an URI / NEU collaboration

Levo ILP Machine

ILP Speed-Ups in the Tens !

student **David Morano**
advisors **Professor David Kaeli**
Professor Augustus Uht

NUCAR seminar 00/02/25

Outline

- motivation
- background
- key concepts and high-level machine organization
- major machine components
  - instruction window
  - architected registers
  - memory interface
- resource flow examples
- branch predication examples
- conclusions

Motivation

- **we want programs to run faster !**
- **we usually want single threaded programs to run faster !**
- **our approach is Instruction Level Parallelism (ILP)**
- **we want existing ISAs and compiled programs to benefit (micro-architectural solution)**
- **we need to employ some or most of the following design principles :**
  - **we need to have a wide instruction window to expose the ILP**
  - **we need a wide issue machine to increase IPC**
  - **we need to manage and exploit minimal control dependencies**
  - **we need to perform control and data speculation**
- **we end up with the Levo machine ! (one way to get high ILP)**
- **ILP in the tens !**

an URI / NEU collaboration

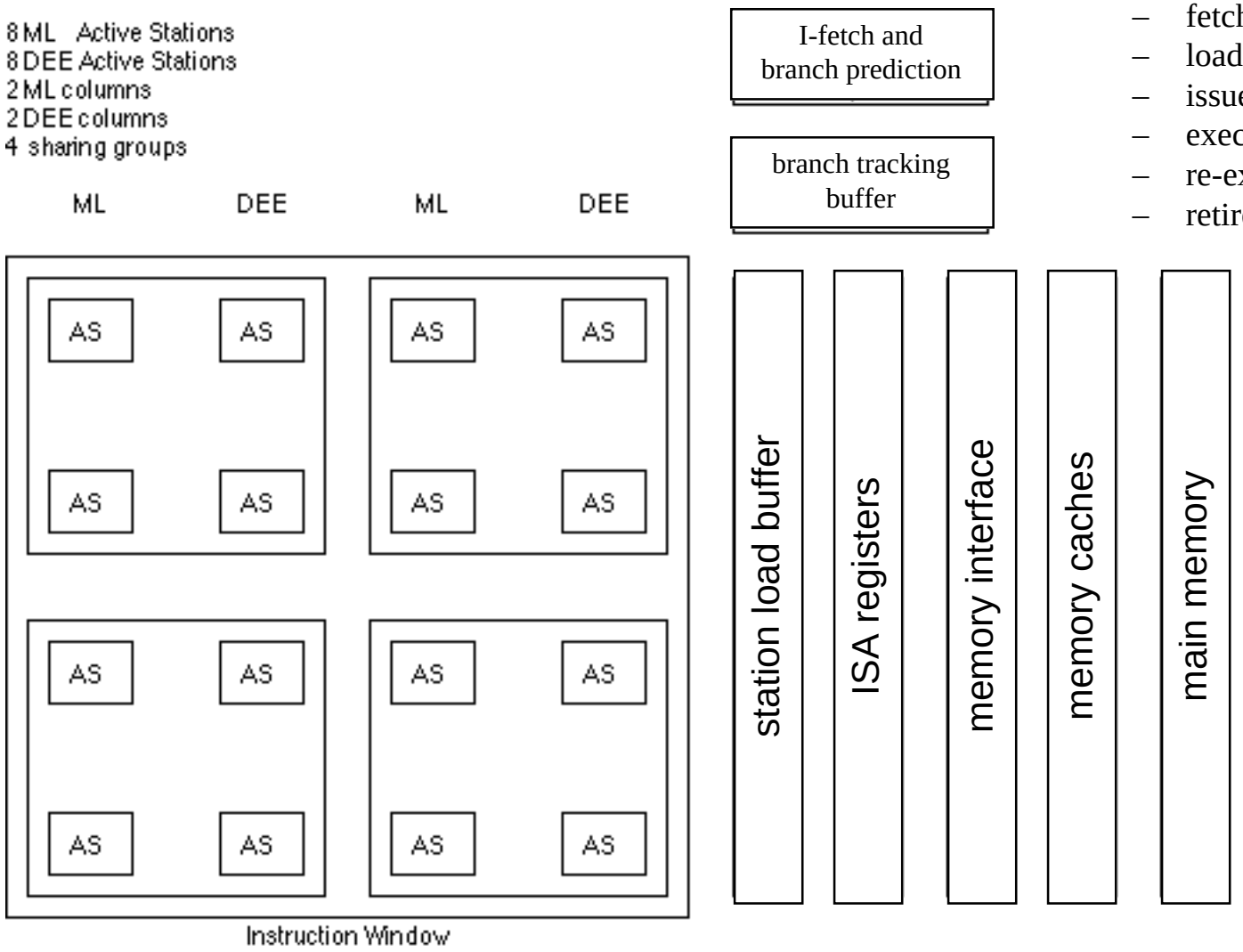
Background

- **1972 Garold S. Tjaden -** detection of concurrency and reduced control dependencies
- **1986 Uht -** extracting hardware concurrency
- **1991 Popescu, et al -** time tags for squashing on a branch misprediction
- **1991 Uht -** theory of minimal control dependencies
- **1995 Cleary, Pearson, Kinawi -** Warp Engine, used "time-stamps" for state roll-back of speculatively executed "events"
- **1995 Sohi, Breach, Vijaykumar -** Multiscalar Processor, concurrently executing pieces of a single path execution, uses compilation assistance communicated through the machine ISA
- **1995 Uht, Sindagi -** disjoint eager execution (DEE)
- **1997 Uht, Sindagi, Somanathan -** branch effect reduction techniques
- **1997 Uht -** verification of ILP speedups for DEE (TR)
- **1997 Uht -** high performance memory for ILP (TR)
- **1997 Uht -** Levo high-ILP computer (TR)

