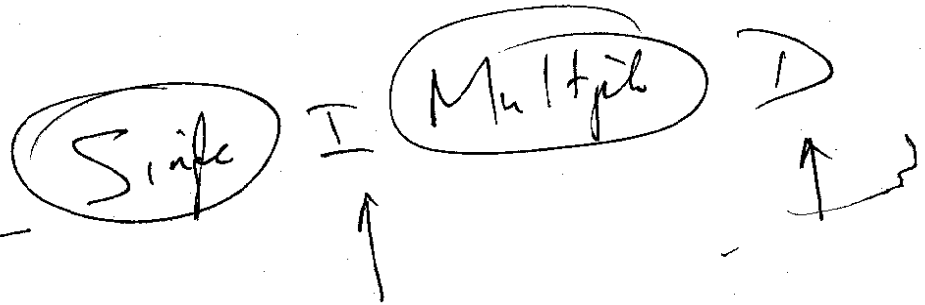
Single Thread Parallelism

Outline

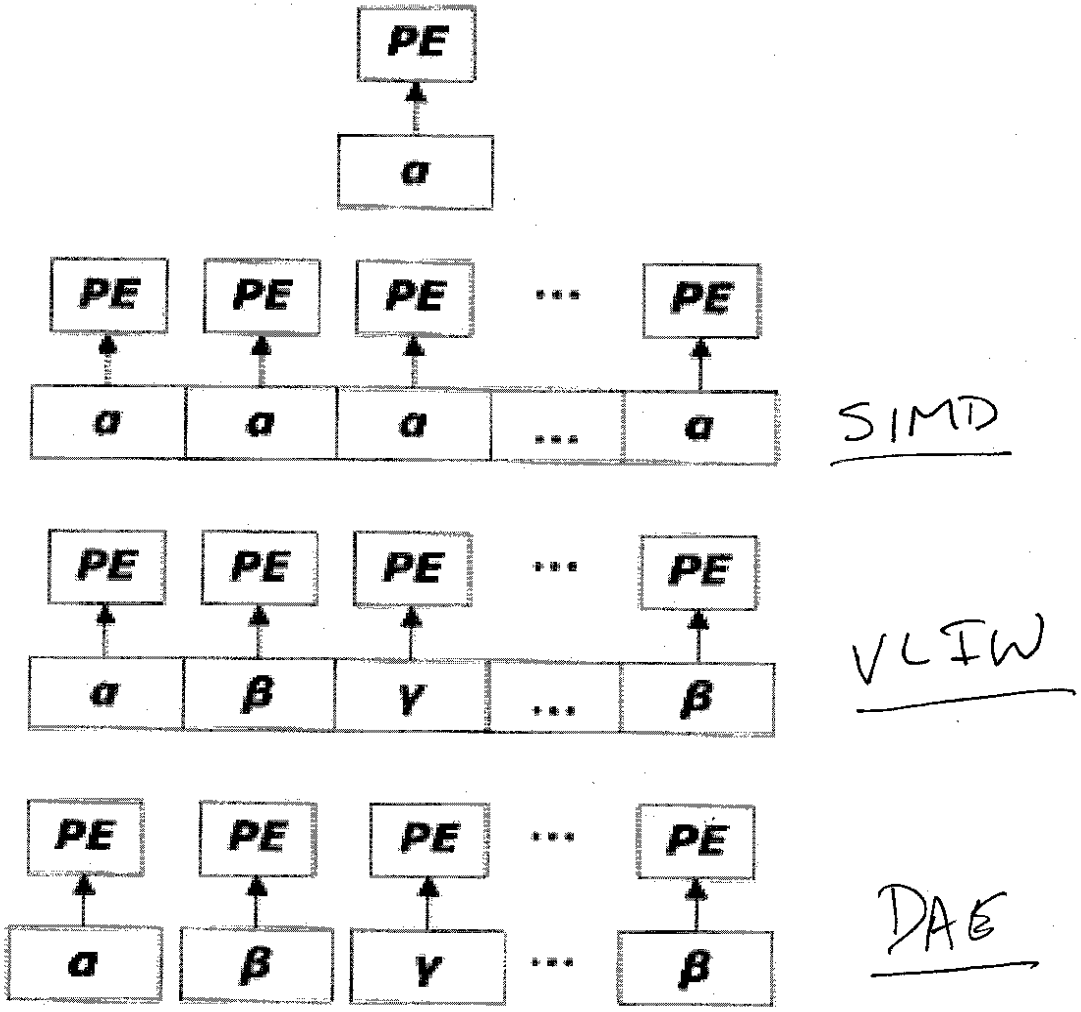


- ***Pipelining***
- ***SIMD (both vector and array processing)***
- ***VLIW***
- ***DAE***
- ***HPS***
- ***Data Flow***

Michael J. Flynn
SJCC 1967

MMX
SSE 1, 2, 3

HPS As Evolution



SIMD/MIMD

SISD The Typical Pentium-Pro, for example
MISD
SIMD *Array Processor, Vector Processor*
MIMD *Multiprocessor*

and, Note:

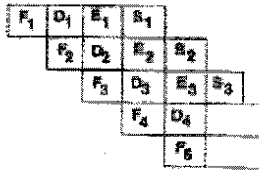
Pipelined SISD | SIMD

SYSTOLIC ARRAY

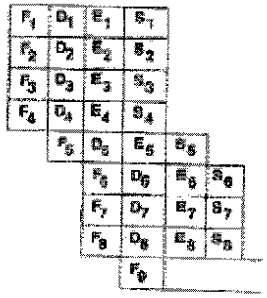
HT KUNG

Pipelining

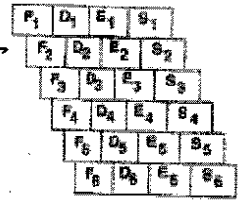
Pipelined:



Superscalar:



Superpipelined:



