Computer Architecture:
Fundamentals, Tradeoffs, Challenges

Chapter 12: Single Thread Parallelism

Yale Patt
The University of Texas at Austin

Austin, Texas
Spring, 2023

Outline

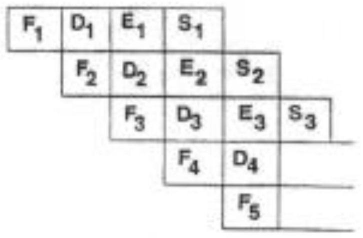
- ***Granularity of Concurrency***
- ***Pipelining***
- ***SIMD (both vector and array processing)***
- ***VLIW***
- ***DAE***
- ***HPS***
- ***Data Flow***

Granularity of Concurrency

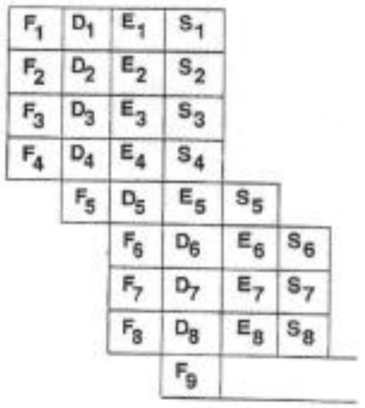
- *Intra-instruction (pipelining)*
- *Parallel instructions in a single thread (SIMD, VLIW)*
- *Tightly-coupled (multiprocessor)*
- *Loosely-coupled (multi-computer network)*

Pipelining

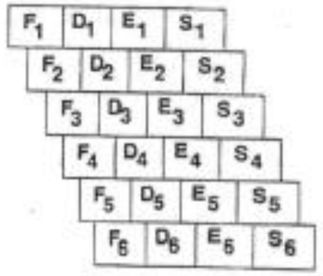
Pipelined:



Superscalar:



Superpipelined:



SIMD/MIMD

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

and, Note:

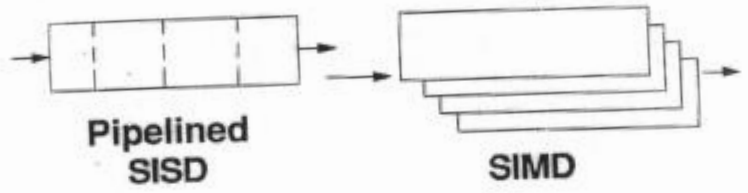
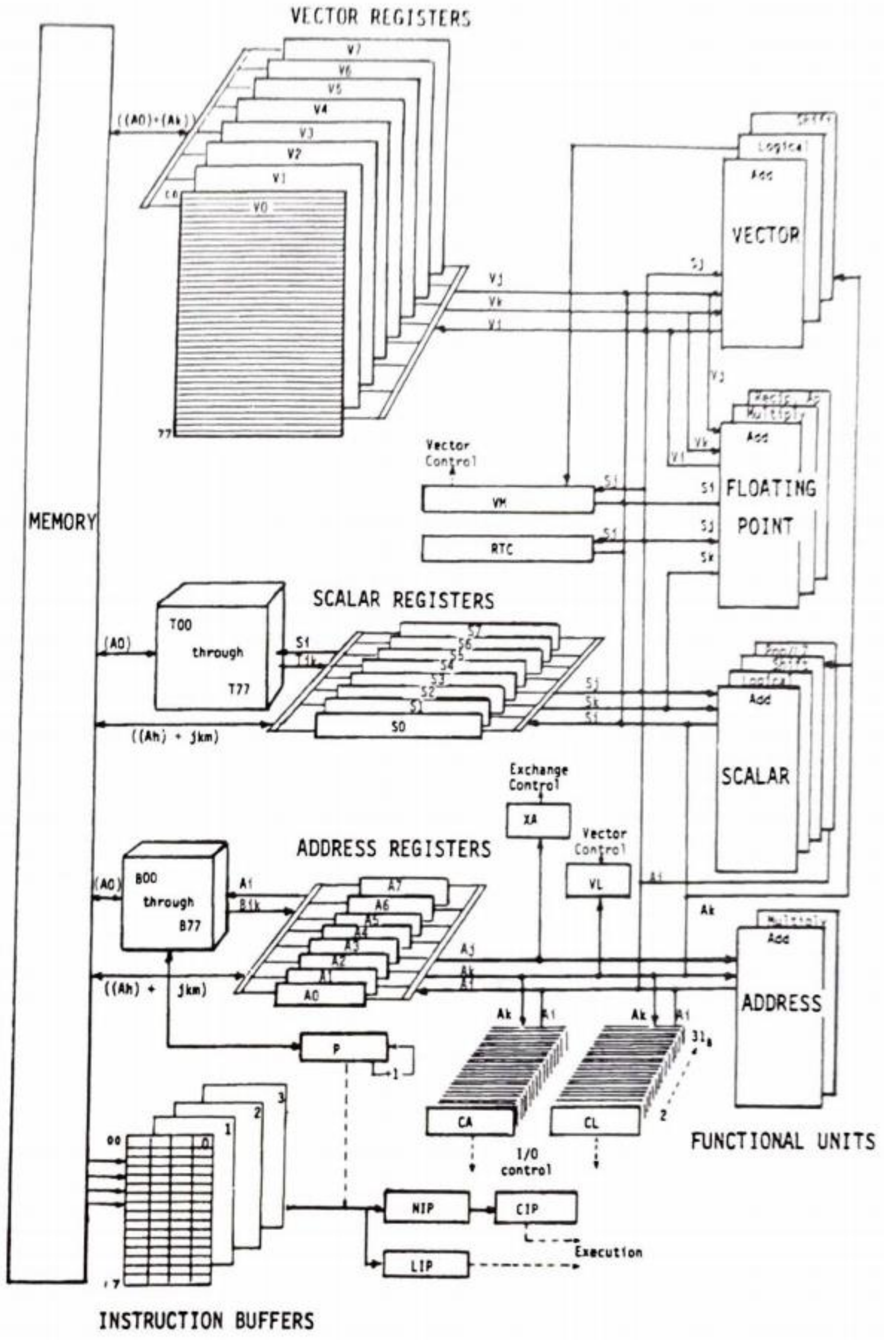


Fig. 5. Block diagram of registers.



