Explorations in Instruction Level Parallelism

student **David Morano**
advisors **Professor David Kaeli**

Northeastern University
Computer Architecture Research

NUCAR talk 2003-12-19

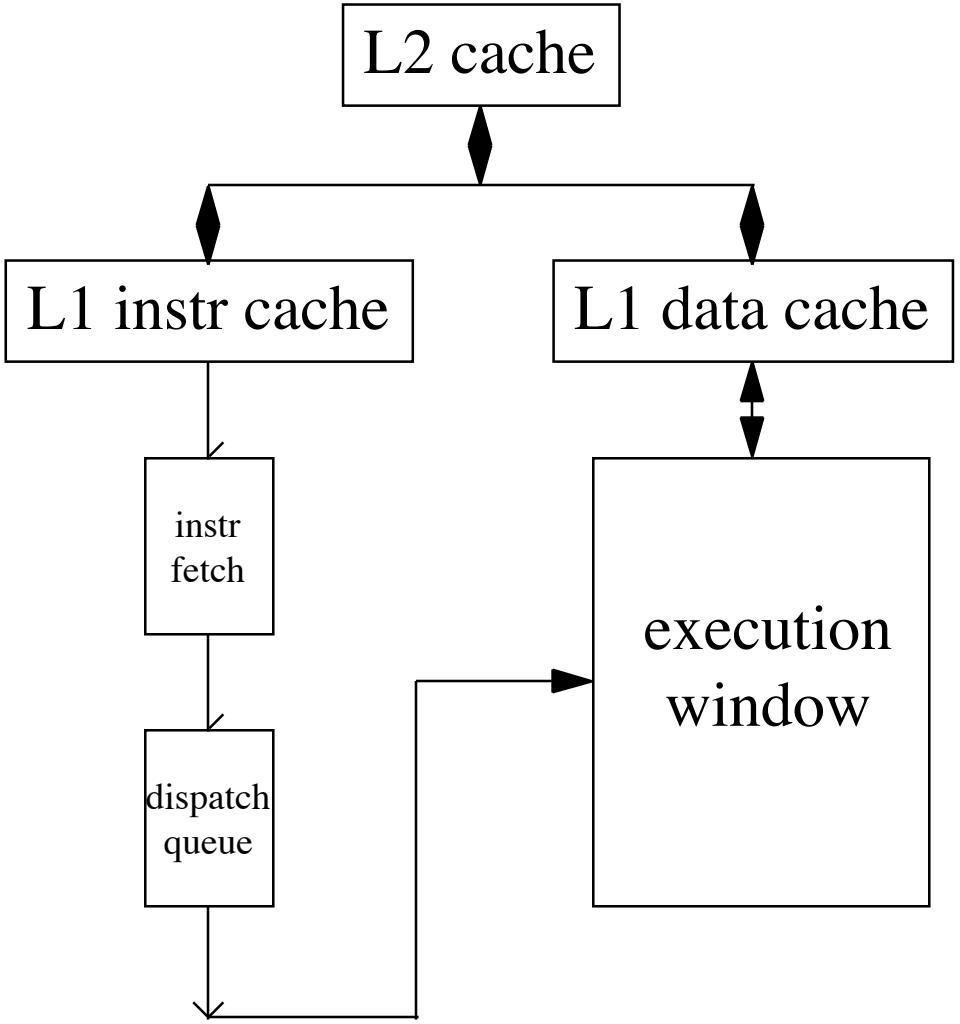
Outline

- **introduction**
- **microarchitecture overview**
- **execution window**
- **basic operation**
- **function units**
- **active station**
  - active station state
  - operands
  - operand snooping
  - some state
- **machine stages**
- **example execution**
  - register operands
  - memory operands