Key Concepts

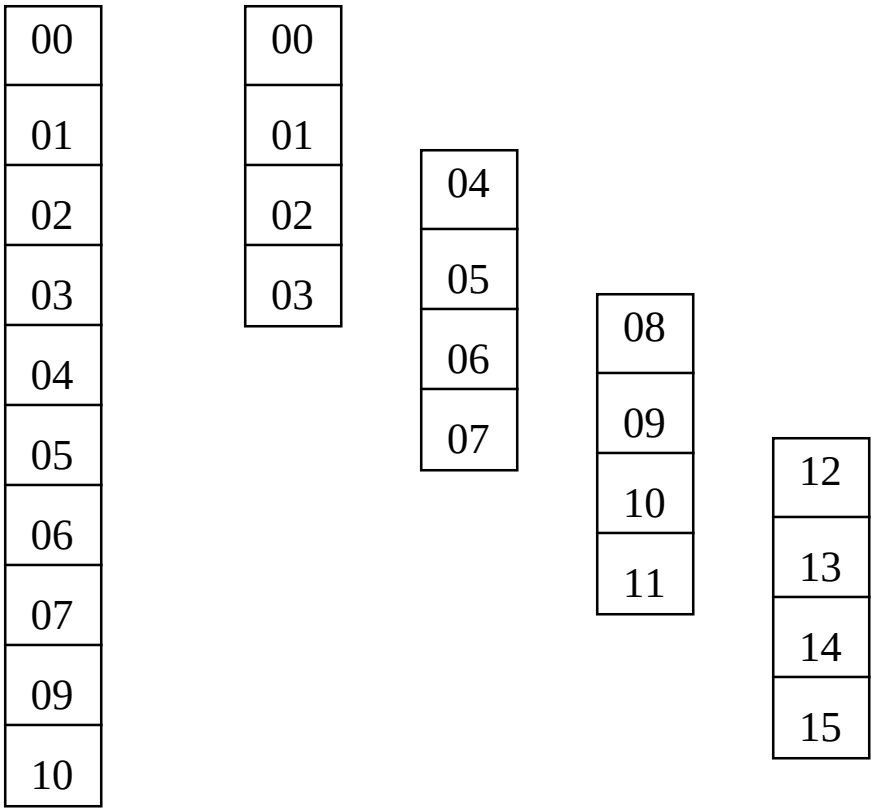
- **disjoint eager execution (old)**
- **resource flow computing (new)**
  - **execution proceeds primarily on resource availability rather than data or control dependencies !**
- **hardware branch predication (new)**
- **wide instruction window and issue (old)**
  - "A 21st century microprocessor may well [issue] up to dozens of instructions [per cycle, peak] ..." -- Dave Patterson, 1995
- **active station concept (new - extension of a reservation station)**
  - **instructions re-execute as necessary until retired**
- **execution sharing groups (new and old)**
- **result forwarding buses (extended CDB)**
- **high-bandwidth interleaved memory interface (old)**
  - **redundant load/store elimination**
- **scalability (new for this scale of an ILP machine)**

High-Level Block Diagram



- **execution flow**
  - fetch
  - load
  - issue
  - execute
  - re-execute !
  - retire