Vector processing example

- ***The scalar code:***

  for i=1,50

  A(i) = (B(i)+C(i))/2; Vectorizable!

- ***The vector code:***

```
lvs 1                  ; load vector stride
lvl  50                ; load vector length
vld  V0,B              ; load V0 from memory, starting at address B
vld  V1,C              ; load V1 from memory, starting at address C
vadd V2,V0,V1          ; V2 < --- V0 + V1
vshfr  V3,V2,1         ; V3 < --- V2 divided by 2 (shift right one bit)
vst   V3,A             ; store V3 to memory, starting at address A
```

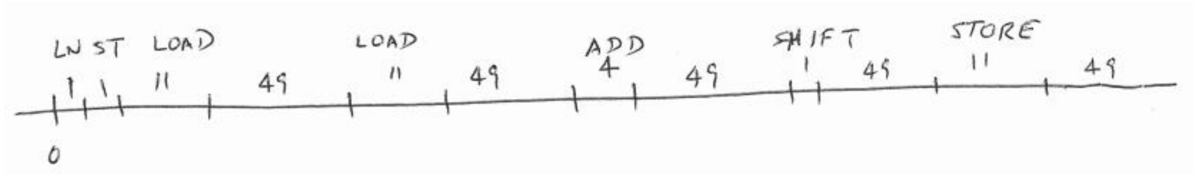
Vector processing example (continued)

- ***Baseline: with a Scalar Processor:***
  - ***Loads/Stores take 11 cycles***
  - ***Add takes 4 cycles***
  - ***Shift takes 1 cycle***
  - ***Iteration Control takes 2 cycles***

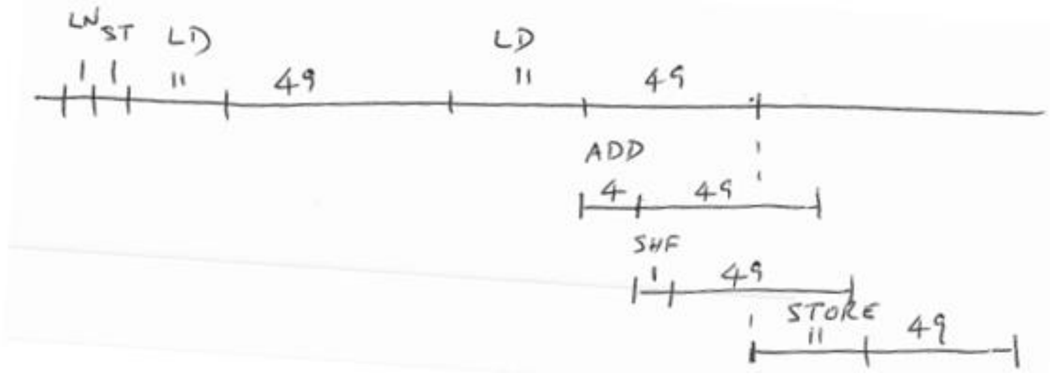
- ***50 iterations of (LD, LD, Add, Shift, Store, Iteration Ctl)***
  - ***50 x (Load, Load, Add, Shift, Store, Iteration_Ctl)***
  - ***50 x (11 + 11 + 4 + 1 + 11 2) = 50 x 40 = 2000 clock cycles***

Vector processing example (continued) Vector Processor Timing

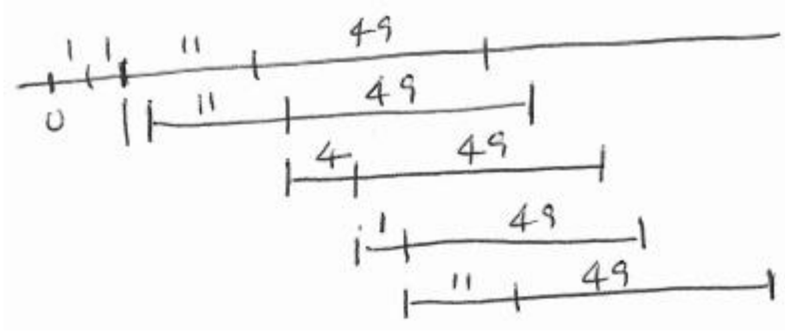
- *Vector code (no vector chaining):* **285 clock cycles**