SIMD

Vector Processors, Array Processors

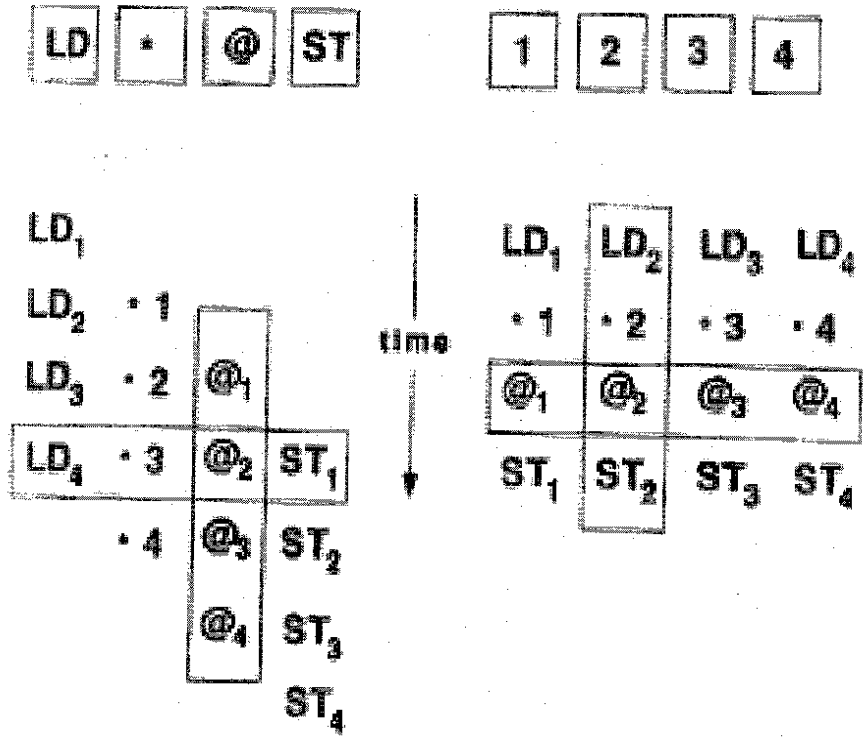
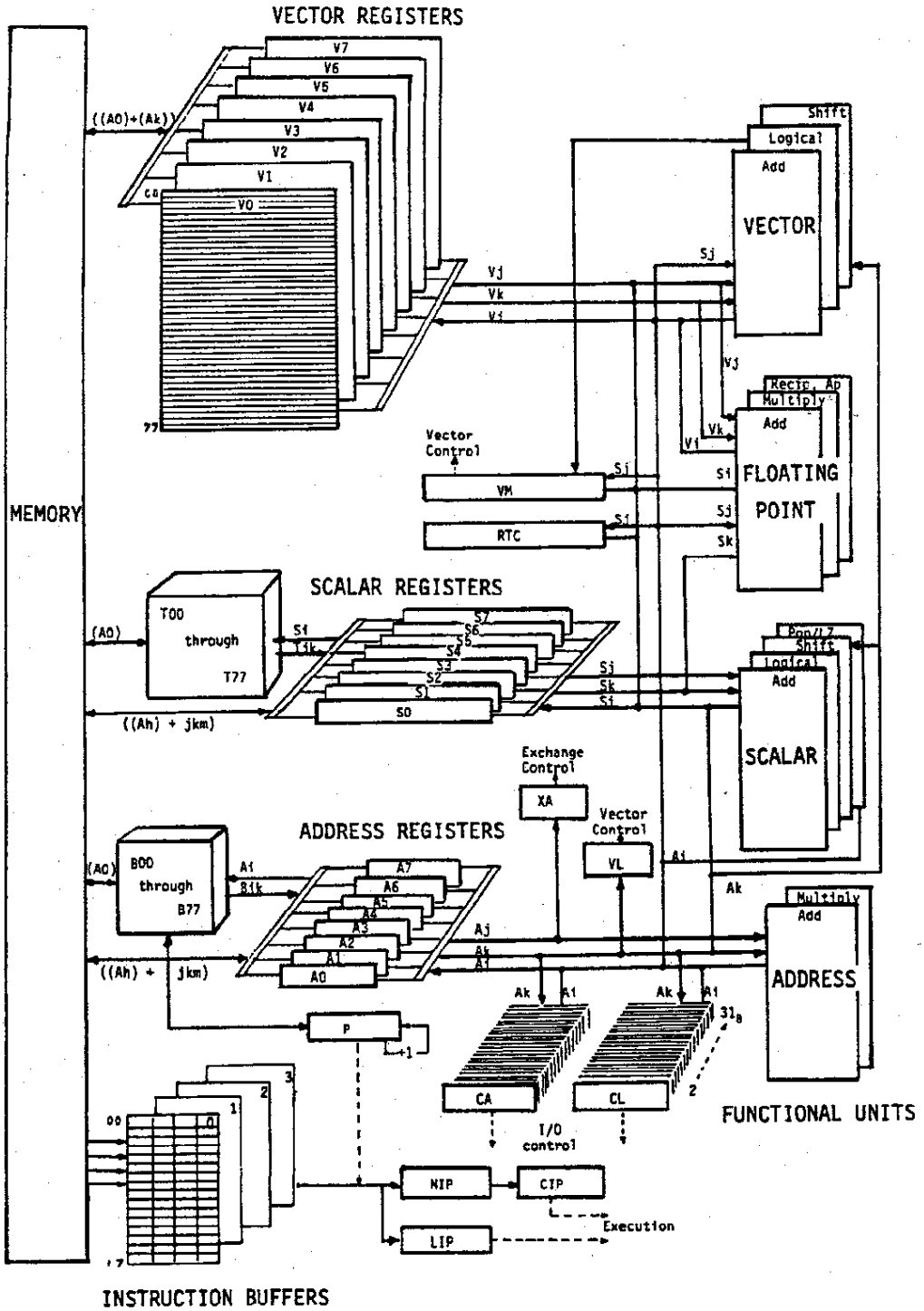


Fig. 5. Block diagram of registers.