Folded Instruction Window



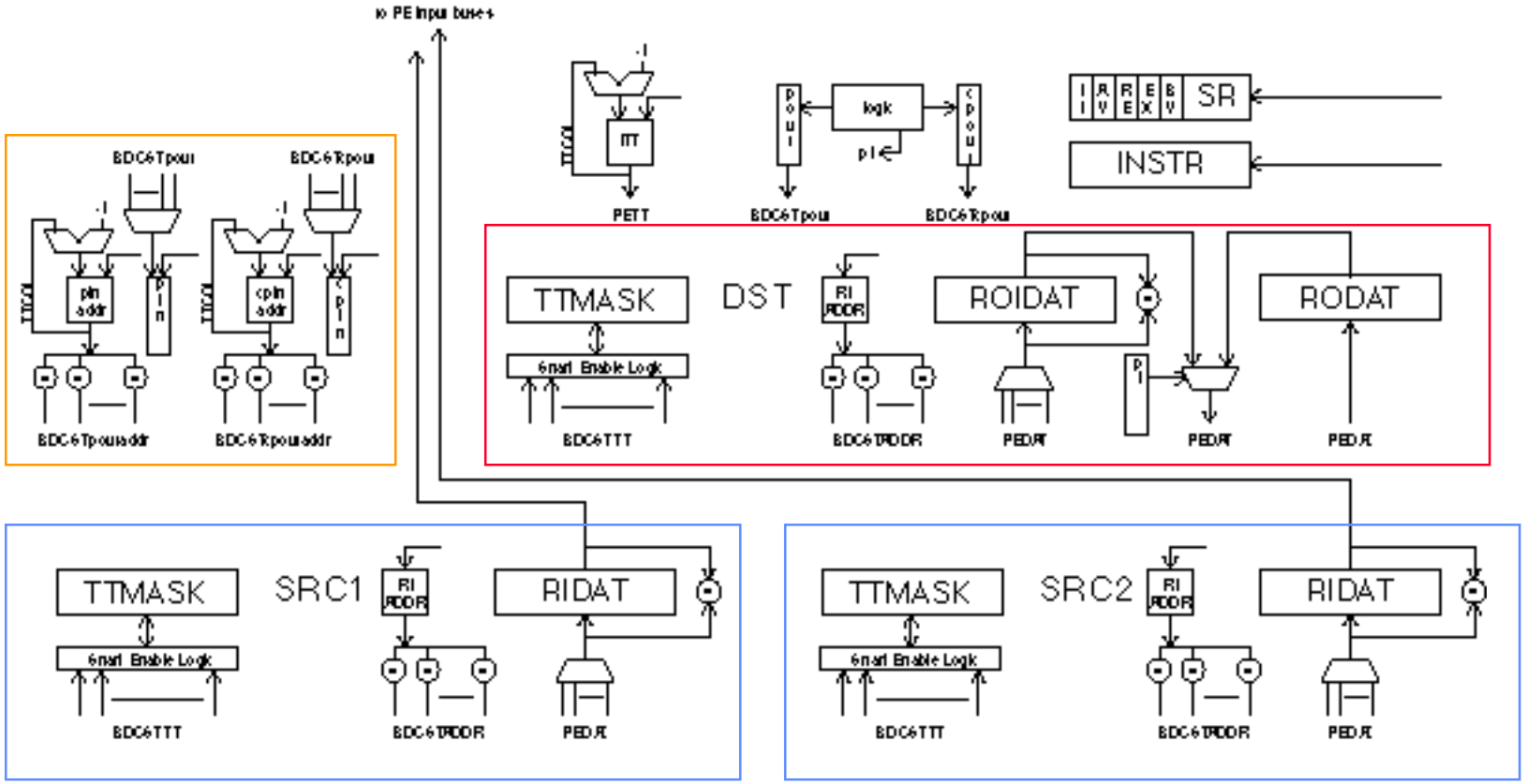
00	04	08	12
01	05	09	13
02	06	10	14
03	07	11	15

- **logical view of instructions**
- **columns form groups of instructions that are retired together**
- **peak simultaneous load is a whole column**
- **peak simultaneous execution is the number of sharing groups in the machine**

Active Station Functions

- sort of an extension to the reservation station
- serves as a place to wait for operands and control flow changes
- holds an instruction until it is ready to retire
- snoops all result forwarding buses coming to it for new operands
- snoops for control flow (prediction) changes by watching predicates representing the branch domains that the instruction is in
- re-executes !! an instruction when one or more of its operands change
- can squash its own execution with a "relay" operation (restores its output data as if it had not executed)
  - this operation notifies later instructions about the squash also supplying the restored output data value
- can rebroadcast its output on a switch from a DEE path to the ML path
- it slices, dices, and and can do windows !