- *Vector code (with chaining):* **182 clock cycles**

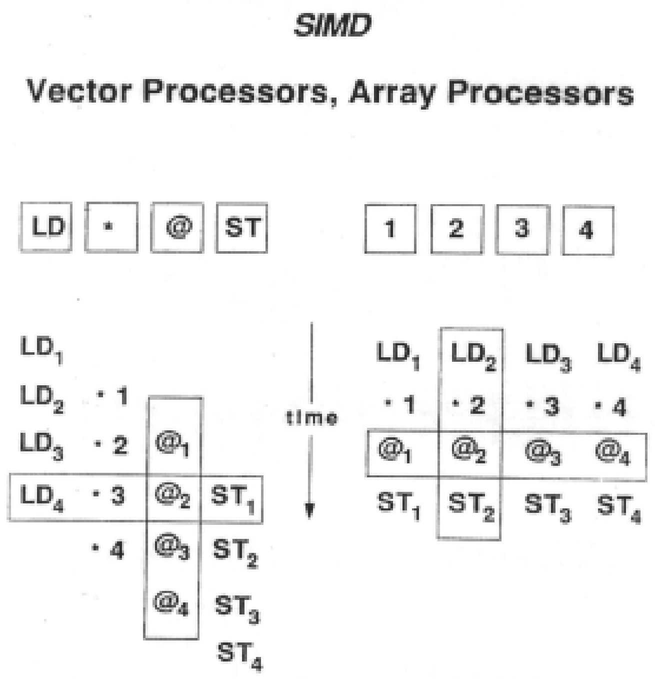


- *Vector code (with 2 load, 1 store port to memory):*



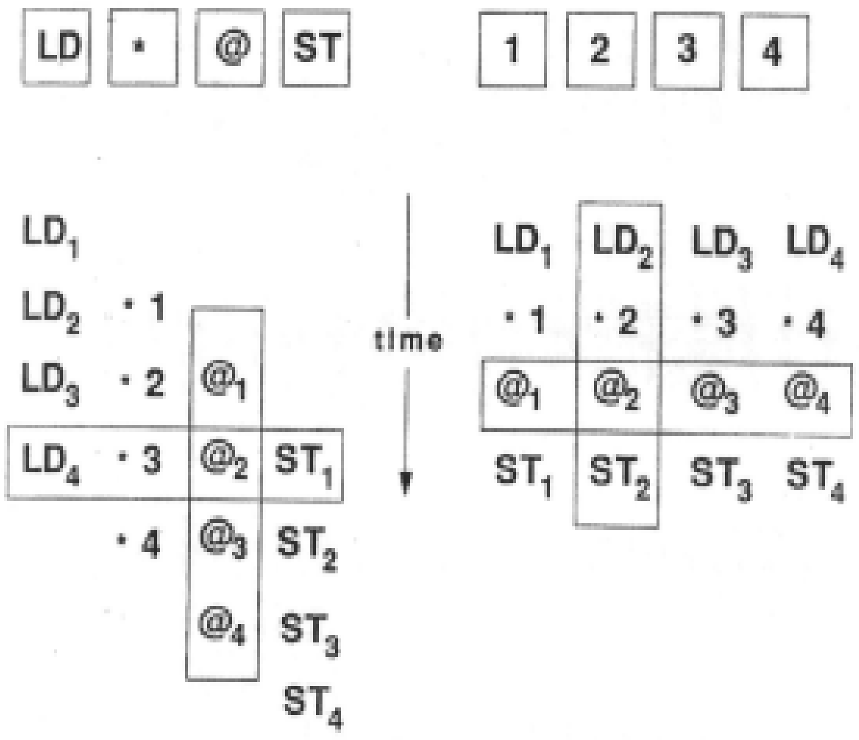
79 clock cycles

Important to note that SIMD can be either Vector Processors or Array Processors



SIMD

Vector Processors, Array Processors

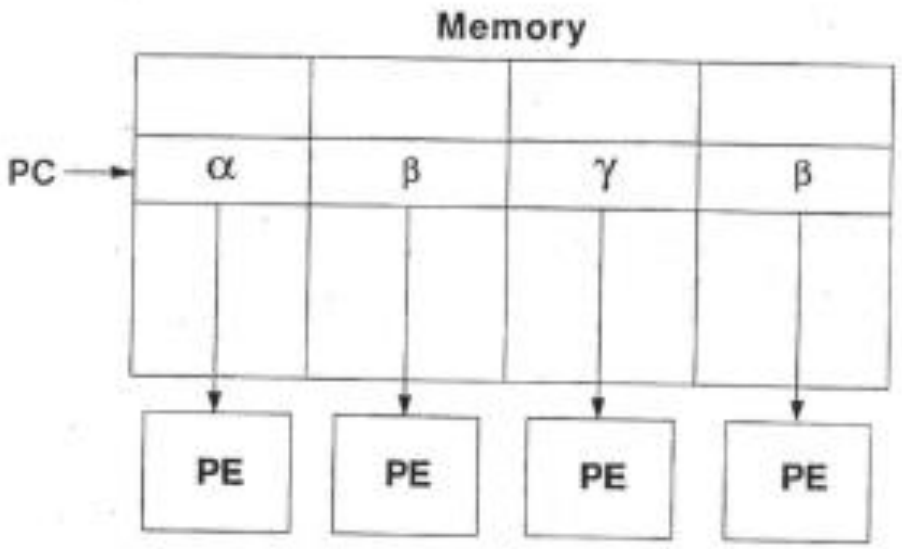


VLIW

* Static Scheduling

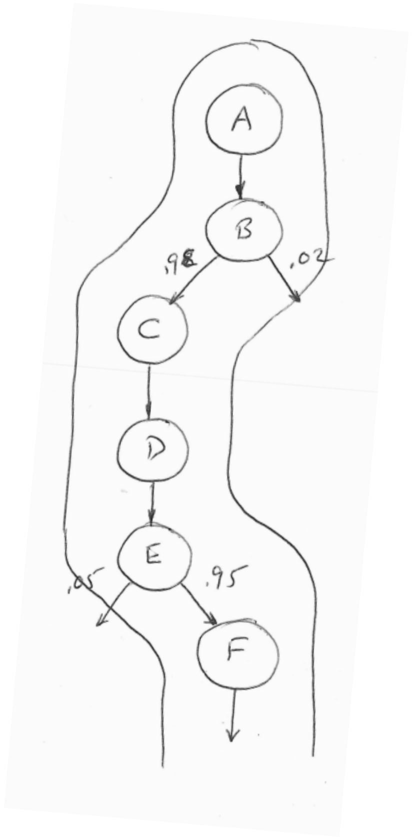
- Everything in lock step
- Trace Scheduling

