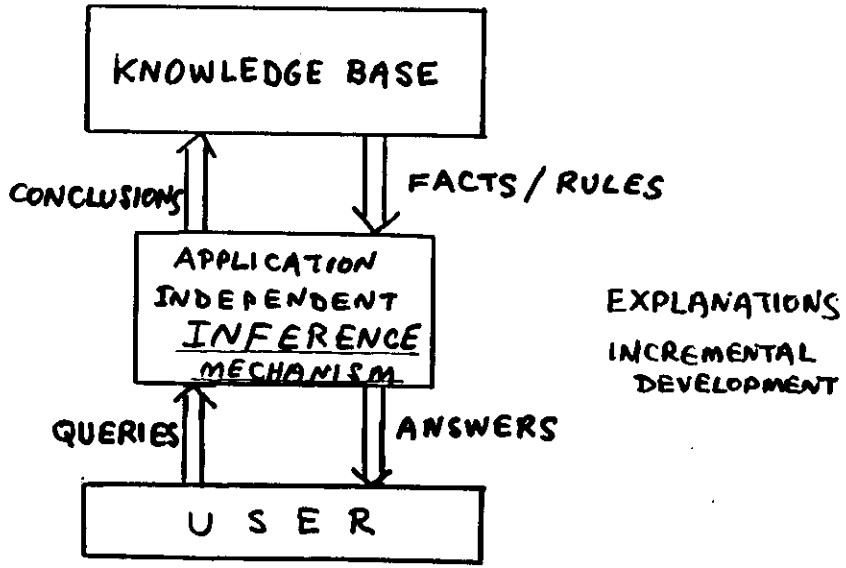
$$\text{ceiling} = 1 + \text{network's diameter}$$

LOGIC MACHINES & LOGIC PROGRAMMING SYSTEMS

- PROGRAMMING BY "DESCRIPTION"

  YOU CONSTRUCT A PROGRAM
  BY DESCRIBING ITS APPLICATION AREA
  BY SAYING WHAT IS TRUE

- ⇓ PUT INTO



EXPLANATIONS
INCREMENTAL DEVELOPMENT

- ORIGINS TRACED BACK TO LEIBNIZ (17th CENTURY)
- PRACTICALLY STARTED MUCH LATER

  1965 ROBINSON + COLMERAUER 1970

  RESOLUTION PROLOG

PROLOG

- SUBSET OF 1ST ORDER PREDICATE CALCULUS "THE HORN CLAUSES"
- THREE BASIC FORMS OF CLAUSES

1. A. FACTS/ASSERTIONS } AXIOMS
2. $B \leftarrow A_1 \& A_2 \& \cdots \& A_n$ RULES } AXIOMS
3. $\leftarrow G_1 \& G_2 \& \cdots \& G_m$ GOALS. } THEOREMS QUESTIONS.

- TWO INTERPRETATION OF $B \leftarrow A_1 \& \ldots \& A_n$ (HEAD, BODY)

1. DECLARATIVE

   B IS TRUE IF A_1 AND ... AND A_n ARE TRUE

2. PROCEDURAL

   TO PROVE B FIRST PROVE A_1 AND ... AND A_n.

- $A, B, A_1, \ldots, A_n, G_1, \ldots, G_m$ ATOMS OF THE FORM:

  $P(t_1, t_2, \ldots, t_n)$ CALLED PREDICATE OF n ARITY

  t_i TERM [constant (a, 1.5, true) | constant function $f(t_1', \ldots, t_m')$ | variable (X, Y, ..)]

- THE PREDICATE EXPRESSES A RELATIONSHIP AMONG ITS ARGUMENTS $t_1, \ldots, t_n$.