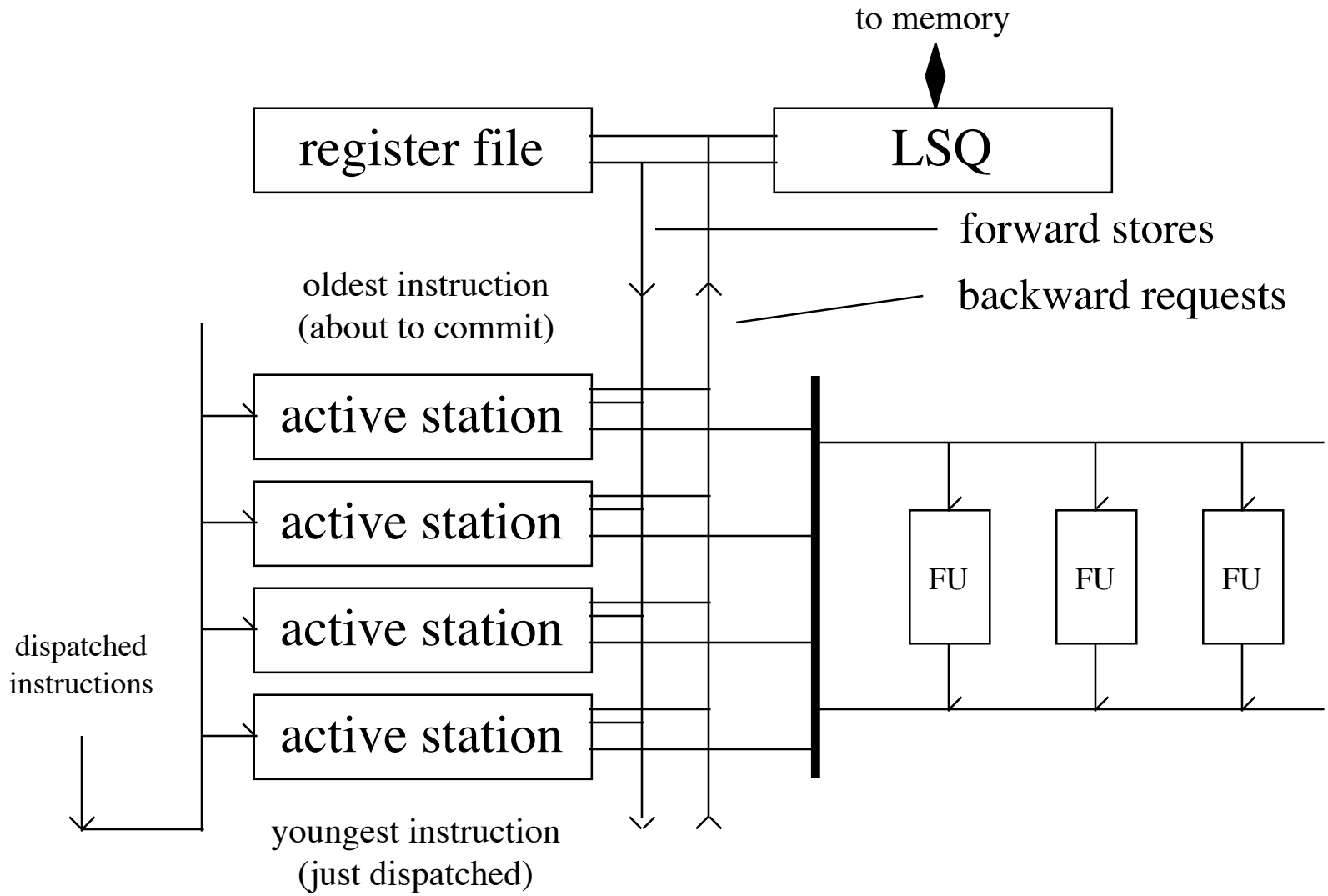
introduction

- **new parallel oriented microarchitecture**
- **constraints**
  - binary compatibility to existing ISAs (using Alpha)
- **we want to explore OoO execution with**
  - breaking control dependencies
  - breaking value depedencies (Levo last-value style)
  - using time-tags as the dependency enforcement mechanism
  - allowing for easy re-executions without pipeline flushes !!