* Generic Model

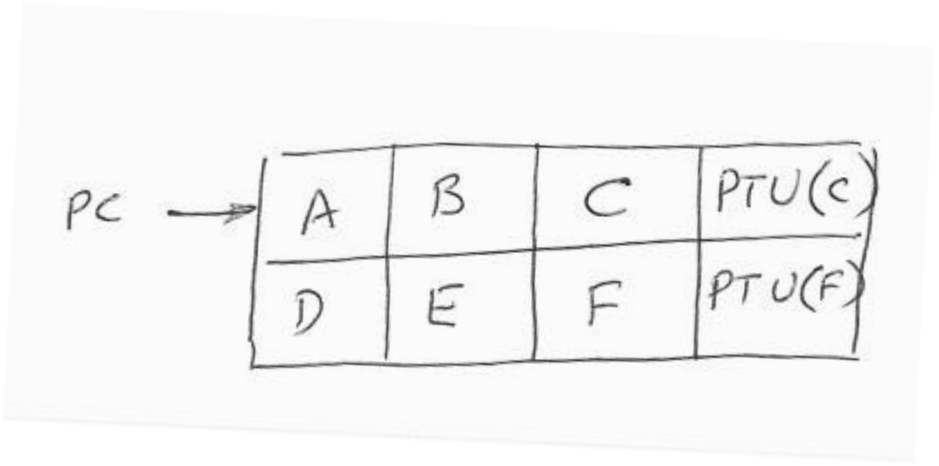


Trace Scheduling

Instruction flow

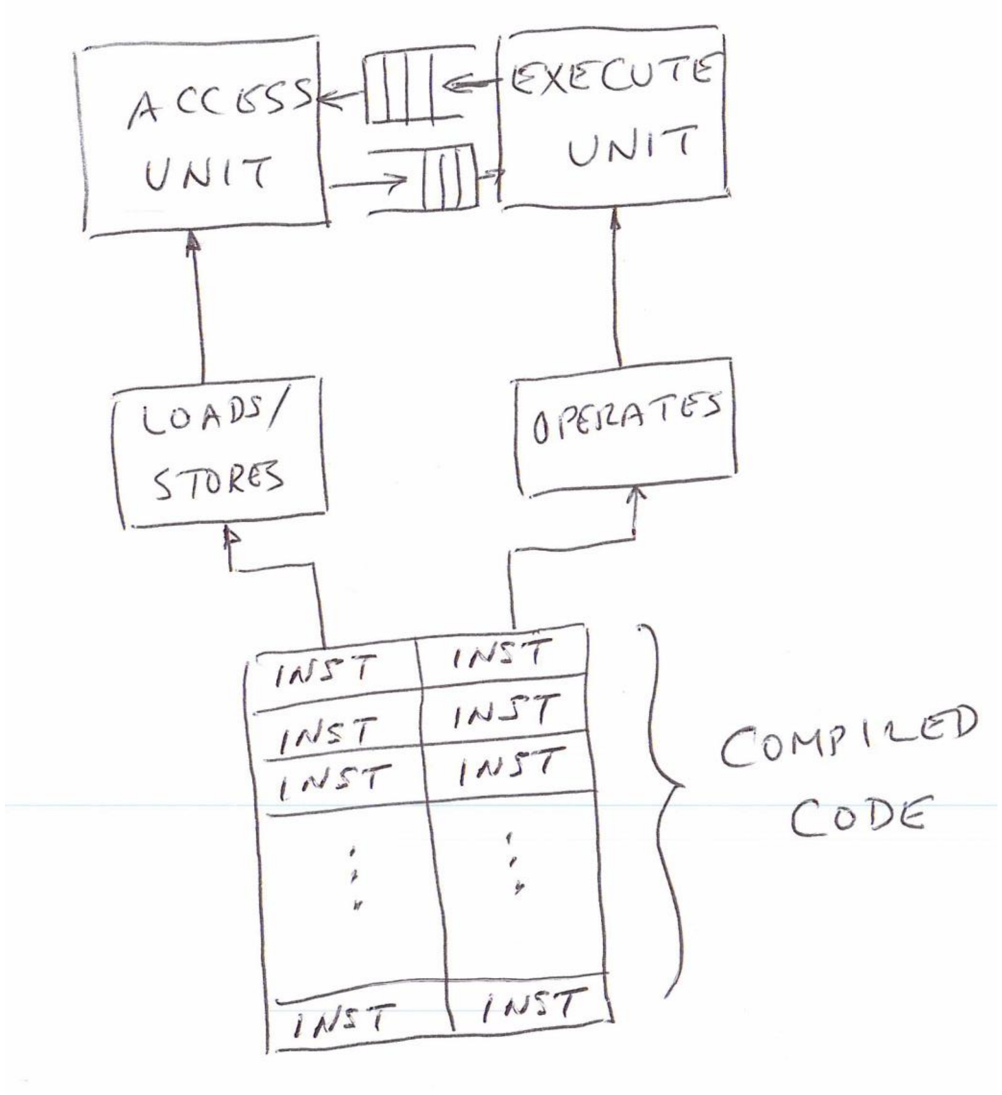


VLIW Code

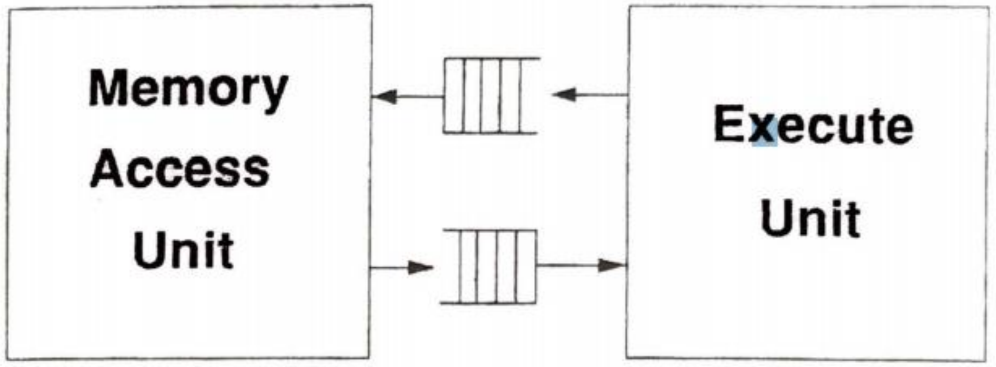


A	B	C	PTU(C)
D	E	F	PTU(F)

The Decoupled Access Execute (DAE) Paradigm
(by Andrew Pleszkun (SMA), Jim Smith (DAE)