PROLOG PROGRAM

- IT IS A SEQUENCE OF CLAUSES WHOSE VARIABLES ARE CONSIDERED TO BE UNIVERSALLY QUANTIFIED

- ITS EXECUTION IS A

  GOAL-DRIVEN

  TOP-DOWN INTERPRETATION

  BACKWARD REASONING

  PROCESS

  AXIOMS ⟵ THEOREMS

- AND IT CAN BE REPRESENTED BY AN

  AND-OR RESOLUTION TREE

  PROOF - SEARCH

- TRAVERSED IN A

  DEPTH-FIRST LEFT-TO-RIGHT

  ORDER

- VIA

  RESOLUTION - UNIFICATION

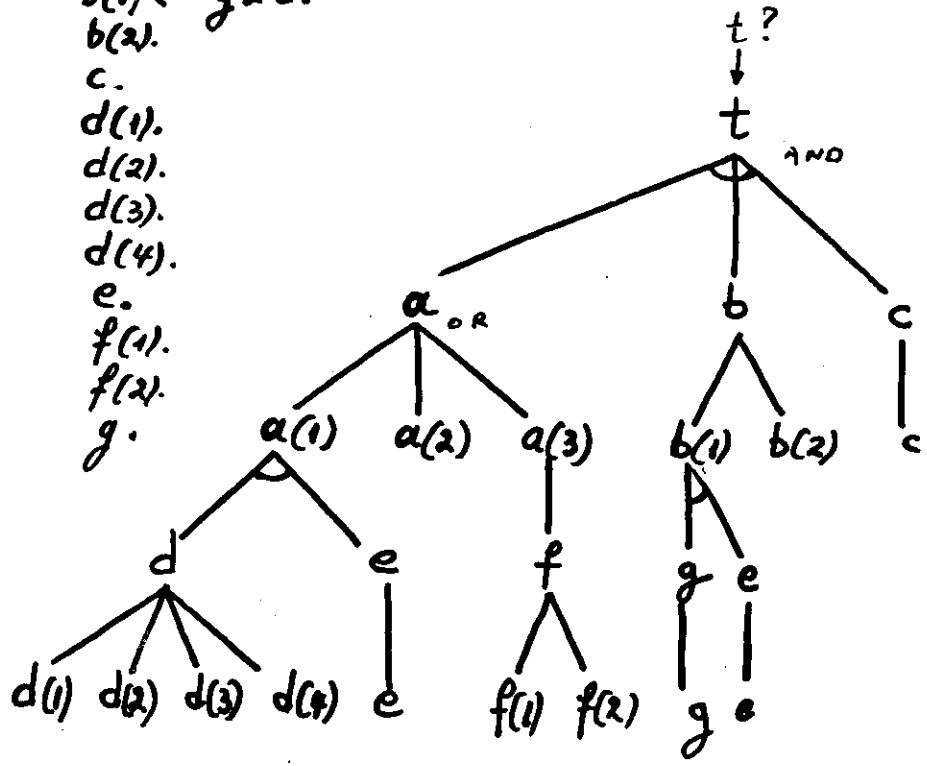
  &

  BACKTRACKING

AND-OR RESOLUTION TREE

$\leftarrow t.$ goal.

$t \leftarrow a \,\&\, b \,\&\, c.$ RULES & FACTS
$a(1) \leftarrow d \,\&\, e.$
$a(2).$
$a(3) \leftarrow f.$
$b(1) \leftarrow g \,\&\, e.$
$b(2).$
$c.$
$d(1).$
$d(2).$
$d(3).$
$d(4).$
$e.$
$f(1).$
$f(2).$
$g.$



RESOLUTION IN GENERAL

GIVEN: $A \leftarrow B \,\&\, C \,\&\, D$

$F \leftarrow A \,\&\, G$

RESOLVE INTO: $F \leftarrow B \,\&\, C \,\&\, D \,\&\, G$

"NON-DETERMINISTIC"

"WHEN AN ATOM ON THE LEFT HAND SIDE UNIFIES WITH AN ATOM ON THE RIGHT HAND SIDE A NEW CLAUSE CAN BE DEDUCED"

LUSH-RESOLUTION IN PROLOG

GIVEN GOAL $\leftarrow A \,\&\, B \,\&\, C$

& CLAUSE $A \leftarrow D \,\&\, E$

RESOLVE INTO

$\leftarrow D \,\&\, E \,\&\, B \,\&\, C$

"WHEN THE FIRST SUBGOAL OF THE GOAL UNIFIES WITH THE HEAD OF A CLAUSE A NEW GOAL IS DEDUCED."