An example: Vector processing

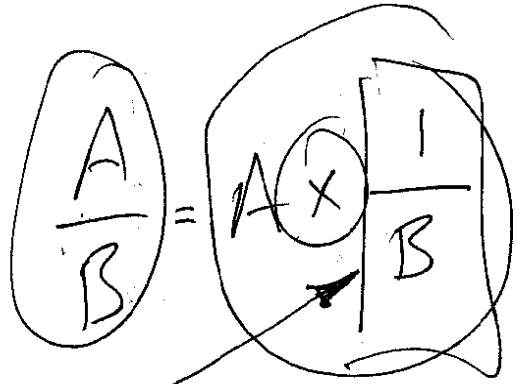
- **The scalar code:**

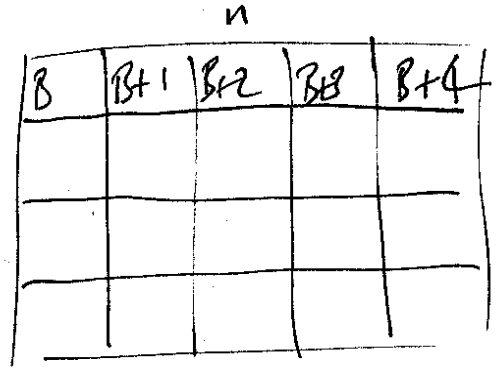
```
for i=1,50
A(i) = (B(i)+C(i))/2;
```

- **The vector code:**

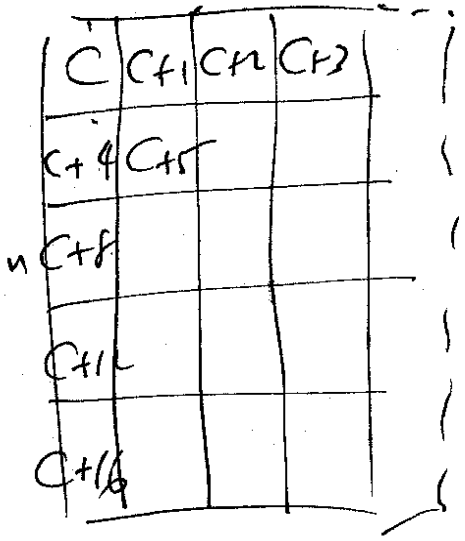
```
lvs 1
lvl  50
vld  V0,B
vld  V1,C
vadd V2,V0,V1
vshf  V3,V2,1
vst   V3,A
```

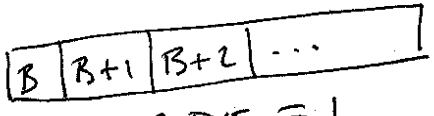
RECIPROCAL APPROXIMATE (FASTER THAN DIVIDE)