Early Form of Decoupled - Access/Execute



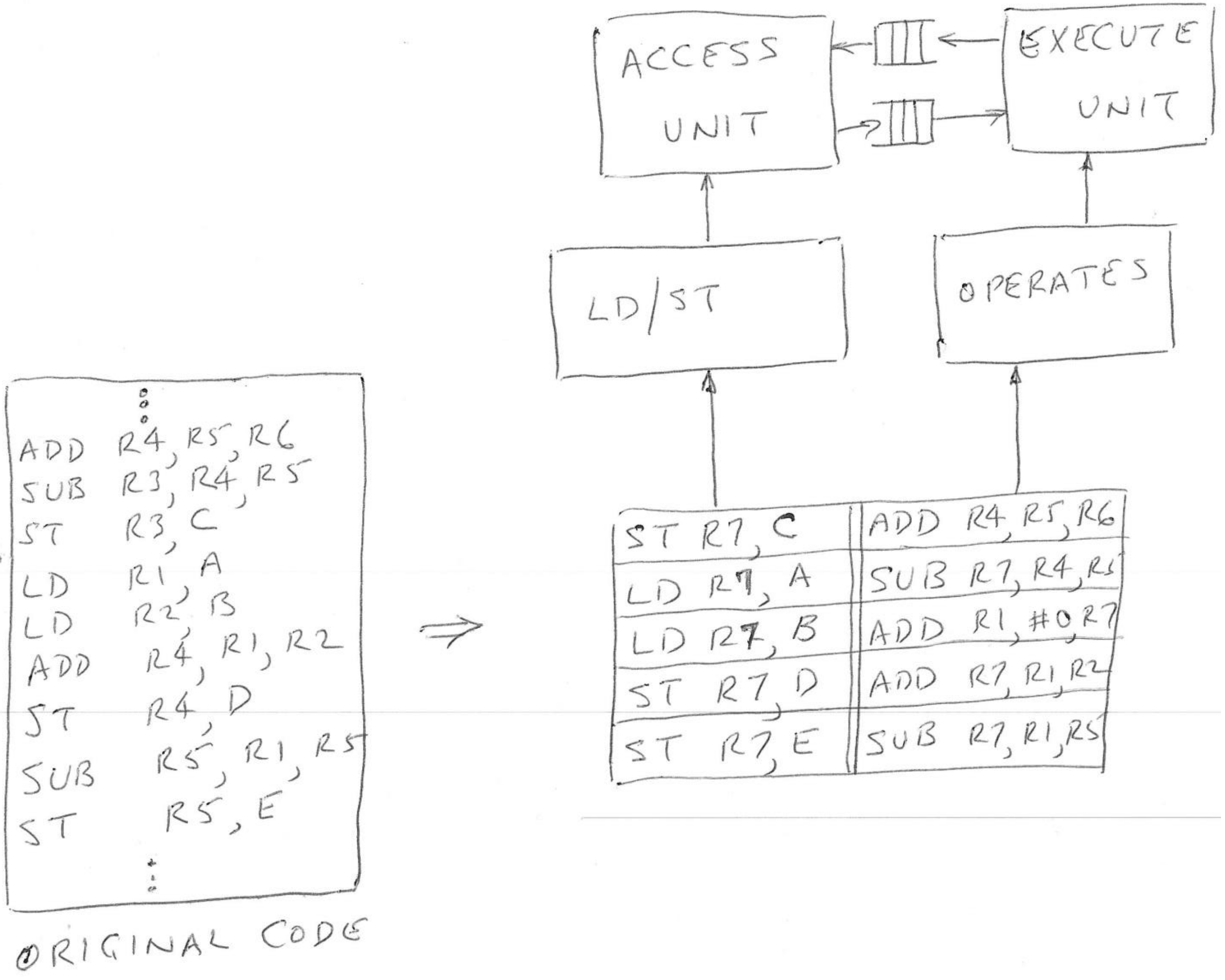
- ★ Andrew Plezskun, Univ. of Illinois

  SMA

- ★ James E. Smith, Univ of Wisconsin

  DAE

DAE (an example)



```
⋮
ADD  R4, R5, R6
SUB  R3, R4, R5
ST   R3, C
LD   R1, A
LD   R2, B
ADD  R4, R1, R2
ST   R4, D
SUB  R5, R1, R5
ST   R5, E
⋮
```