Active Station Detail



Logical and Physical Groupings

logical organization

mainline (ML) path			
00	04	08	12
01	05	09	13
02	06	10	14
03	07	11	15

DEE path 1	
00	04
01	05
02	06
03	07

DEE path 2	
00	04
01	05
02	06
03	07

physical organization

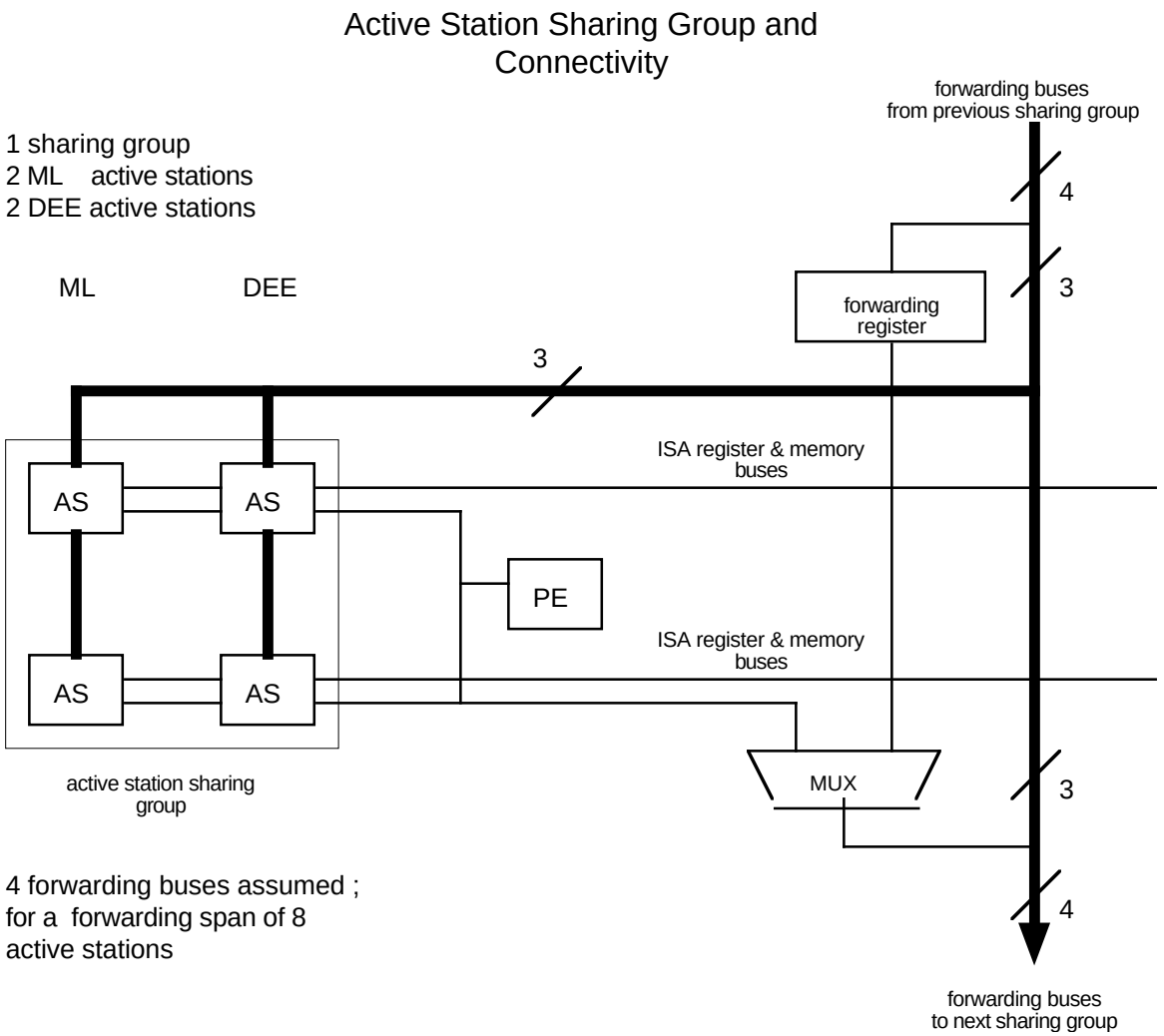
ML	D1
00	00
01	01
02	02
03	03

ML	D1
04	04
05	05
06	06
07	07

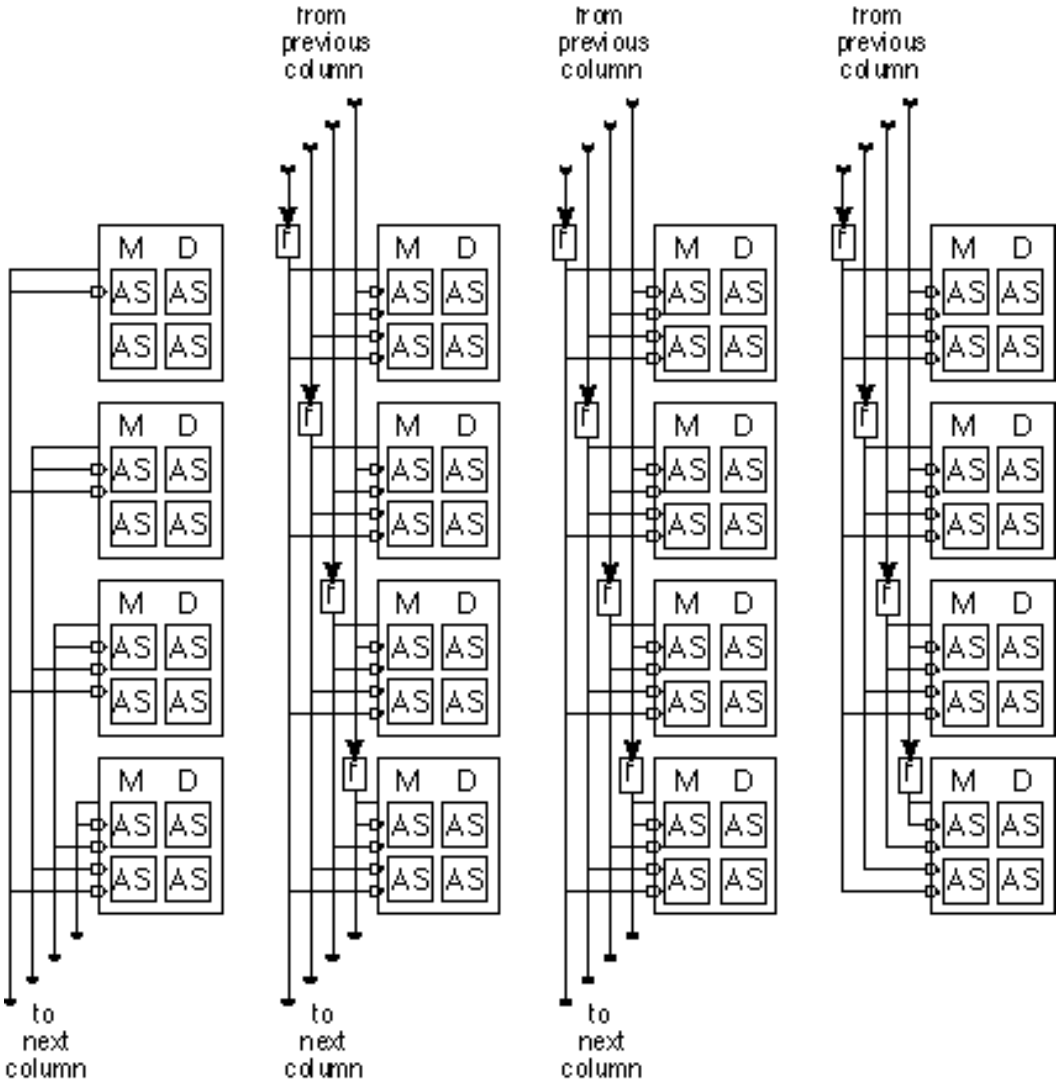
ML	D2
08	00
09	01
10	02
11	03

ML	D2
12	04