microarchitecture overview



execution window



basic operation

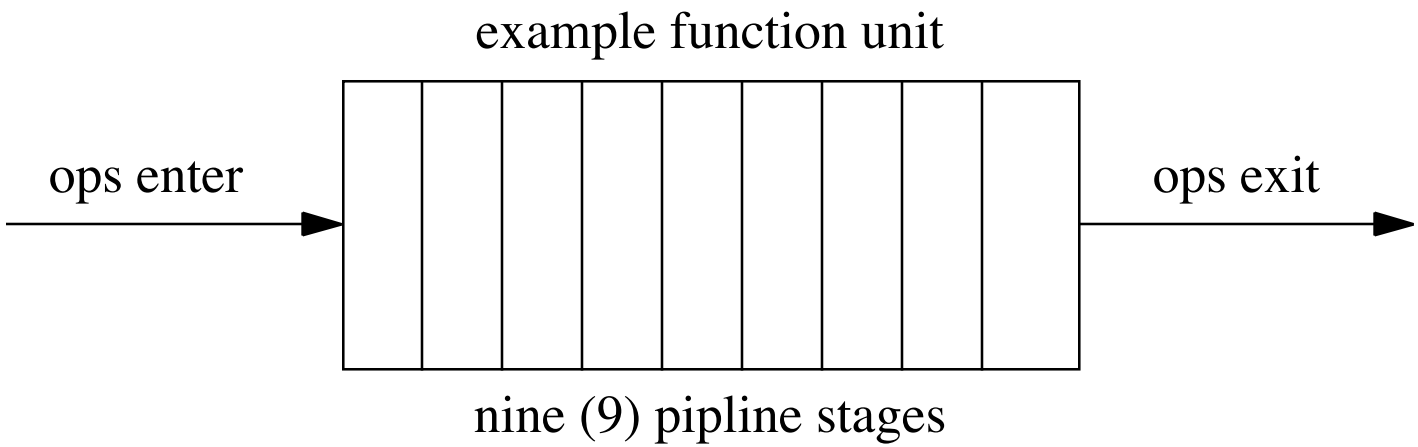
- **instructions are decoded at fetch time**
- **decoded instructions are stored in fetch buffers**
- **decoded instructions are dispatched to ASes**
- **ASes contend with each other for an issue (execution) slot**
- **ASes send an "operation" along with its input operands to a FU when an execution is needed**
- **the AS waits for the FU execution result**
  - should it send re-executions while waiting ?
  - should it attempt to cancel an execution that is not needed ? -- answer "no"
- **resulting operand returns to the originating AS**
- **AS forwards the result operand to other ASes in program-ordered future**
- **ASes who snarf new (different) input operands proceed to re-execute**

function units

- **nine different types -- def stages -- example numbers**
  - IALU 1 20
  - IMULT 7 4
  - IDIV 12 2
  - FADD 1 2
  - FCMP 1 2
  - FCVT 4 2
  - FMULT 4 2
  - FDIV 3 1
  - FSQRT 4 1
- **each has a different number of stages (configurable)**
- **there are different numbers of each (configurable)**
  - instructions that can't get an FU in a given cycle have to stall
- **they are pipelined and new operations can enter each cycle**