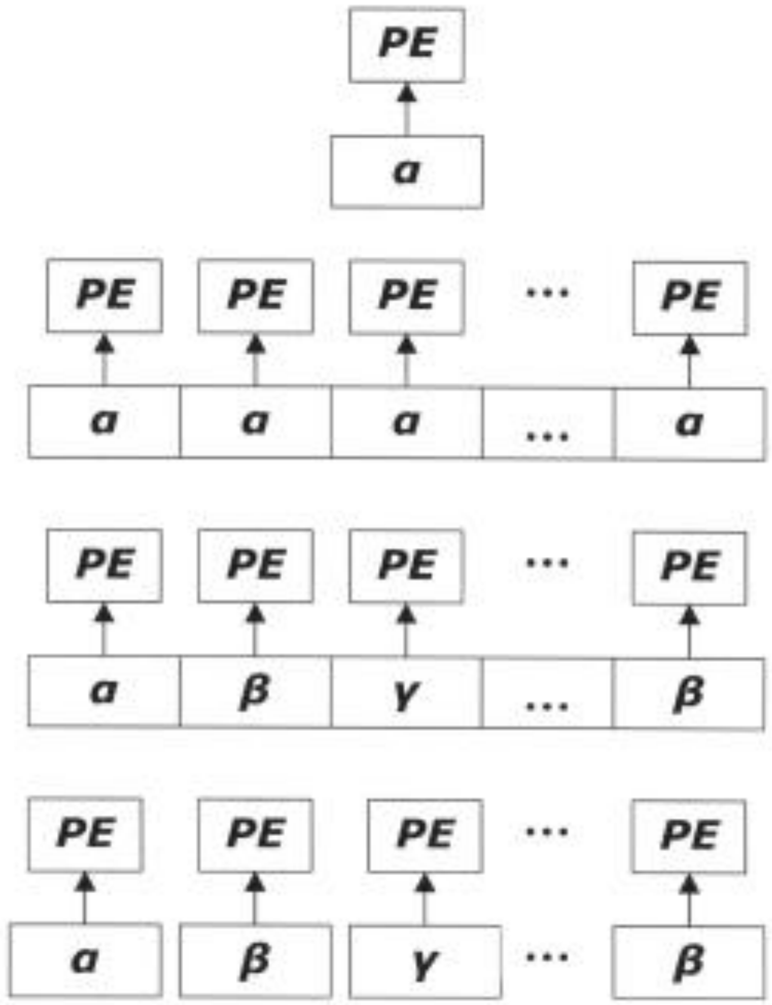
⇒

ST R7, C	ADD R4, R5, R6
LD R7, A	SUB R7, R4, R5
LD R7, B	ADD R1, #0, R7
ST R7, D	ADD R7, R1, R2
ST R7, E	SUB R7, R1, R5

Timing of the DAE Example

3	1
4	2
5	5
7	6
8	7

HPS As Evolution



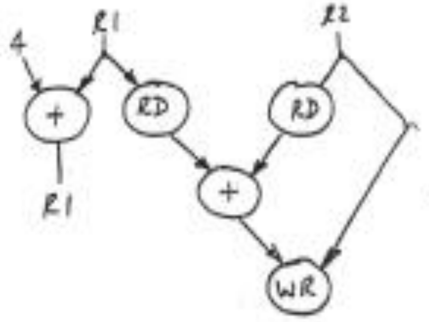
The HPS Paradigm

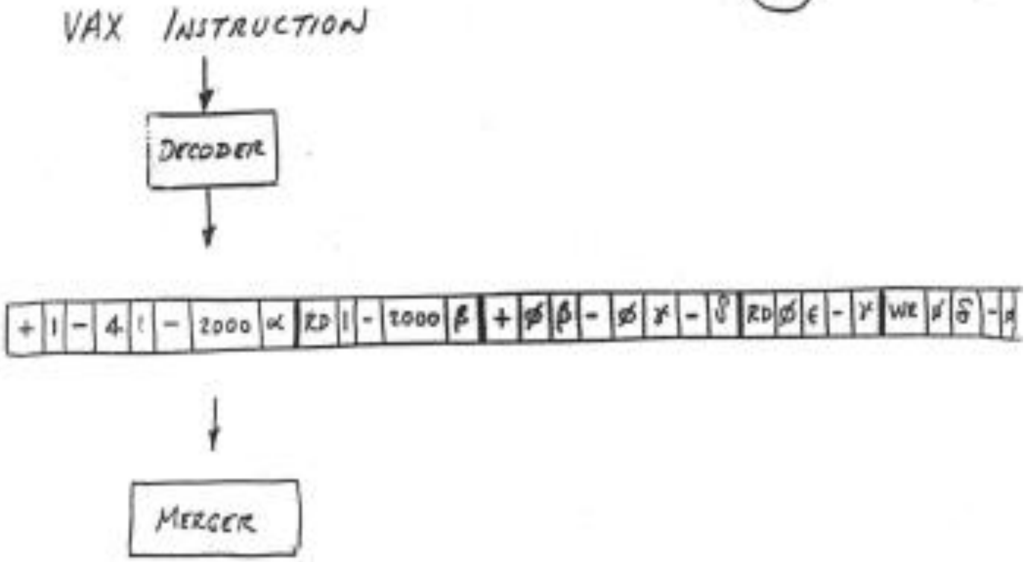
- ***Processing micro-ops! (Restricted Data Flow)***
- ***Incorporated the following:***
  - ***Aggressive branch prediction***
  - ***Speculative execution***
  - ***Wide issue***
  - ***Out-of-order execution***
  - ***In-order retirement***
- ***First published in Micro-18 (1985)***
  - ***Patt, Hwu, Shebanow: Introduction to HPS***
  - ***Patt, Melvin, Hwu, Shebanow: Critical issues***

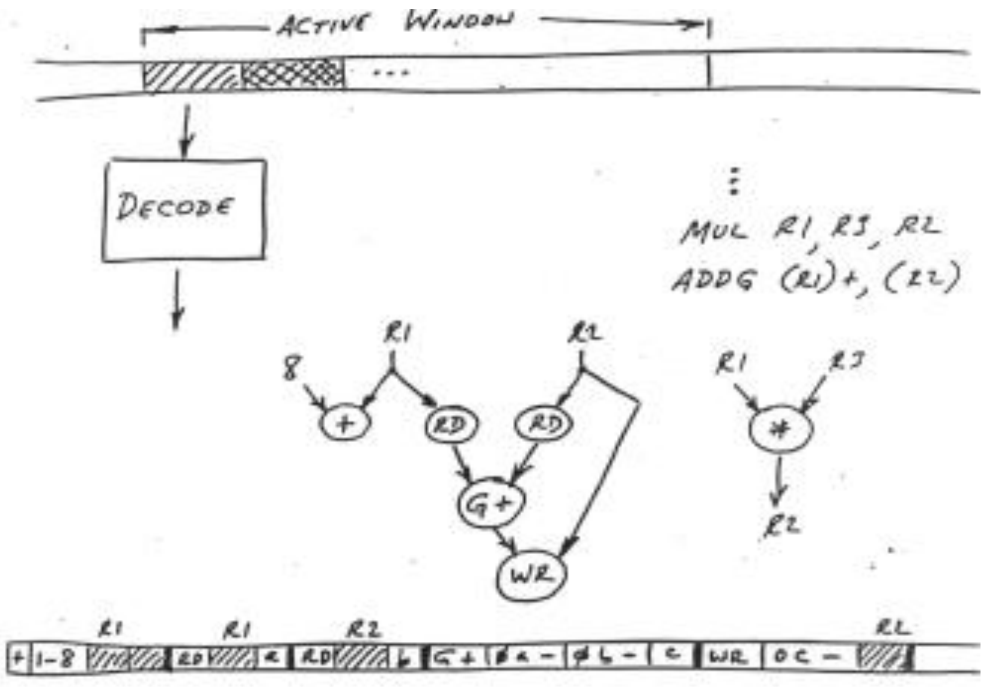
HPS
(RESTRICTED DATA FLOW)

FOR EXAMPLE, THE VAX INSTRUCTION:

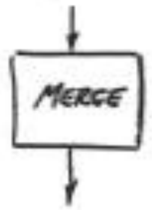
ADDL2 (R1)+, (R2)