13	05
14	06
15	07

Active Station Sharing Group



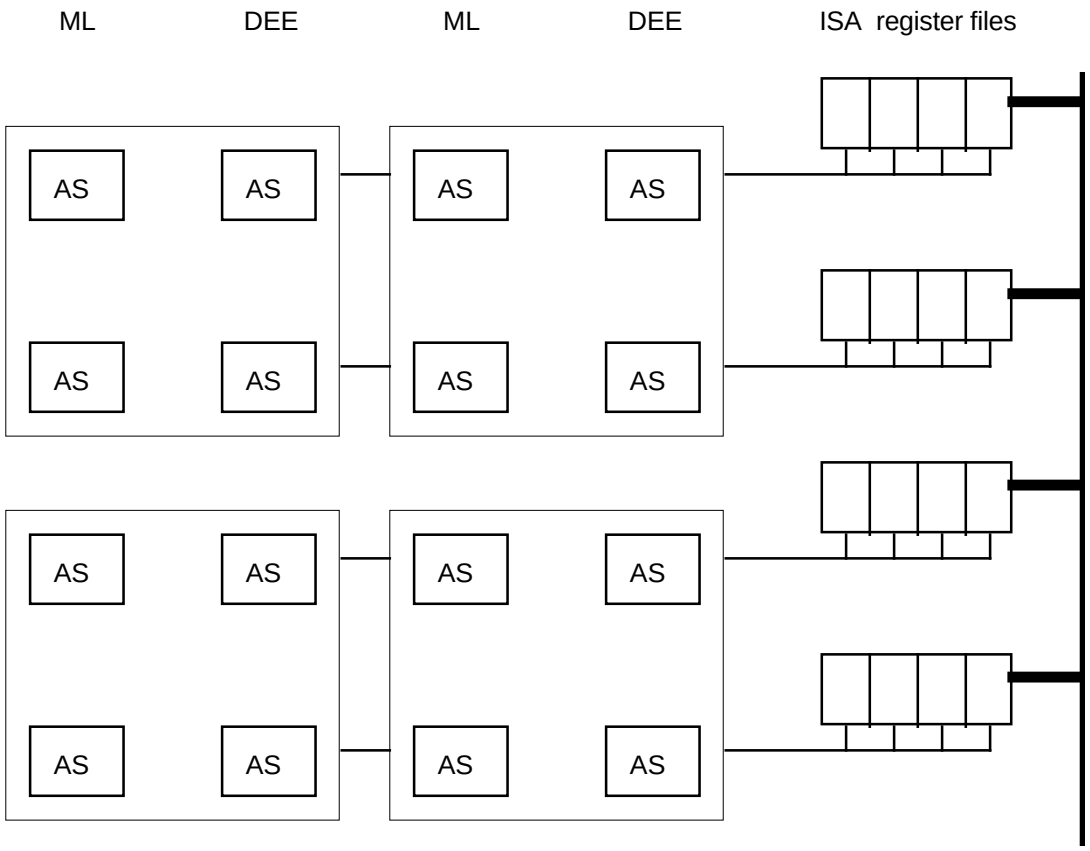
Sharing Group Forwarding Buses



Architected Register Files

8 ML Active Stations
8 DEE Active Stations
2 ML columns
2 DEE columns
4 sharing groups

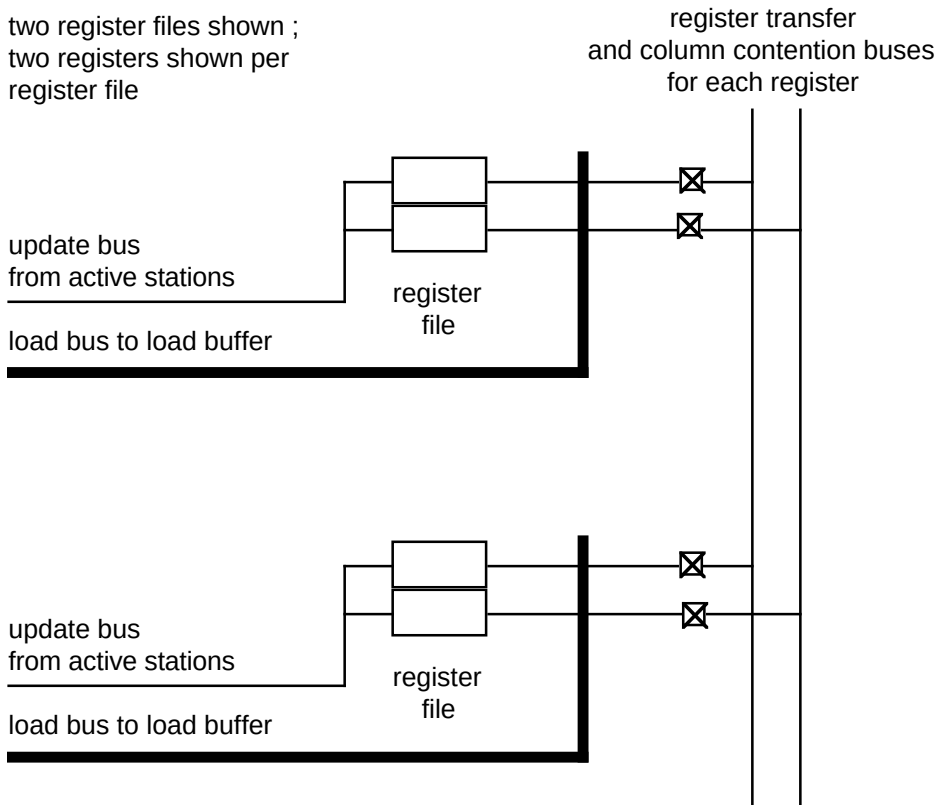
- **duplicated for bandwidth (reading or writing)**
- **reading and writing have different mappings**
- **an update coherency mechanism is used**