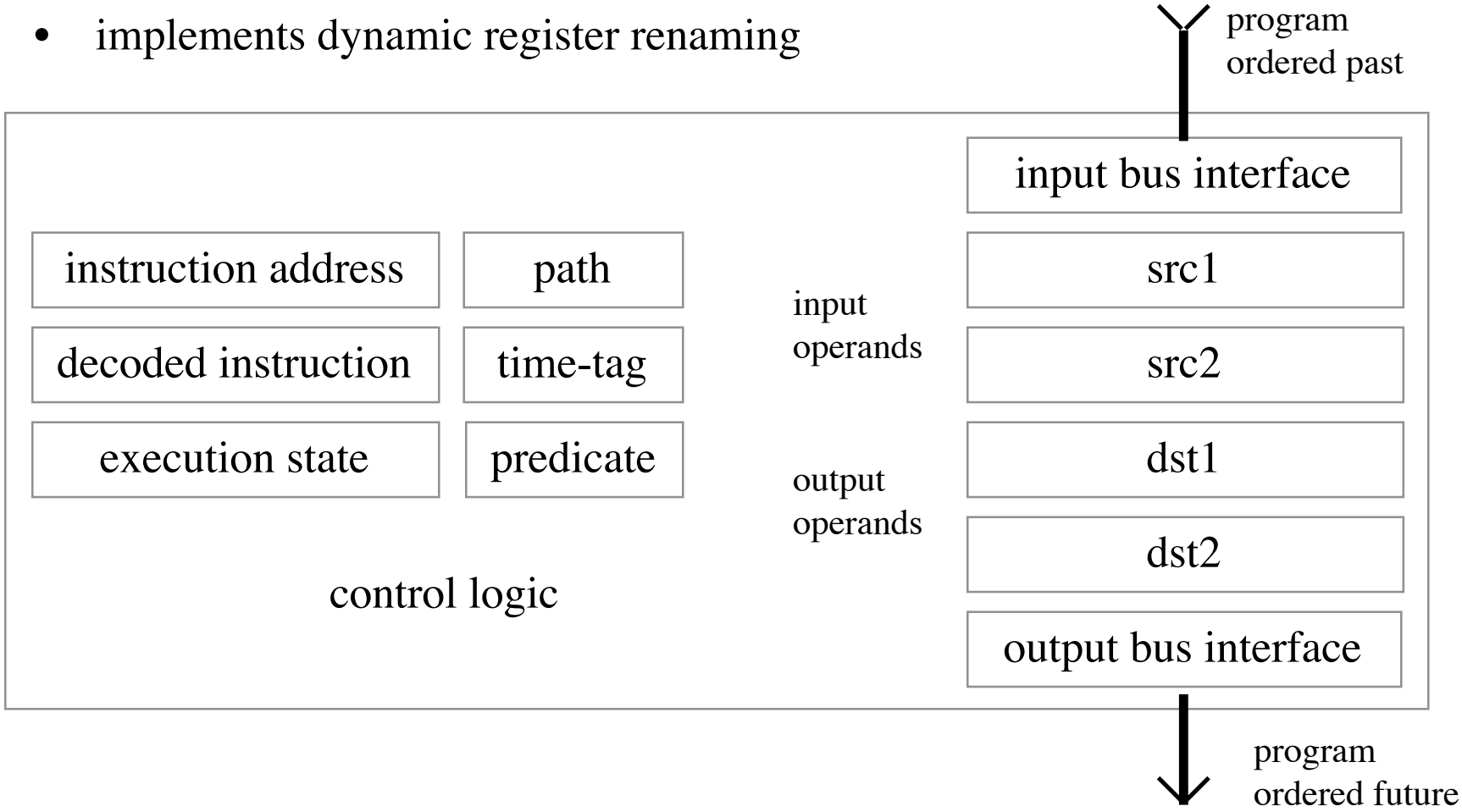
function units

- **operations enter from ASes**
- **exiting operations RETURN to their originating AS !**