VAX INSTRUCTION
DECODER
+ | 1 | - | 4 | 1 | - | 2000 | α | RD | 1 | - | 2000 | β | + | φ | β | - | φ | γ | - | δ | RD | φ | ε | - | γ | WR | φ | δ | - | φ
MERGER

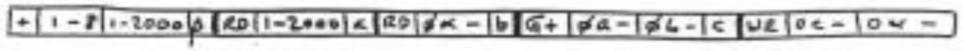


```
⋮
MUL R1, R3, R2
ADDG (R1)+, (R2)
```



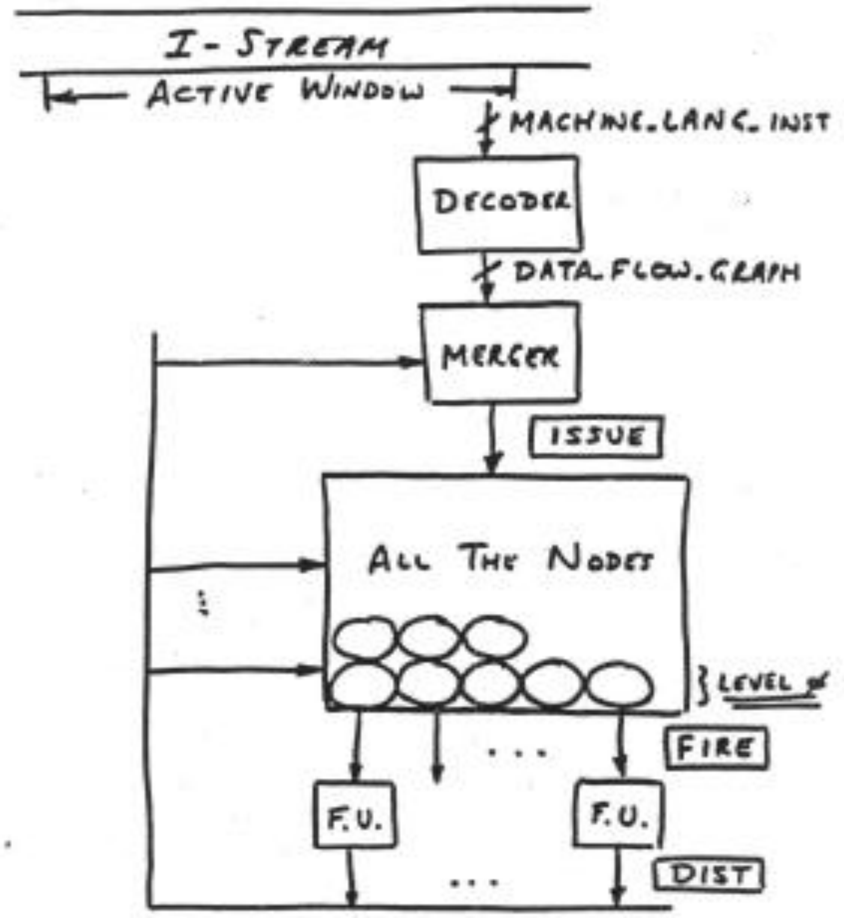
à la Tomasulo

R1	1	—	2000
R2	0	α	—
R3	1	—	100



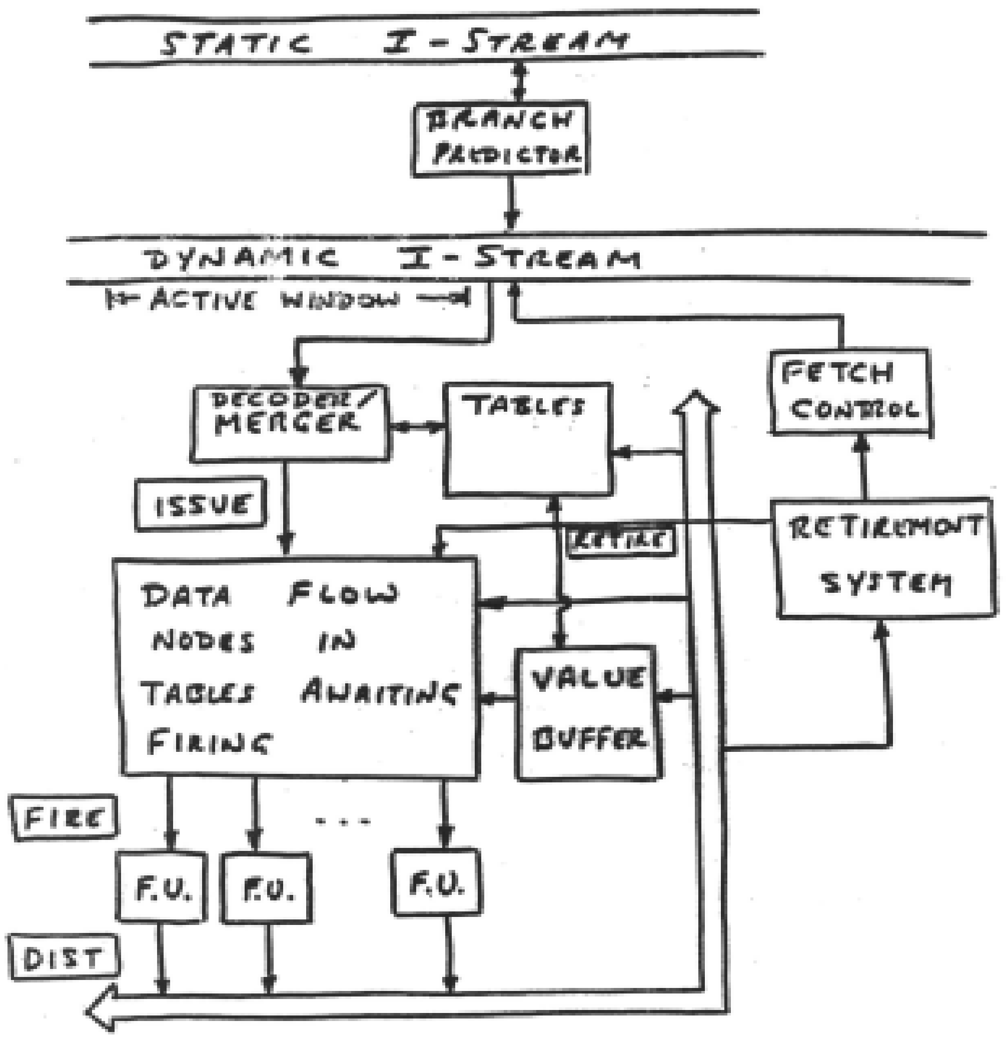
R1	φ	β	—
R2	φ	α	—
R3	1	—	100

HPS - What Is It?