"DEPTH-FIRST SEARCH STRATEGY"

UNIFICATION IN GENERAL

- IT IS THE PROCESS OF FINDING A SET OF BINDINGS FOR THE VARIABLES OF TWO EXPRESSIONS THAT MAKES THEM LOOK ALIKE
- EXAMPLES

$P(X, a) \xleftrightarrow[[X/b,\ Y/a]]{\text{unify}} P(b, Y)$

$P(X, X) \xleftrightarrow{\text{do not unify}} P(a, b)$

~~$[X/a,\ X/b]$~~

$f(X, g(Y), T) \xleftrightarrow[\begin{bmatrix} X/f(a,b) \\ Y/g(a,c) \\ T/Z \end{bmatrix}]{\text{unify}} f(f(a,b), g(g(a,c)), Z))$

$f(X, g(Y), T) \xleftrightarrow[\begin{bmatrix} X/f(a,b) \\ Y/g(Y,c) \\ T/Z \end{bmatrix}]{\text{do not unify}} f(f(a,b), g(g(Y,c)), Z))$ ← OCCUR CHECK

- BACKTRACKING OCCURS WHEN UNIFICATION FAILS OR ANOTHER SOLUTION IS NEEDED

EXAMPLES

```
factorial(0,1).
factorial(N,X) <- lt(N,0) &
                  write('N is negative').
factorial(N,X) <- M:=N-1 &
                  factorial(M,Y) &
                  X:=N*Y.

greek(socrates).
greek(dias).
human(socrates).
mortal(X) <- human(X).

<- mortal(X).
yes, X = socrates

<- greek(X) & ¬ mortal(X).
yes, X = dias

<- ¬ mortal(dias).
yes

<- ¬ mortal(X).
fail.
```

"CLOSED WORLD ASSUMPTION"


```
QUICKSORT([],[]).
QUICKSORT([X1!X],Y) <-
        PARTITION(X,X1,U,V) &
        QUICKSORT(U,U1) &
        QUICKSORT(V,V1) &
        APPEND(U1,[X1!V1],Y).

PARTITION([X1!Rest],X,[X1!Left],Right) <-
        LE(X1,X) & / &
        PARTITION(Rest,X,Left,Right).
PARTITION([X1!Rest],X,Left,[X1!Right]) <-
        PARTITION(Rest,X,Left,Right).

PARTITION([],*,[],[]).

APPEND([],List,List).
APPEND([Head!Tail],List,[Head!NewTail]) <-
        APPEND[Tail,List,NewTail).
```

PARALLELISM IN LOGIC PROGRAMMING

UNIFICATION PARALLELISM

MATCH $A(X, Y, a, f(Z))$
$A(S, b, T, f(T))$

"UNIFY ALL TERMS CONCURRENTLY"

AND PARALLELISM

$a(X, Y, U) \leftarrow b(X, Z) \,\&\, c(X, Y) \,\&\, d(T)$

"PROVE ALL SUBGOALS CONCURRENTLY"

OR PARALLELISM

$\leftarrow g(X)$

$g(X) \leftarrow m(X) \,\&\, n(X)$

$g(X) \leftarrow n(X) \,\&\, p(X)$

"APPLY ALL RULES CONCURRENTLY"
NO BACKTRACKING !!

- THE ABOVE PARALLELISM IS IMPLICIT IN LOGIC PROGRAMMING BUT NOT IN PROLOG

INFERENCE
PROLOG MACHINES

SEQUENTIAL

ICOT's	PSI	30 KLIPS
Kobe Un.	PEK	40 KLIPS
VEC	HPM	280 KLIPS
WARREN	APM	450 KLIPS
Berkley	PLM	425 KLIPS
UTHL	RPM	555 KLIPS