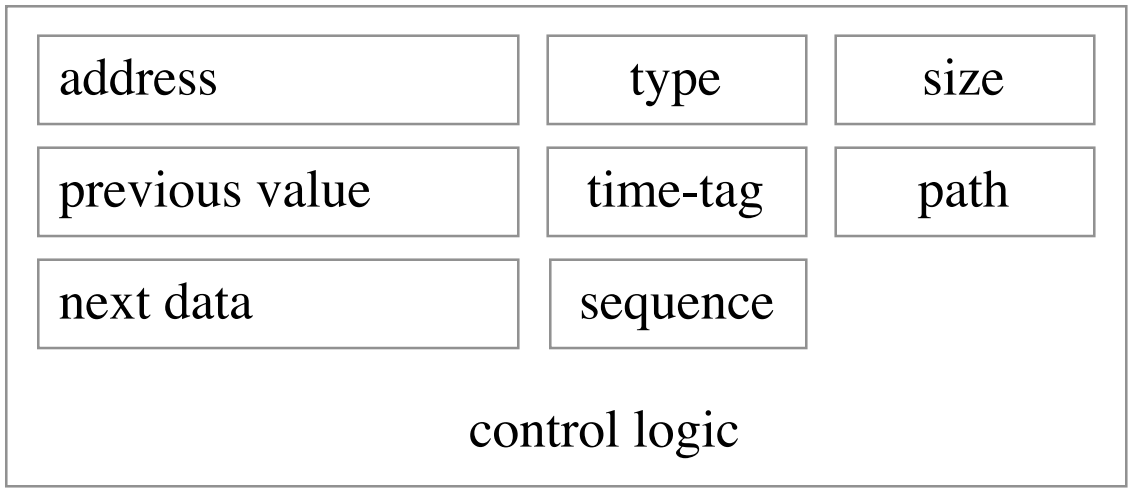
active station

- similar to Tomasulo's reservation station
- implements dynamic register renaming