* Restricted Data Flow

HPS

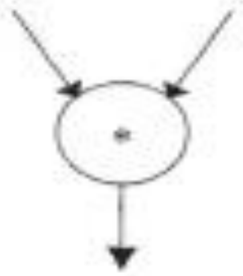

STATIC I-STREAM
BRANCH PREDICTOR
DYNAMIC I-STREAM
ACTIVE WINDOW
FETCH CONTROL
DECODER/MERGER
TABLES
ISSUE
RETIRE
RETIREMENT SYSTEM
DATA FLOW NODES IN TABLES AWAITING FIRING
VALUE BUFFER
FIRE
F.U.
F.U.
...
F.U.
DIST

Data Flow

- ***Data Driven execution of inst-level Graphical code***
  - ***Nodes are operators***
  - ***Arcs specify a producer/consumer relationship***
- ***Only REAL (i.e., flow) dependencies constrain processing***
  - ***Flow dependencies do (read-after-write)***
  - ***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")

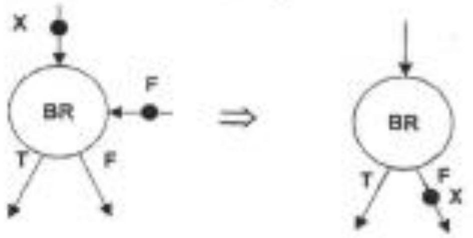
Fires When Ready

The Firing Rule:

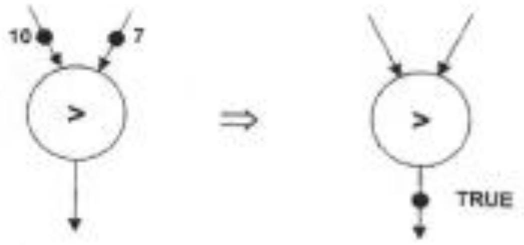
When all Inputs Have Tokens

(Note: Safe vs. Queues)

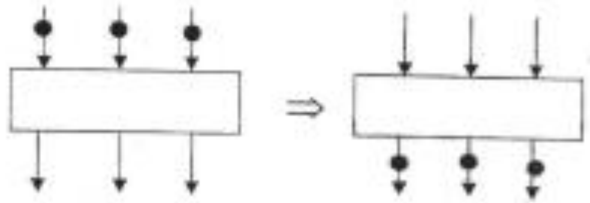
∗*Conditional*



∗*Relational*

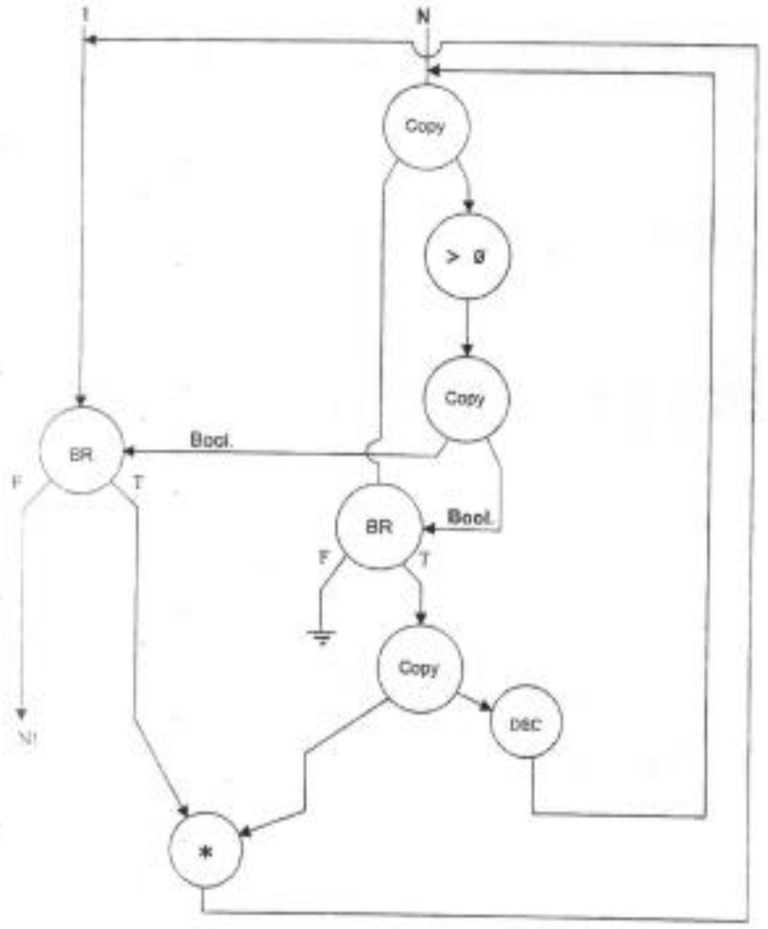


∗Barrier Synch



An Example Data Flow Program:

Factorial *(Done, Iteratively)*



Concerns with Full Data Flow

- ***Difficulty with taking an interrupt or exception***
  - ***Too much state information for a consistent state***
    - ***Some instructions have no data yet***
    - ***Some instructions have one operand and not the other***
  - ***Too much latency (time to create a consistent state***
- ***Difficulty in verifying the data flow engine***
  - ***The complexity of the engine***
  - ***Multiple instruction flows***
- ***Size of the representation of an algorithm***
- ***Handling recursion***
  - ***One operand is produced before the recursive call***
  - ***One operand is produced after the recursive call***
  - ***Another descriptor required to keep track of the iterations***

teşekkür ederim

Tesekkur ederim!