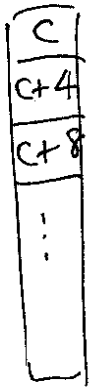


ROW MAJOR ORDER

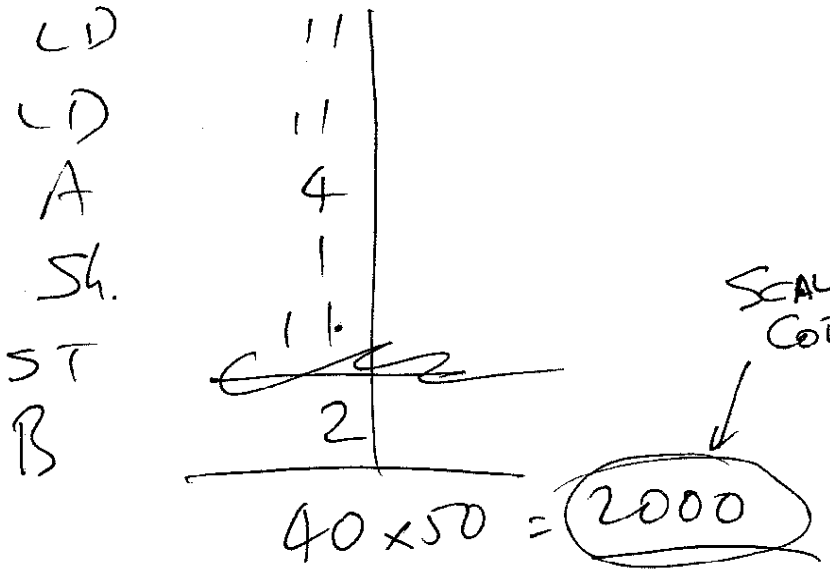


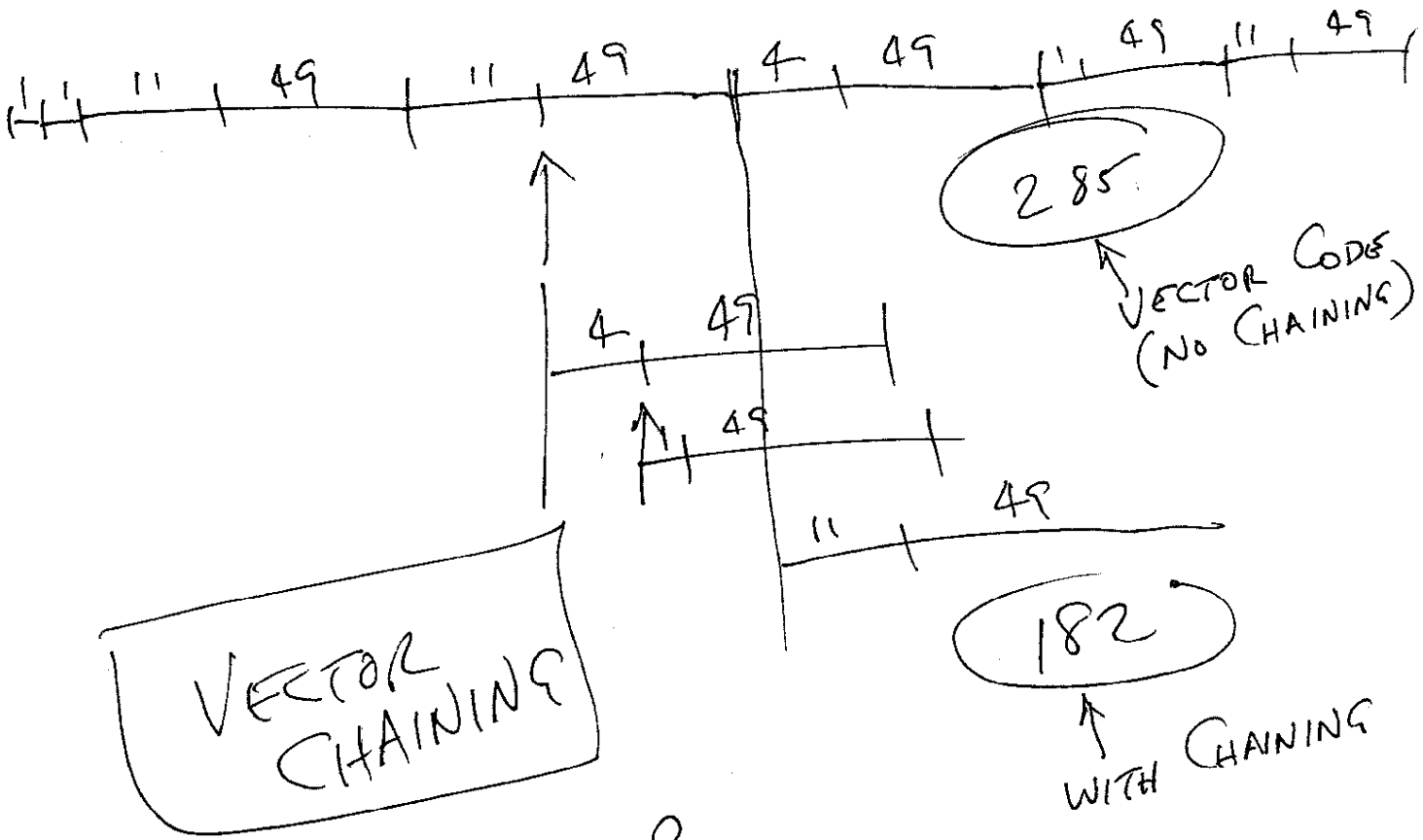


STRIDE = 1

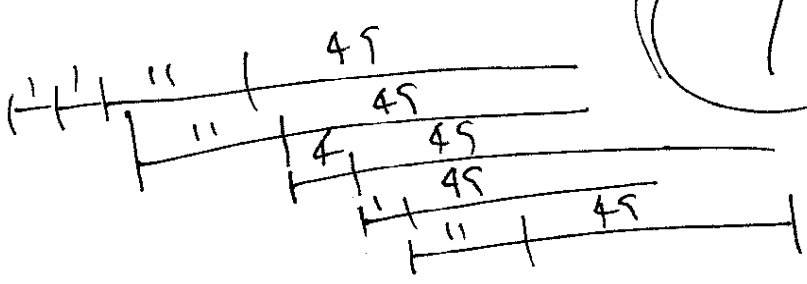


STRIDE = 4


LD 11
LD 11
A 4
SH. 1
ST 11
B 2
SCALAR CODE
40 × 50 = 2000

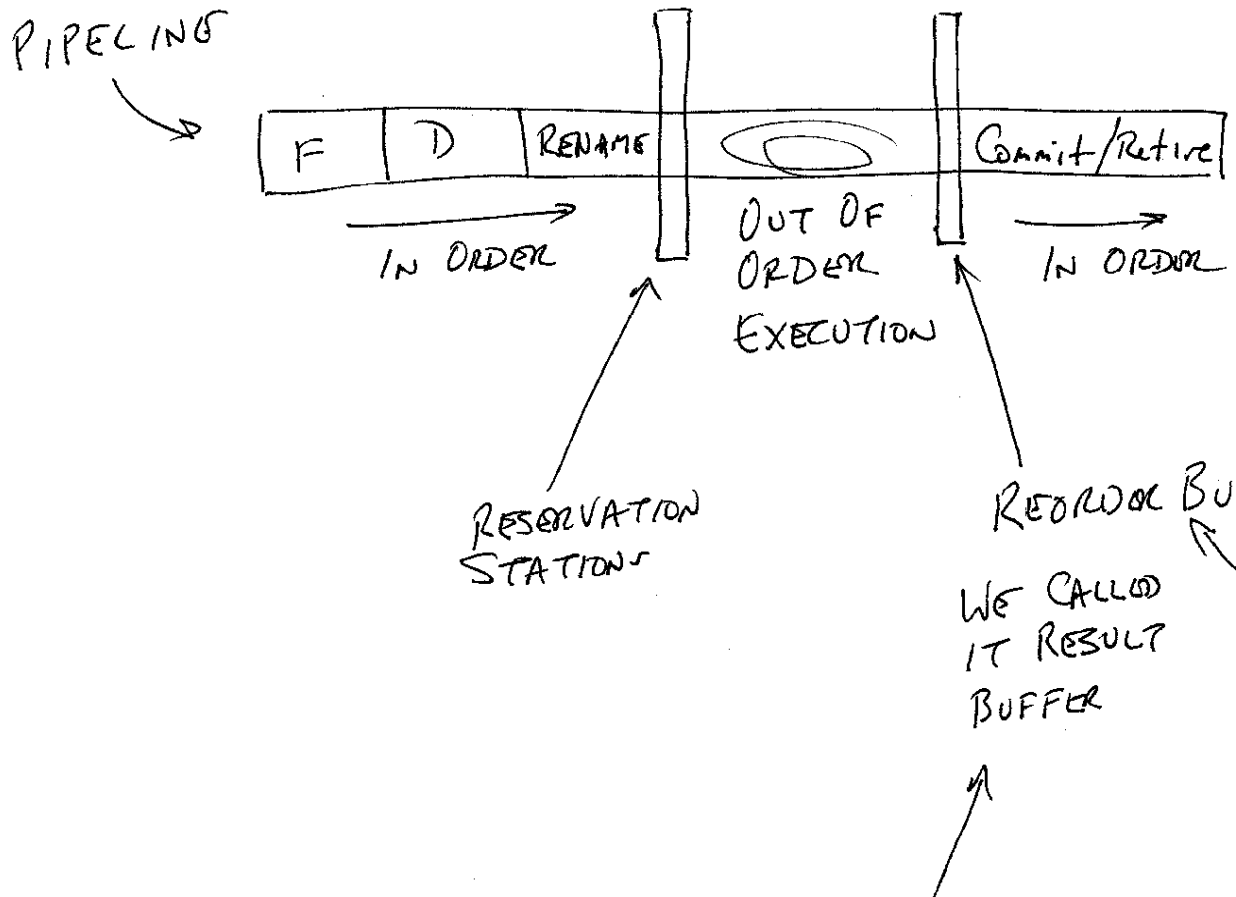

285
VECTOR CODE (NO CHAINING)
VECTOR CHAINING
182
WITH CHAINING

CRAY-XMP


79
WITH 2 LOAD, 1 STORE PIPE

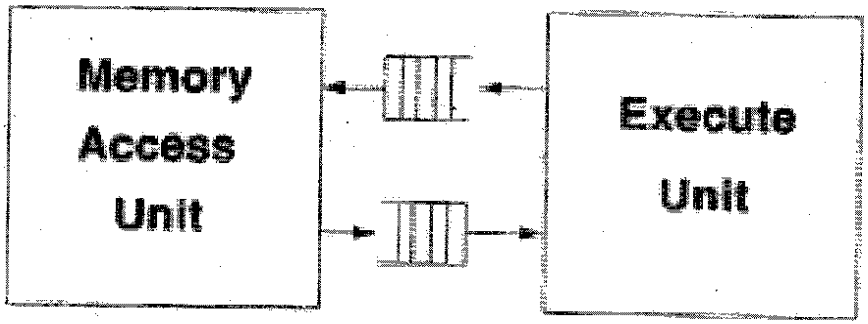
Vector processing example (continued)

- ***Scalar code (loads take 11 clock cycles):*** 2000
- ***Vector code (no vector chaining):*** 285
- ***Vector code (with chaining):*** 182
- ***Vector code (with 2 load, 1 store port to memory):*** 79


PIPELINE
F
D
RENAME
COMMIT/RETIRE