HOSTED SYSTEMS

NCR/32-000	53 KLIPS
SYMBOLICS 3600	110 KLIPS

UNIFICATION COPROCESSORS

SYRACUSE	SUM	
AT&T	HUU	10-20 μsec
IMAG OPALE U		

PARALLEL

ICOT's?	PIM-R PIM-D
Tokyo U	PIE
DADO	COLUMBIA U

COMPARE TO

IBM/3090 VMPROLOG 200 KLIPS

VAX 780 C-PROLOG 1.5K

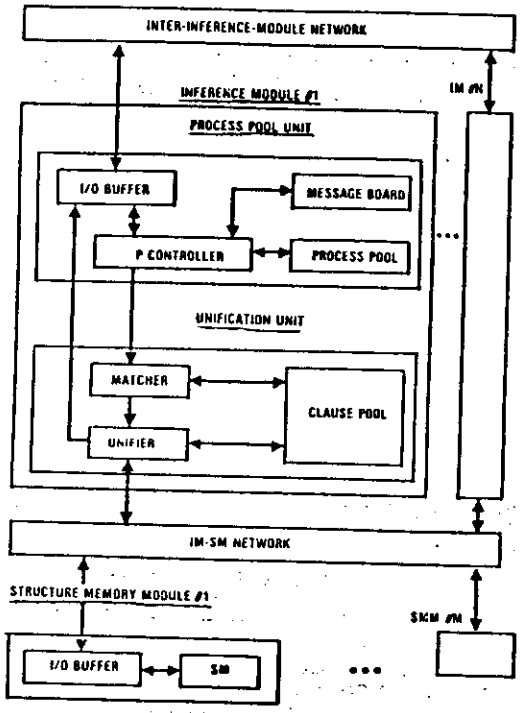
APPLE II Int. 8 LIPS

PIM-R REDUCTION BASED PIM

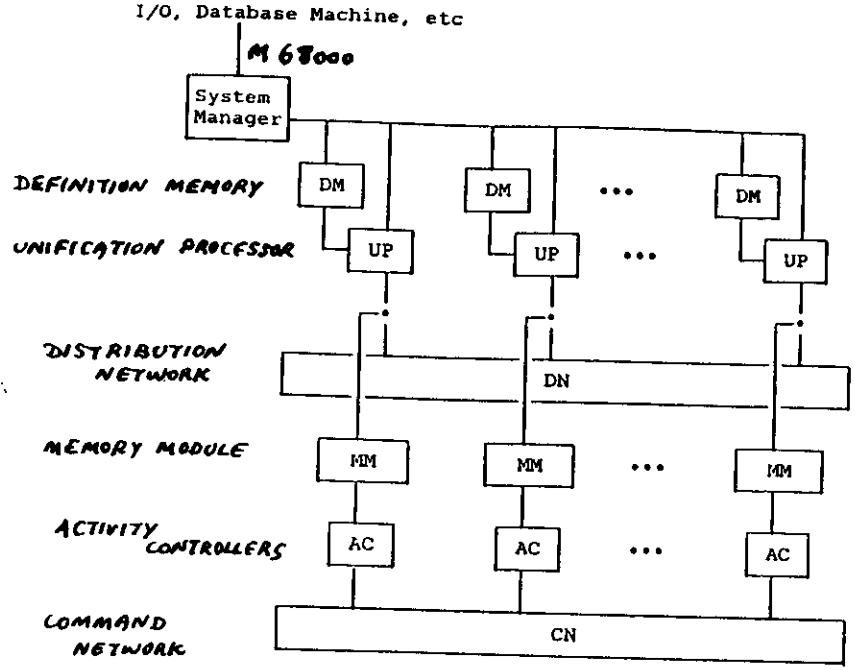
ICOT '83 & HITACHI

TOKYO UNIV. PARALLEL INFERENCE ENGINE PIE

1983



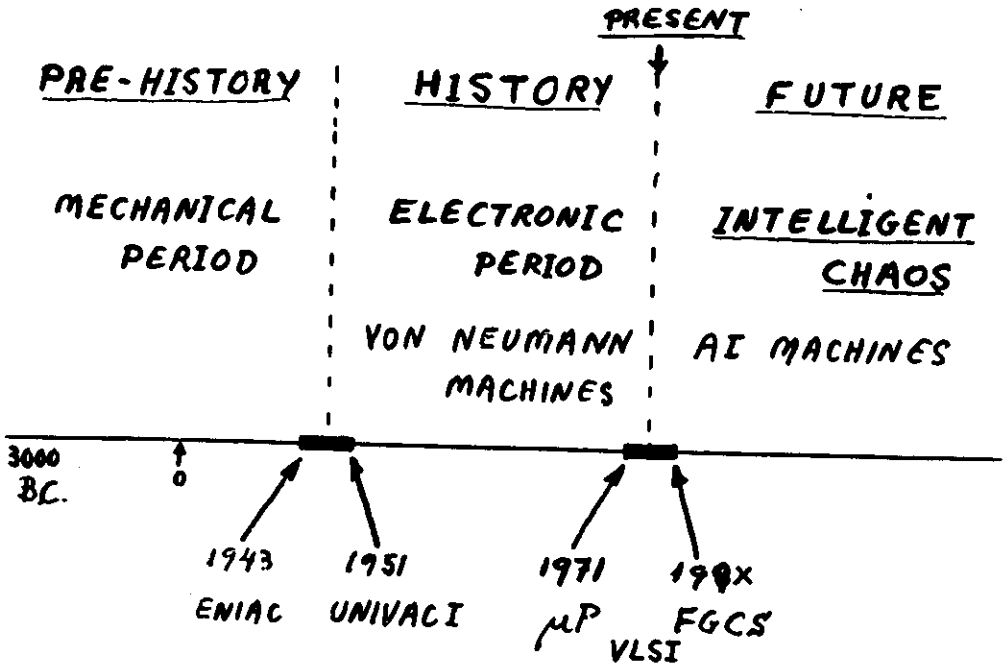
- PACKET COMMUNICATION REDUCTION MACHINE
- CONSISTS OF PPU's, UU's, SMU's
- EXECUTES LOGIC PROGRAMS WITH
  OR CLAUSES IN PARALLEL
  AND CLAUSES IN SERIAL
  } OR-PARALLELISM
- AND PARALLEL EXECUTION OF CONCURRENT PROLOG
- EACH CLAUSE POOL STORES ALL CLAUSES