IT SCALES !

Register Files Detail

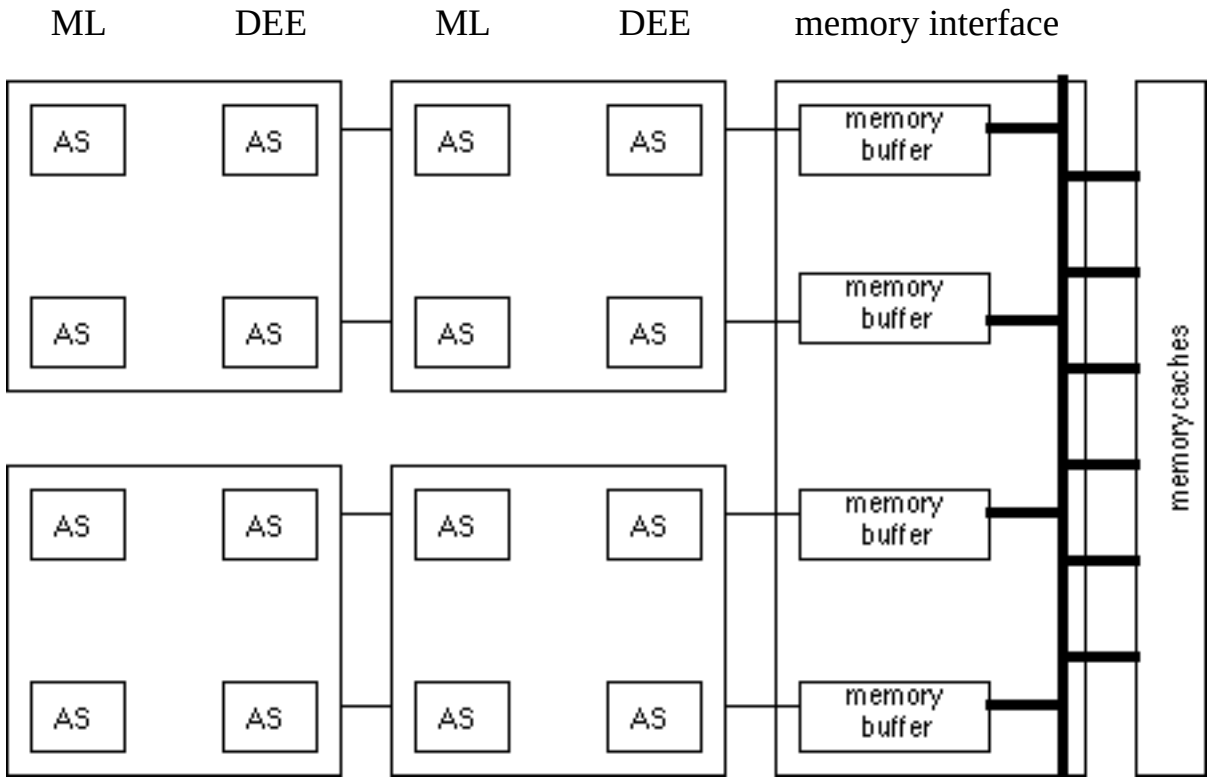
- reading and writing have different mappings
- coherency uses a wired "or" bus for each register



Memory Interface

- **memory and caches are interleaved**
- **redundant references are eliminated**

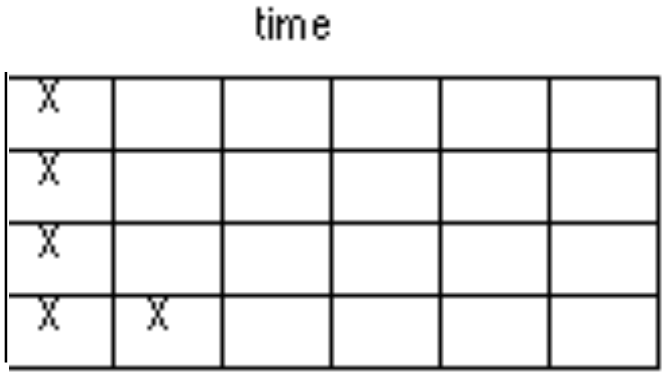
8 ML Active Stations
8 DEE Active Stations
2 ML columns
2 DEE columns
4 sharing groups



Resource Flow Example

- **X -- a ML execution**
- **R -- relay operation**
- **B -- broadcast only operation**
- **D -- disjoint path execution**

instruction		
	00	**r9 <- r0 op r1**
	10	**r3 <- r5 op r6**
	20	**r3 <- r7 op r8**
	30	**r2 <- r3 op r8**