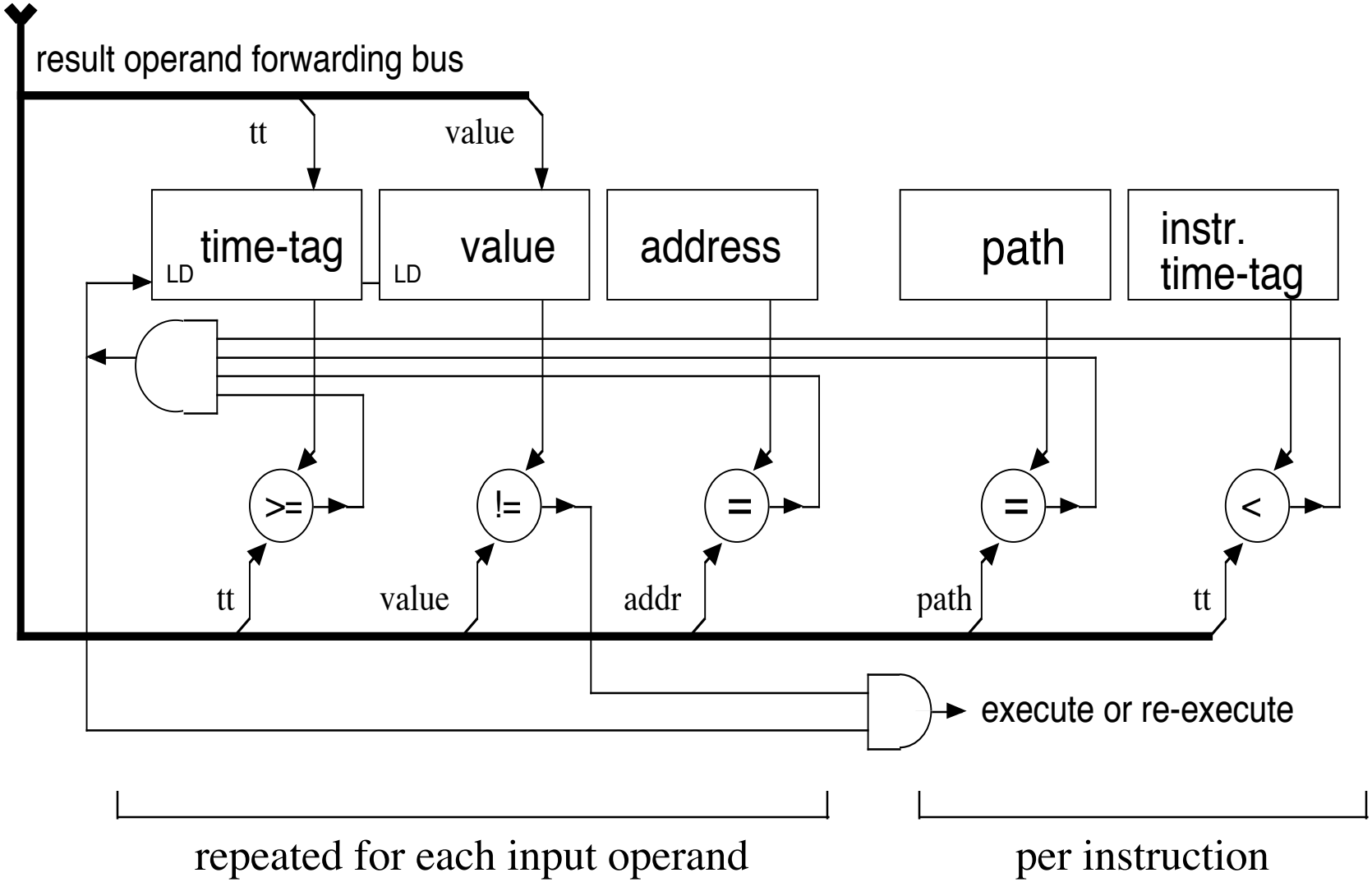
operand block

- holds all information about one operand
- includes necessary logic to snoop for updates



- operand names take the form -- type : path : time-tag : seq : addr
- example for a register -- "register : 1 : 27 : 3 : r6"
- predicates are operands also but have additional state