IN ORDER
OUT OF ORDER EXECUTION
IN ORDER
RESERVATION STATIONS
REORDER BUFFER
QUICK SOHI
WE CALLED IT RESULT BUFFER
INSTRUCTIONS RETIRED IN ORDER MEANS WE CAN RECOVER THE STATE OF THE MACHINE JUST BEFORE THE INSTRUCTION.
∴ PRECISE EXCEPTIONS
NOT TIME for Tomasulo AND 360/91

Early Form of Decoupled - Access/Execute

IN ORDER



* Andrew Plezskun, Univ. of Illinois

SMA

* James E. Smith, Univ of Wisconsin

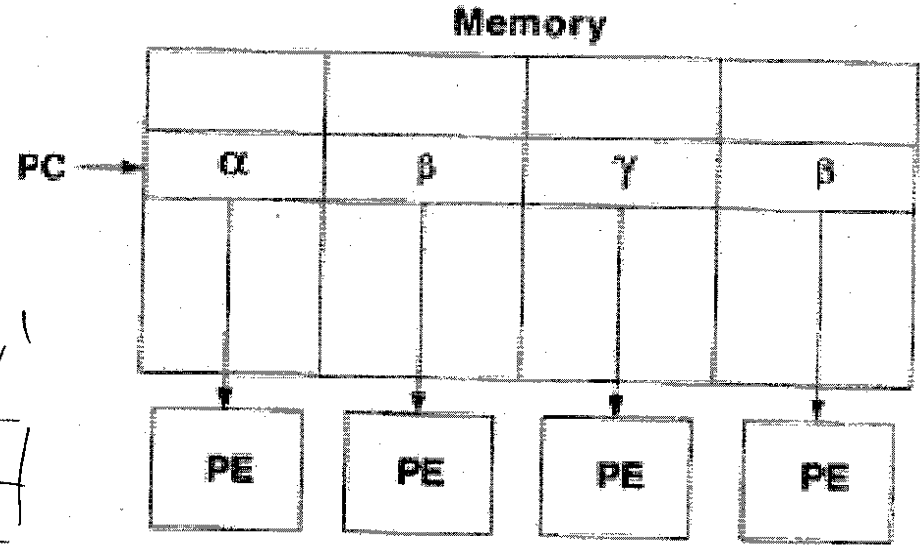
DAE

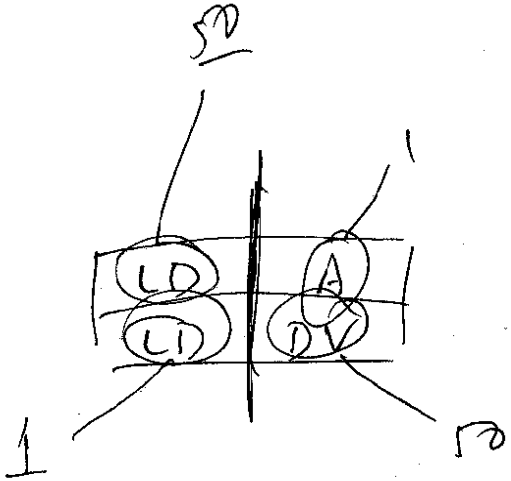
VLIW (JOSH FISHER)

MULTIFLOW TRACE 7

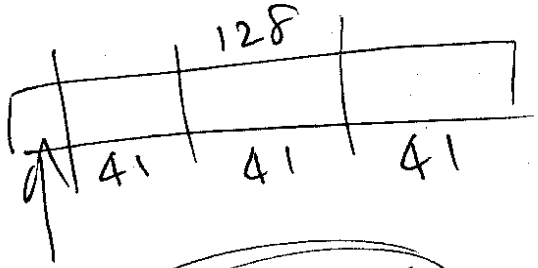
BOB RAU CYDROME CYDRA-5

- ★ **Static Scheduling**
  - **Everything in lock step**
  - **Trace Scheduling**
    - Good if Highly Skewed
- ★ **Generic Model**