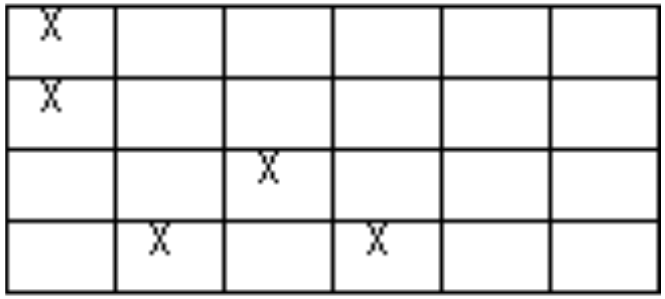
- all execute immediately
- instruction 30 re-executes when it gets an updated version of r3
- instruction 30 ignores the output from 10 instead preferring that from 20

Resource Flow Example

- **X -- a ML execution**
- **R -- relay operation**
- **B -- broadcast only operation**
- **D -- disjoint path execution**

instruction		
	00	**r9 <- r0 op r1**
	10	**r3 <- r5 op r6**
	20	**r3 <- r7 op r8**
	30	**r2 <- r3 op r8**

time



- 00 and 10 execute immediately, instructions 20 and 30 delayed for some reason
- instruction 30 executes when it gets r3 from 10
- instruction 20 eventually gets to execute
- instruction 30 re-executes with the output from 20

Resource Flow Example

- **X -- a ML execution**
- **R -- relay operation**
- **B -- broadcast only operation**
- **D -- disjoint path execution**

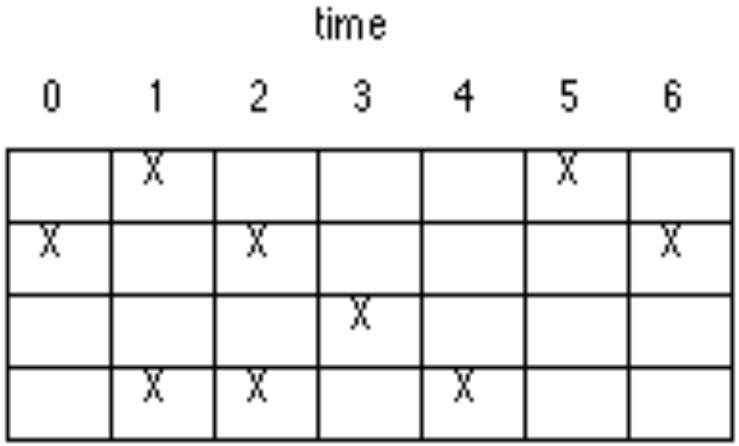
instruction		time					
00	**r3 <- r5 op r6**		X				
10	**r1 <- r3 op r1**	X		X			
20	**r3 <- r5 op r6**	X					

- 10 and 20 execute immediately, instruction 00 delayed
- instruction 10 does not re-execute due to its own broadcast of register r1 or because of the broadcast of r3 from instruction 20
- instruction 00 eventually executes
- instruction 10 re-executes because of forwarded value from 00

Resource Flow Example

- **X -- a ML execution**
- **R -- relay operation**
- **B -- broadcast only operation**
- **D -- disjoint path execution**

instruction		
	00	**r2 <- r0 op r1**
	10	**r3 <- r2 op r0**
	20	**r2 <- r0 op r4**
	30	**r5 <- r2 op r4**



- 10 gets to execute first for whatever reason
- 00 and 30 then get to execute
- instructions 10 and 30 re-execute because of forwarded value from 00
- 20 finally gets to execute
- 30 will re-execute because of the update from 20
- 00 re-executes for whatever reason causing 10 to re-execute but none later

Resource Flow Example

- **forwarding span delays**

- **X -- a ML execution**
- **R -- relay operation**
- **B -- broadcast only operation**
- **D -- disjoint path execution**

instruction		time 0	1	2	3	4	5	6
000	**r2 <- r1 op r0**	X						
040	**r3 <- r2 op r0**	X		X				
080	**r4 <- r2 op r0**	X			X			
120	**r5 <- r3 op r0**	X				X		

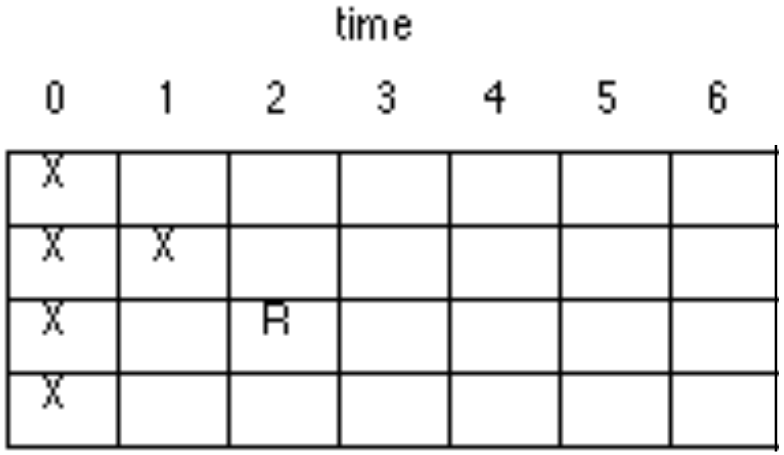
- assume a forwarding span of 32 instructions
- all instructions get to execute immediately
- r2 output from 000 is latched and only reaches 040 after a clock delay due to span distance
- the same r2 from 000 reaches instruction 080 after a forwarding span delay (1 clock) also
- 120 executes due to updated value (r3) from 040 also after a forwarding span delay