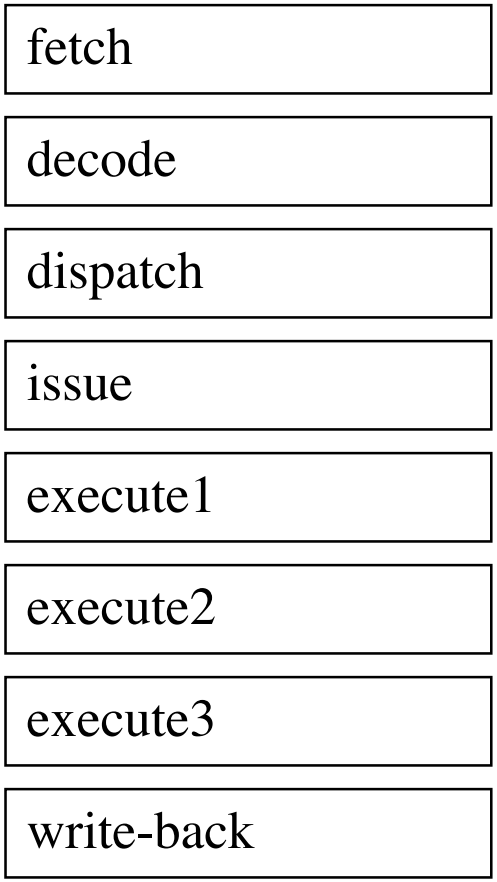
snoop/snarf operation



some simplified AS state

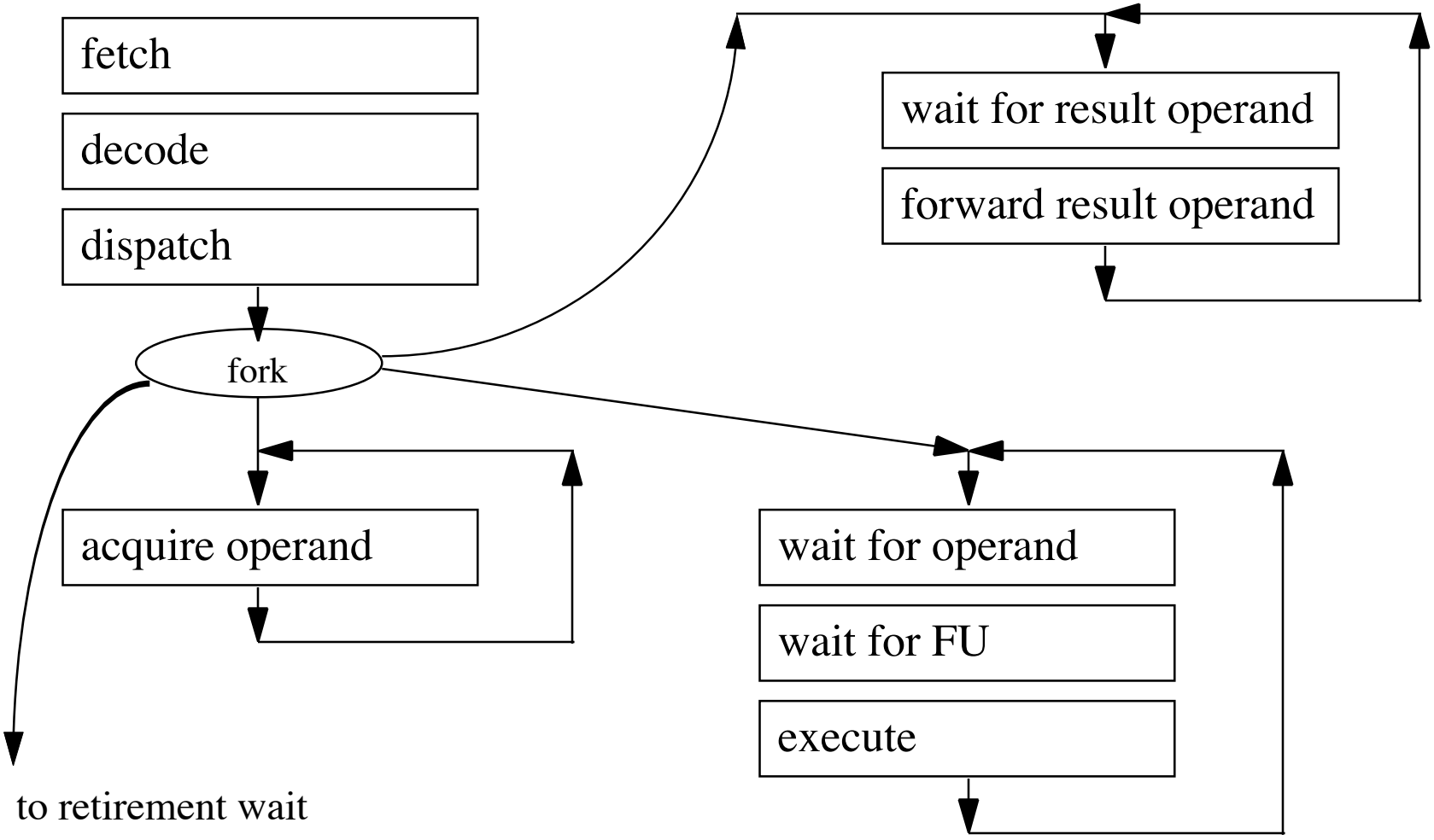
- acquiring input operands
- execution is needed (wait for FU availability)
- executing
- executed at least once
- result operand was requested by another AS
- result operand needs to be forwarded
- operand is being forwarded (used for commitment determination)

machine stages (0)