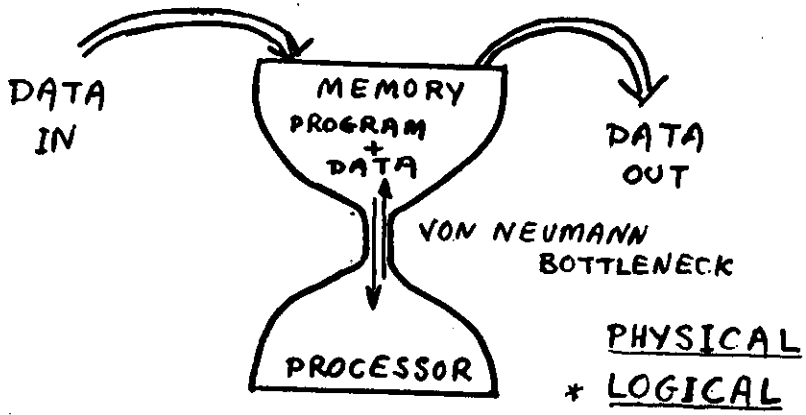


STATUS : SIMULATION
UP HAS BEEN DESIGNED
& Programmed 200 ns cycle time

- OR-PARALLELISM
- SEVERAL UNIFICATION PROCESSORS ≈ 1000
- ACTIVITY CONTROLLERS HOLD CURRENT PROOF TREE - MAKE LOAD BALANCING
- DEFINITION MEMORIES HOLD CLAUSES (COPIES)
- MEMORY MODULES HOLD GOAL CLAUSES
- DN DISTRIBUTES GOALS
- 100 MLIPS - 1 GLIPS TARGET PERFORMANCE
  TESTS : 170 SPEED-UP USING 256 UP's