Branch Predication Example

- **minimal control dependency**
- **instruction relaying**

- **X -- a ML execution**
- **R -- relay operation**
- **B -- broadcast only operation**
- **D -- disjoint path execution**

instruction		
	00	**r2 <- r1 op r0**
	10	**b_op r2, 030**
	20	**r3 <- r1 op r0**
	30	**r4 <- r1 op r0**



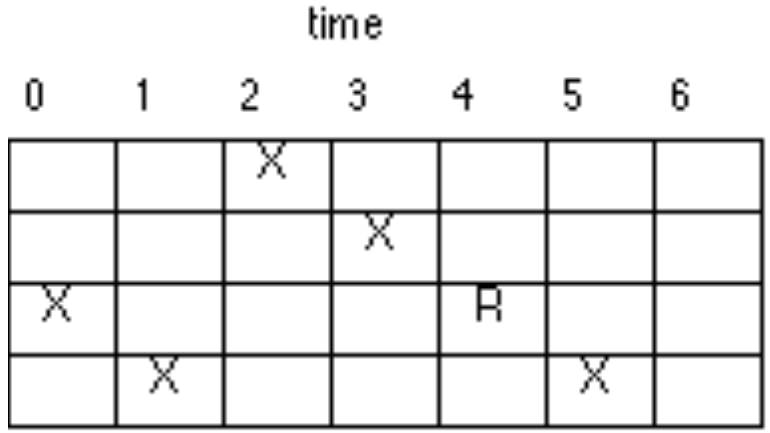
- the branch is initially predicted to be NOT taken !
- all instructions get to execute immediately
- instruction 10 re-executes due to new value from 00
- the branch changes its prediction due to being re-executed !
- instruction 20 (in the branch domain) forwards the original value for r3
- instruction 30 does not have to re-execute even though the branch prediction changed !

Branch Predication Example

- **minimal control dependency**
- **instruction relaying**
- **time-tag comparisons**

- **X -- a ML execution**
- **R -- relay operation**
- **B -- broadcast only operation**
- **D -- disjoint path execution**

instruction		
	00	**r2 <- r0 op r1**
	10	**b_op r2, 030**
	20	**r2 <- r3 op r0**
	30	**r4 <- r2 op r0**



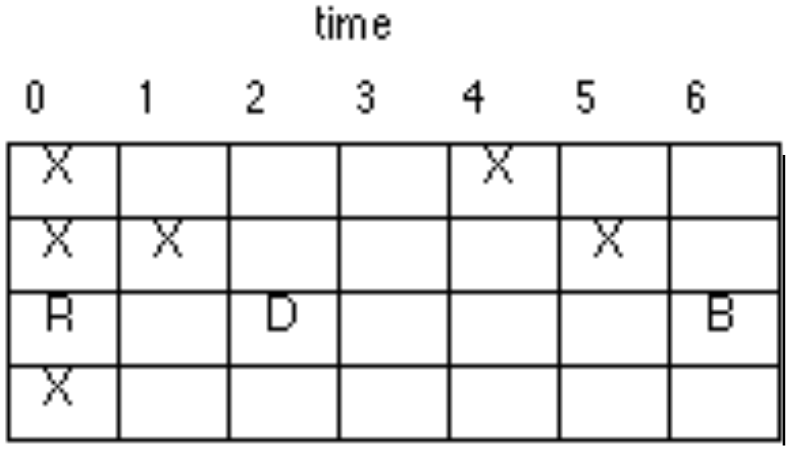
- the branch is initially predicted to be NOT taken !
- instruction 20 executes because it is in the predicted path
- instruction 30 executes due to new value from 20
- 00 finally executes causing the branch to re-execute bit not 30 ! (later in time)
- the branch is now predicted taken causing instruction 20 to relay
- the relayed value from 20 causing 30 to re-execute

Branch Predication Example

- **DEE execution example**
- **minimal control dependency**
- **instruction relaying**
- **time-tag comparisons**

- **X -- a ML execution**
- **R -- relay operation**
- **B -- broadcast only operation**
- **D -- disjoint path execution**

instruction		
	00	**r2 <- r0 op r1**
	10	**b_op r2, 030**
	20	**r3 <- r0 op r1**
	30	**r4 <- r0 op r1**



- the branch is predicted to be taken at the start of this example
- all instructions are executed immediately because they are in the main-line path, 20 is a relay
- 10 re-executes due to updated value from 00
- instruction 20 executes in a DEE path some time later
- 00 re-executes causing 10 to re-execute resolving the branch to not-taken and the DEE path to switch and become the new ML path, this causes instruction 20 to perform a broadcast

Conclusions

- **we realize disjoint eager execution**
- **we realize resource flow execution**
- **hardware branch predication**
- **wide instruction window and issue**
- **active stations are the key to instruction re-execution as necessary**
- **result forwarding buses (extended CDBs) provide for data updates and scalability**
- **the machine is linearly scalable with instruction window, issue, and execution resources !**

- **the future of ILP is**
- **the future of speculative execution is**
- **the future of computing is**

an URI / NEU collaboration

END