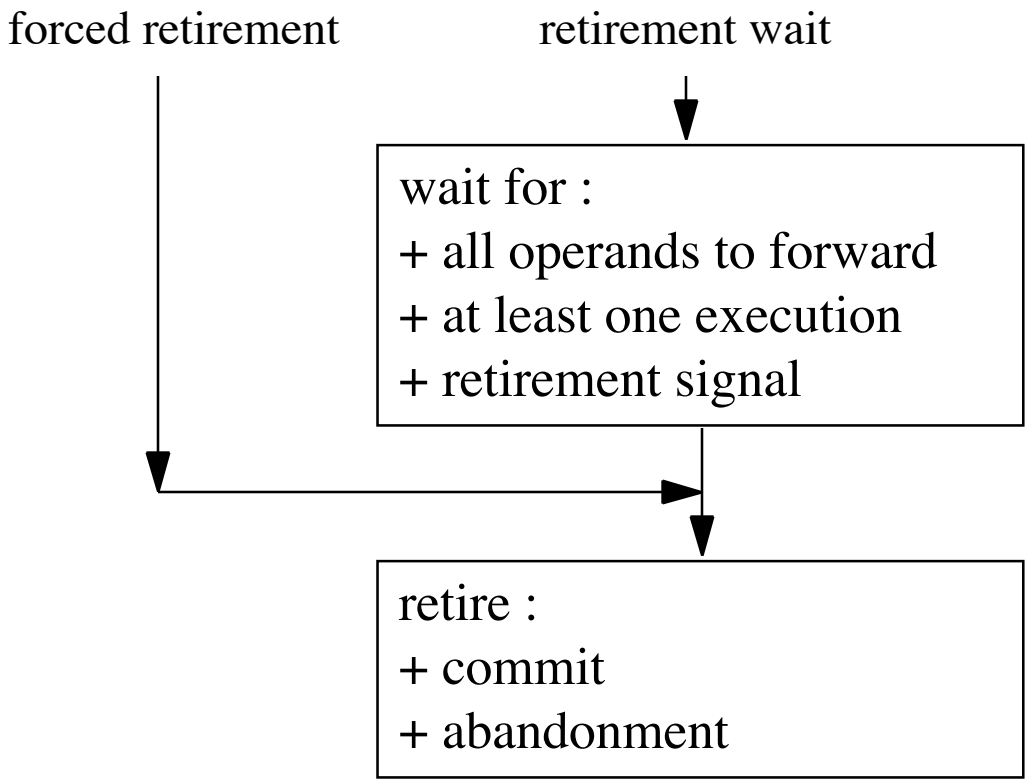


- **conventional pipeline**
- **various execution stages**

machine stages (1)



machine stages (2)



example execution (registers)

code fragment

label	*TT*	*instruction*
I1	0	r3 <= 1
I2	1	r4 <= r3 + 1
I3	2	r3 <= 2
I4	3	r5 <= r3 + 2

example execution schedule

cycle	*execute*	*forward*	*snarf*
-1		Ix{r3=?}, Iy{r4=?}	I2{r3=?}, I4{r3=?}
0	I1	Iz{r5=?}	
1		I1{r3=1}	I2{r3=1}, I4{r3=1}
2	I2, I4		
3	I3	I2{r4=2}, I4{r5=3}	
4		I3{r3=2}	I4{r3=2}
5	I4		
6		I4{r5=4}	

- we want *r3* to have value =2 after execution of I3
- we want *r5* to have value =4 after execution of I4

- I4 executes in clock 2 after snarfing *r3* from I1, resulting in *wrong* result
- I4 executes again after snarfing *r3* from I3, giving correct result

example execution (memory)

code fragment

label	*TT*	*instruction*
I1	0	(ma1) <= 1
I2	1	r4 <= (ma1)
I3	2	(ma1) <= 2
I4	3	r5 <= (ma1)

example execution schedule and transactions

cycle	*execute*	*forward*	*snarf*
-1		Ix{(ma1)=?}	I2{(ma1)=?}, I4{(ma1)=?}
0	I1		
1		I1{(ma1)=1}	I2{(ma1)=1}, I4{(ma1)=1}
2	I2, I4		
3	I3	I2{r4=1}, I4{r5=1}	
4		I3{(ma1)=2}	I4{(ma1)=2}
5	I4		
6		I4{r5=2}	

- **"ma1" -- some memory address, "(ma1)" -- some memory value**
- **we want *r4* to have committed value =1, from execution of I1**
- **we want *r5* to have committed value =2, from execution of I4**

- **I4 executes in clock 2 after snarfing *(ma1)* from I1, resulting in *wrong* result**
- **I4 executes again after snarfing *(ma1)* from I3, giving correct result**

status

- working on memory hierarchy to mimic Simple-Scalar
- need to finish coding LSQ
- code and verify proper memory operand handling (NULLIFYing)
- other