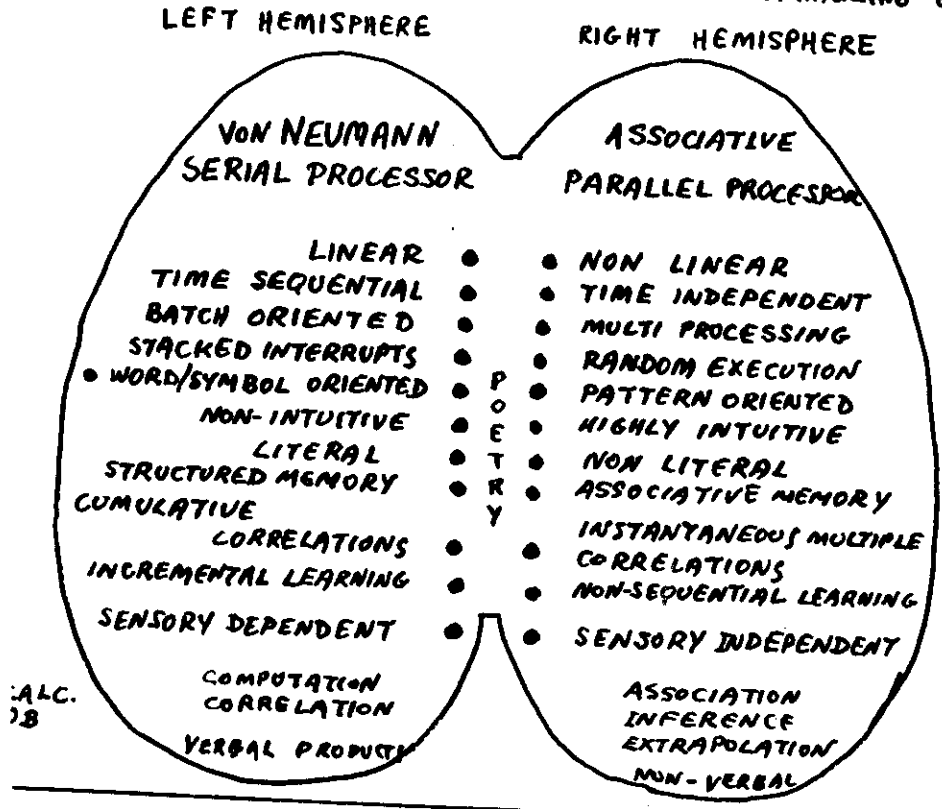
THE EVOLUTION OF COMPUTERS



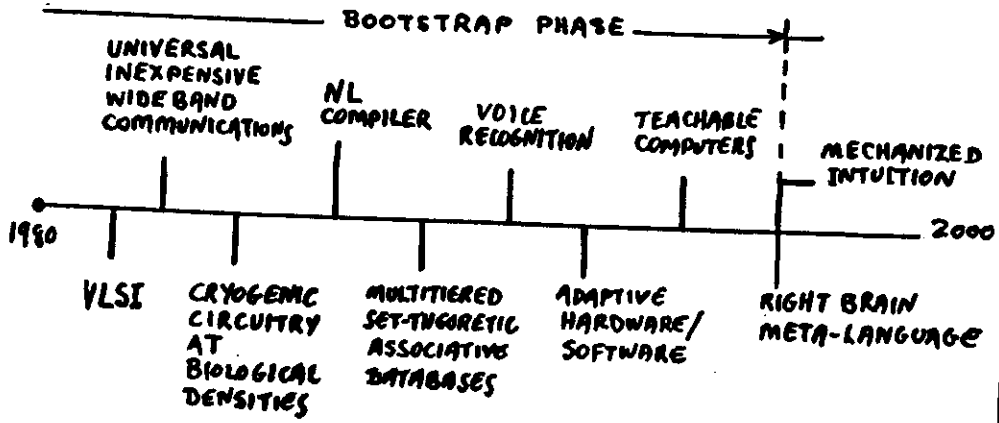


OPPOSING CHARACTERISTICS OF CORESIDENT BRAIN COMPUTERS

V. RAUZINO DAT 82



PROJECTED TECHNOLOGICAL EVOLUTION TOWARS RIGHT BRAIN ERA



1. SELECTIVE BIBLIOGRAPHY

1. K. Hwang and F.A. Briggs, COMPUTER ARCHITECTURE AND PARALLEL PROCESSING, McGraw – Hill, New York, 1984.

2. K.L. Clark and S.A. Tarnlund, LOGIC PROGRAMMING, Academic Press, 1982.

3. F.B. Chambers, DISTRIBUTED COMPUTING, Academic Press, 1984.

4. FIFTH GENERATION COMPUTER SYSTEMS, North Holland, 1984.

5. H. Glaser, Ch.Hankin and D. Till, PRINCIPLES OF FUNCTIONAL PROGRAMMING, Prentice Hall, 1984.

6. P. Henderson, FUNCTIONAL PROGRAMMING, APPLICATIONS AND IMPLEMENTATION, Prentice Hall, 1980.

7. K. Hwang, Ed., TUTORIAL, SUPERCOMPUTERS: DESIGN AND APPLICATIONS, IEEE Catalog no EH0219 – 6, 1984.

8. T. Moto – oka, Ed., FIFTH GENERATION COMPUTER SYSTEMS, North Holland, 1982.

9. P. Bertolazzi and F.Luccio,Eds., VLSI: ALGORITHMS AND ARCHITECTURES, North Holland, 1985.

10. D.A. Duce, Ed., DISTRIBUTED COMPUTING SYSTEMS, Peregrinus, 1984.

11. R. Kowalski, LOGIC FOR PROBLEM SOLVING, Elsevier Publ., 1979.

12. J.D. Ullman, COMPUTATIONAL ASPECTS OF VLSI, Computer Science Press, 1984.

2. REFERENCES.

2.1 Introductory part.

13) L.O. Hertzberger, New Architectures, Computing in High Energy Physics, Amsterdam, June 25 – 28, 1985.