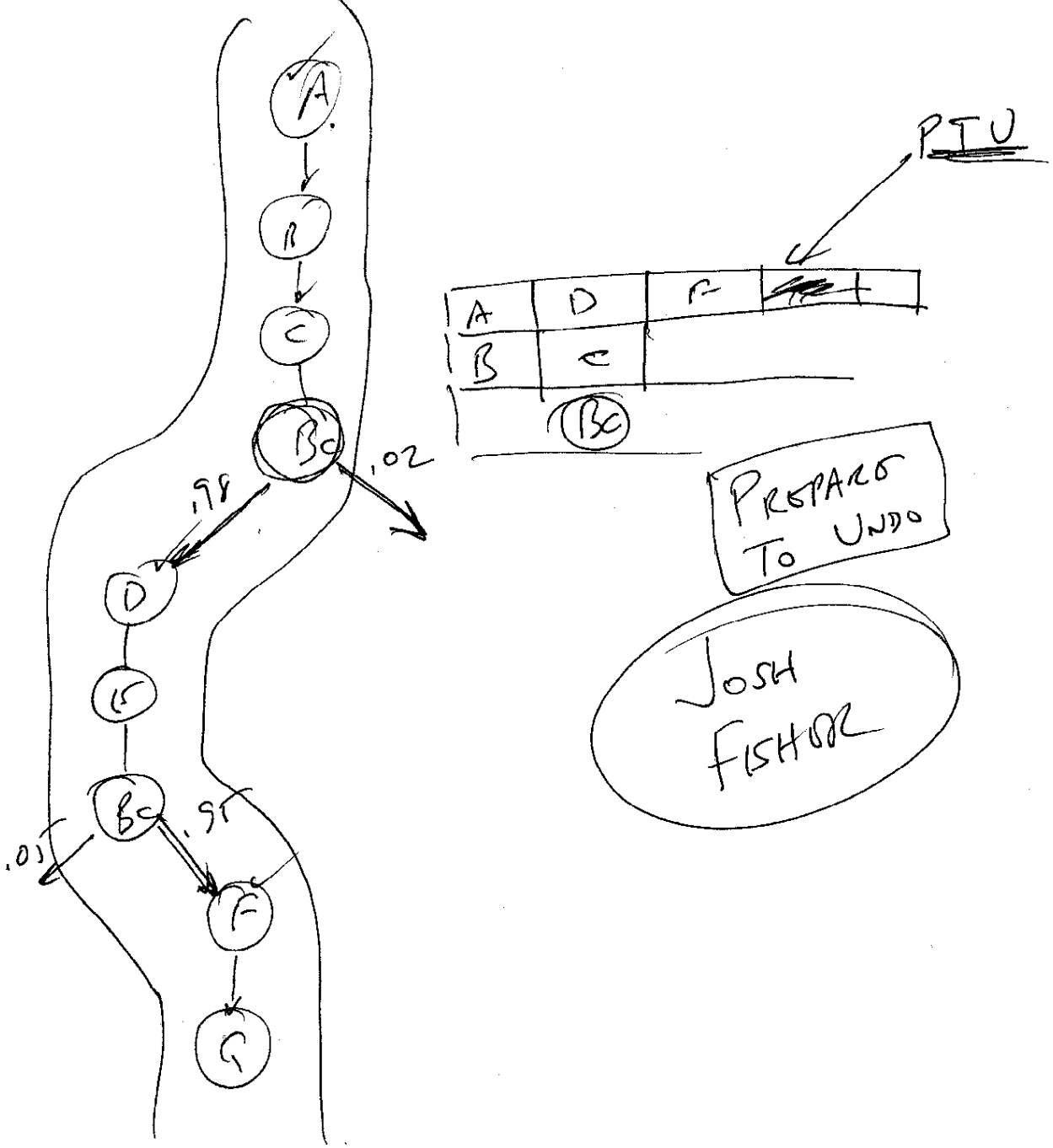




EPIC



TEMPLATE


A
B
C
BC
.98
.02
D
E
BC
.01
.99
F
G
PTU
A
D
B
C
BC
PREPARE TO UNDO
JOSH FISHER

The HPS Paradigm

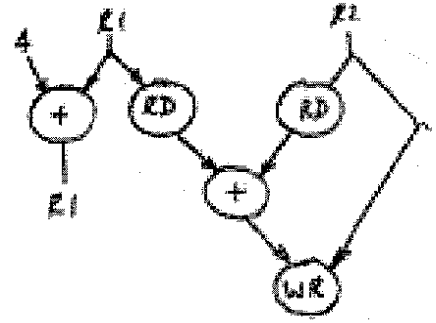
- **Incorporated the following:**
  - Aggressive branch prediction
  - Speculative execution
  - Wide issue
  - Out-of-order execution — Tomasulo Algorithm
  - In-order retirement
- **First published in Micro-18 (1985)**
  - Patt, Hwu, Shebanow: Introduction to HPS
  - Patt, Melvin, Hwu, Shebanow: Critical issues

HPS

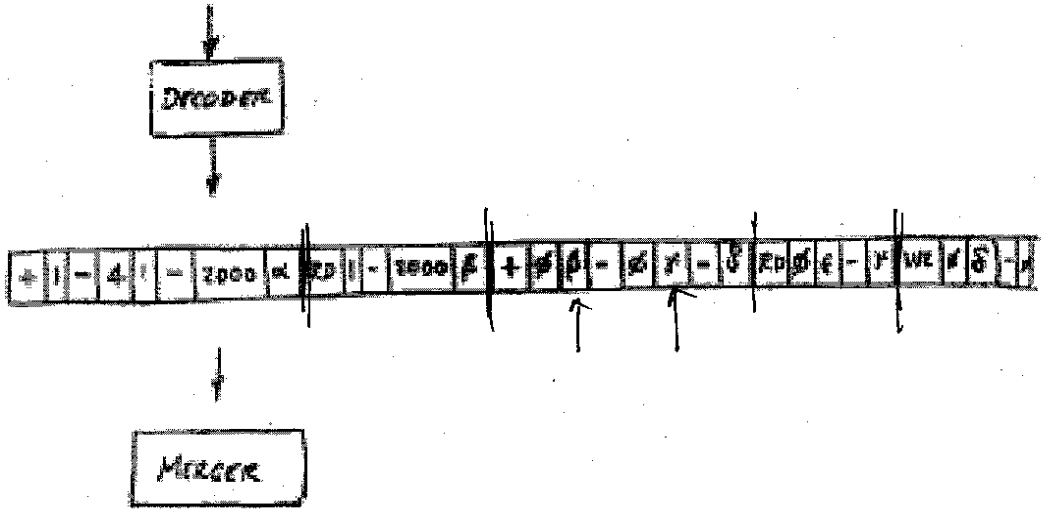
(Restricted Data Flow)

For Example, The VAX Instruction:

ADDL2 (R1)+, (R2)



VAX Instruction

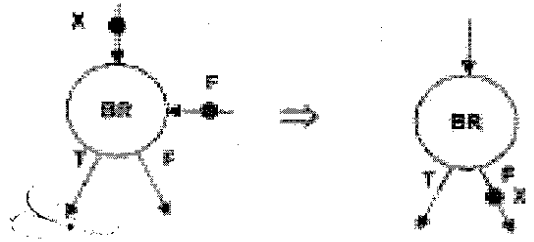


The Firing Rule:

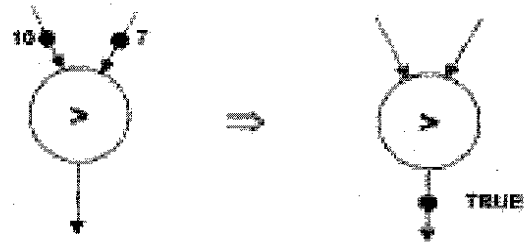
When all Inputs Have Tokens

(Note: Safe vs. Queues)

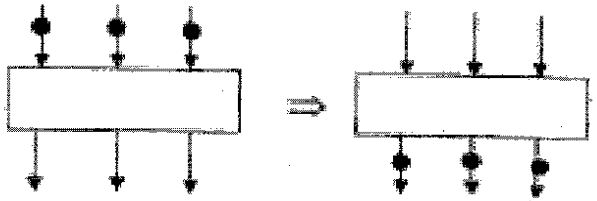
**Conditional*

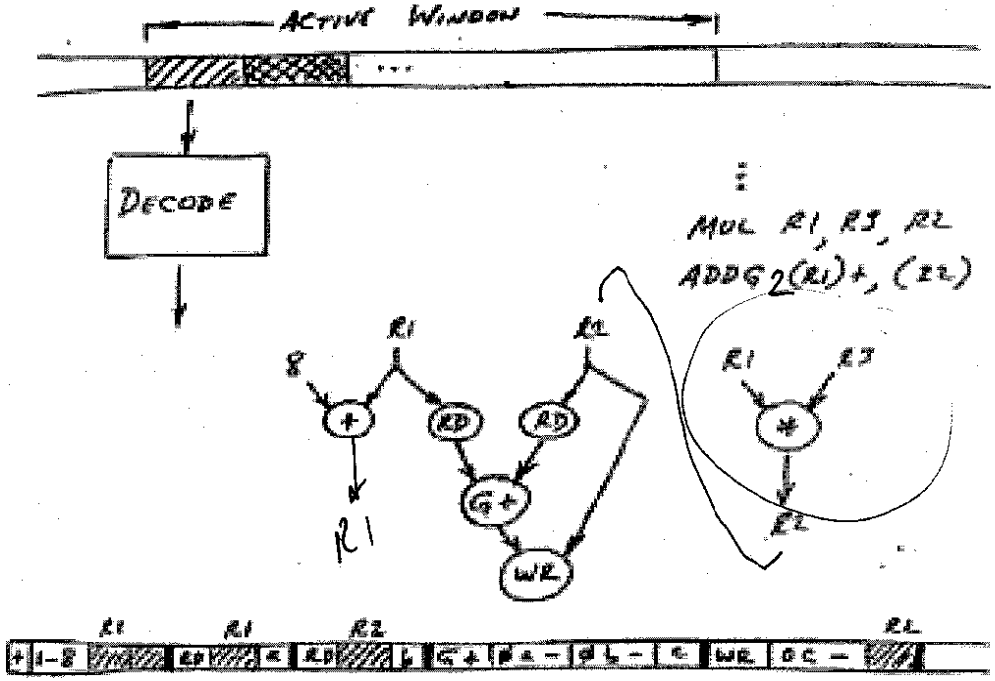


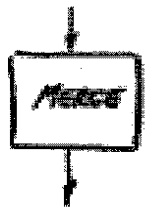
**Relational*



**Barrier Synch*





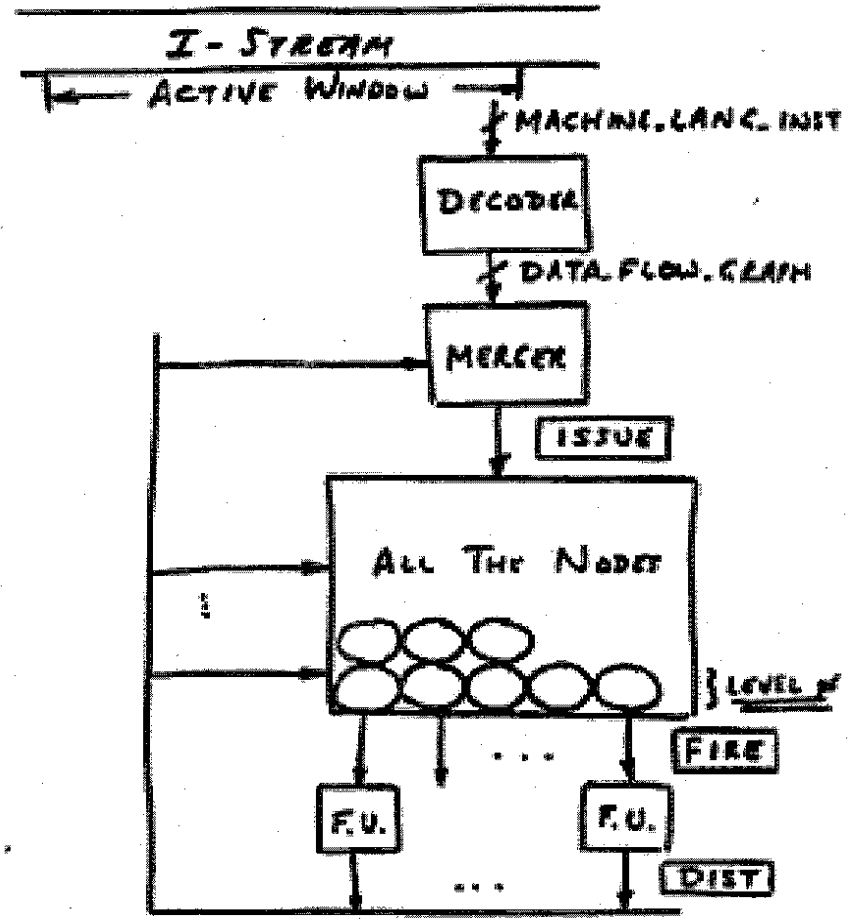


è la Tomasulo

R1	1	—	2000
R2	0	ok	—
R3	1	—	100

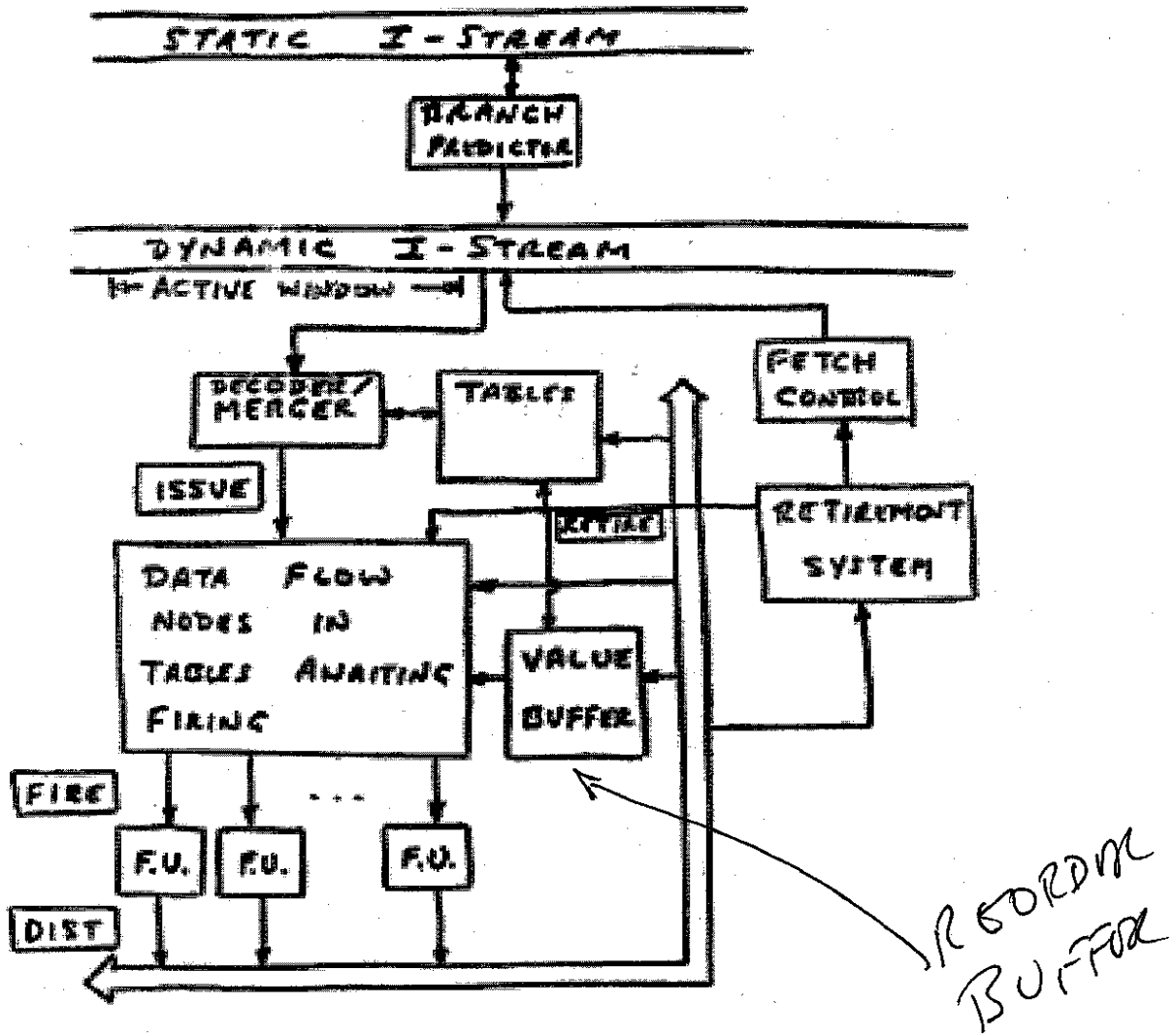
R1	ok	8	—
R2	ok	ok	—
R3	1	—	100

HPS - What Is It ?



* Restricted Data Flow

HPS


STATIC I-STREAM
BRANCH PREDICTOR
DYNAMIC I-STREAM
ACTIVE WINDOW
DECODER/ MERGER
TABLES
FETCH CONTROL
ISSUE
BUFFER
RETIREMENT SYSTEM
DATA FLOW NODES IN TABLES AWAITING FIRING
VALUE BUFFER
FIRE
F.U.
F.U.
F.U.
DIST
REORDER BUFFER

(RAW) HAZARD

(A+B) * C

FLOW DEPENDANCY

DAVE KUCK

STATIC ASSIGNMENT

$x = x + 1$

$y = x + 1$

SINGLE - ASSIGNMENT

~~y~~ = X + 1

DEPENDENCIES (INTRODUCED BY DAVE KUCK)

(1) FLOW DEPENDANCY : (A+B) * C

YOU HAVE TO ADD A, B BEFORE YOU MULTIPLY BY C.

i.e., WRITE (A+B) BEFORE READ

RAW : READ AFTER WRITE HAZARD

(2) ANTI-DEPENDANCY: ADD R1, R2, R3
ADD R2, R3, R4

YOU HAVE TO READ R2 BEFORE YOU OVERWRITE IT

WAR : WRITES AFTER READ HAZARD

(3) OUTPUT DEPENDANCY : ADD R1, R2, R3
ADD R1, R2, R1

WAW : WRITE AFTER WRITE HAZARD

Data Flow

- ***Data Driven execution of inst-level Graphical code***
  - ***Nodes are operators***
  - ***Arcs are I/O***
- ***Only REAL dependencies constrain processing***
  - ***Anti-dependencies don't (write-after-read)***
  - ***Output dependencies don't (write-after-write)***
  - ***NO sequential I-stream (No program counter)***
- ***Operations execute ASYNCHRONOUSLY***
- ***Instructions do not reference memory***
  - ***(at least memory as we understand it)***
- ***Execution is triggered by presence of data***

A Unit of Computation:

The Data Flow Node

OR,

*	R	ARG1	R	ARG2	Dest. Of Result

The Operation
(In Larger Granularity Systems, "The Compound Function")

Fires When Ready

An Example Data Flow Program:

***Factorial* (Done, Iteratively)**