14) D.D. Gajski and J – K. Peir, Essential Issues in Multiprocessor Systems, Computer, June 1985.

15) K. Hwang, Multiprocessor Supercomputers for Scientific/Engineering Applications, Computer, June 1985.

16) P.B. Schneck et al, Parallel Processor Programs in the Federal Government, Computer, June 1985.

17) V. Rauzino, Conversations with an Intelligent Chaos, Datamation, – – – – – 1982.

18) T. Moto – oka, Overview to the Fifth Generation Computer System Project, Proc. 10th Ann Symp Computer Architecture, June 1983.

19) J.T. Schwartz, It's time to examine supercomputer options, SIAM News, v.17, May 1984.

20) A. Gottlieb and J.T. Schwartz, Networks and Algorithms for Very – Large – Scale Parallel Computation.

21) L.S. Haynes et al., A survey of Highly Parallel Computing, Computer, Jan. 1982.

22) S. Yalamanchili and J.K. Aggarwal, Reconfiguration Strategies for Parallel Architectures, Computer, Dec. 1985.

23) A. Gottlieb et al., The NYU Ultracomputer – Designing an MIMD Shared – Memory Parallel Computer, IEEETC, Febr. 1983.

24) K. Hwang and Z.W. Xu, Remps: A reconfigurable Multiprocessor for Scientific Supercomputing, Proc. Int. Conf. Parallel Processing, Aug. 1985.

2.2 Systolic dataflow structures.

25) H.T. Kung, Why Systolic Architectures, Computer, Jan. 1982.

26) A. V. Kulkarni and D.W.L. Yen, Systolic Processing and an Implementation for Signal and Image Processing, IEEETC, Oct. 1982.

27) F.H. Hsu, et al, LINC: The Link and Interconnection Chip, report CMU – CS – 84 – 159, May 1984.

28) H. Kung, The Structure of Parallel Algorithms, Advances in Computers , Academic Press, 1980.

29) A.L. Fisher, Implementation Issues for Algorithmic VLSI Processor Arrays, Ph.D. Thesis, CMU, Oct. 1984.

30) L. Snyder, Introduction to the Configurable Highly Parallel Computer, Computer, Jan. 1982.

31) A.L. Fisher, H.T. Kung and L.M. Monier, Architecture of the PSC: A Programmable Systolic Chip, Proc. Third Caltech Conf. on Very Large Scale Integration, Computer Science Press, March 1983.

32) H.T. Kung, Systolic Algorithms for the CMU Warp Processor, report CMU – CS – 84 – 158, Jan.1984.

33) Ph.L. Lehman, Systolic Arrays for Rapid Processing of Simple Database Transactions, report CMU – CS – 84 – 160, May 1984.

2.3 Dataflow computers

34) J.B. Dennis, Programming Generality, Parallelism and Computer Architecture, Information Processing 68, North Holland , 1969.

35) J.B. Dennis,C.K. Leung and D.P. Misunas, A Highly Parallel Processor Using A Data Flow Machine Language, Computation Structures Group Memo 134 – 2, MIT Jan. 1977.

36) J.B. Dennis et al, Modelling the Weather with a Data Flow Supercomputer IEEE Trans. Comp., July 1984.

37) C. Halatsis, G. Philokyprou and C. Verkerk, Prospects of Data Flow Computing in High Energy Physics, CERN, DD report DD/83/17, Sept. 1983.

38) J.B. Dennis, Data Flow Supercomputers, Computer Nov. 1980.

39) Arvind,V. Kathail and K. Pingali, A dataflow Architecture with Tagged Tokens, MIT report MIT/LCS/TM – 174, Sept. 1980

40) Arvind and R.E. Thomas, I – Structures: An Efficient Data Type for Functional Languages, MIT report MIT/LCS/TM – 178, Sept. 1980

41) Arvind, The U – Interpreter, Computer, Febr. 1982.

42) W. B. Ackerman, Data Flow Languages, Computer, Febr. 1982.

43) I. Watson and J. Gurd, A Practical Data Flow Computer, Computer, Febr. 1982.

44) J.R. Gurd, C.C. Kirkham and I. Watson, The Manchester Prototype Dataflow Computer, Comm. of the ACM, Jan. 1985.

45) W.W. Carlson and K. Hwang, Algorithmic Performance of Dataflow Multi – processors, Computer, Dec 1985.

46) J.C. Syre, The Data Flow Approach for MIMD Multiprocessor systems, PARALLEL PROCESSING SYSTEMS, J. Evans Editor, Cambridge University Press 1981.

47) M. Amamiya et al, A List – processing – oriented data flow machine architecture, AFIPS NCC 1982.

48) N. Takahashi and M. Amamiya, A data flow processor array system: design and analysis, Proc. 1983 Int. Symp. Comp. Architecture, 1983.

49) NEC Electronics, uDP7281 Image pipelined Processor, Preliminary data sheet Febr. 1985.

2.4 Reduction machines.

50) J. Backus, Can Programming Be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs, Comm. of the ACM, Aug. 1978.

51) J.R. Kennaway and M.R. Sleep, Novel architectures for declarative languages, Software and Microsystems, June 1983.

52) S.R. Vegdhal, A Survey of Proposed Architectures for the Execution of Functional Languages, IEEETC, Dec. 1984.

53) K.J. Berkling, A Computing Machine Based on Tree Structures, IEEETC, April 1971.

54) D.A. Turner, A New Implementation Technique for Applicative Languages Software – Practice and Experience, 1979, pp 31 – 49.

55) G.A. Mago, A Network of Microprocessors to Execute Reduction Languages, Part I and Part II, Int. J. of Computer and Information Sciences, 1979, pp 349 – 385 and 435 – 471.

56) R.M. Keller et al, A loosely – coupled applicative multi – processing system, AFIPS NCC 1979.

57) R. Keller and F.C.H. Lin, Simulated Performance of a Reduction – Based Multiprocessor, Computer, July 1984.

58) J. Darlington and M. Reeve, ALICE: a Multi – processor Reduction Machine for the Parallel Evaluation of Applicative languages, Proc. ACM Conf. Functional Languages, Oct. 1981, Portsmouth.

2.5 Logic and Inference machines.

59) R. Kowalski, Algorithm = Logic + Control, Comm. of the ACM, July 1979.

60) A. Colmerauer, PROLOG in 10 Figures, Comm. of the ACM, Dec. 1985.

61) M. R. Genesereth and M.L. Ginsberg, Logic Programming, Comm. ACM, Sept.1985.

62) D. H.D. Warren, An Abstract Prolog Instruction Set, SRI technical note 309, Oct. 1983.

63) E. Tick and D.H.D. Warren, Towards a Pipelined Prolog Processor, New Generation Computing, no 2, 1984.

64) T.P.Dobry, A. Despain and Y.N. Patt, Performance Studies of a Prolog Machine Architecture, Proc. 12th Int. Symp. Comp. Architecture, 1985.

65) K. Taki et al, Hardware design and implementation of the Personal Sequential Inference machine (PSI), Proc. Int. Conf. on Fifth Generation Computer Systems, 1984, ICOT.

66) R. Nakazaki et al , Design of a High – speed Prolog Machine, 12th Int. Symp. Comp. Architecture, 1985.

67) C.G. Ponder and Y.N. Patt, Alternative proposals for implementing Prolog concurrently and impications regarding their respective microarchitectures 11th Int. Symp. Comp. Architecture, 1984.

68) E.Y. Shapiro, Lecture notes on Bagel: a Systolic Concurrent Prolog Machine, ICOT technical memorandum #31.

69) S. Taylor et al, Logic Programming using parallel associative operators, 1984 int. Symp. Logic Programming.

70) T. Moto－oka et al, The architecture of a parallel inference engine － PIE －, Proc. Int. Conf. Fifth Generation Computer Systems, ICOT 1984.
K. Murakami et al, Research on Parallel Machine Architecture for Fifth－Generation Computer Systems, Computer, June 1985.