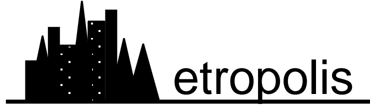
CPU Modeling and Use for Embedded Systems

Lecturer: Trevor Meyerowitz

EE249 Embedded Systems Design

Professor: **Alberto Sangiovanni-Vincentelli**

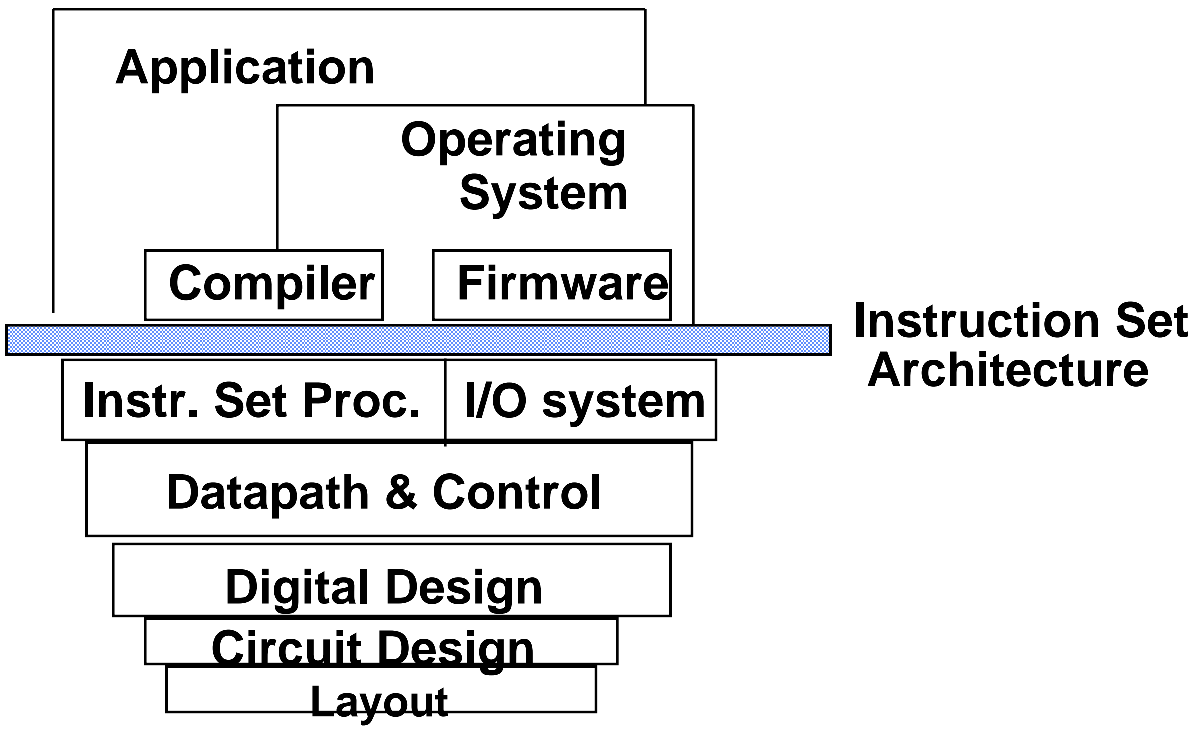
October 21st, 2004



Outline

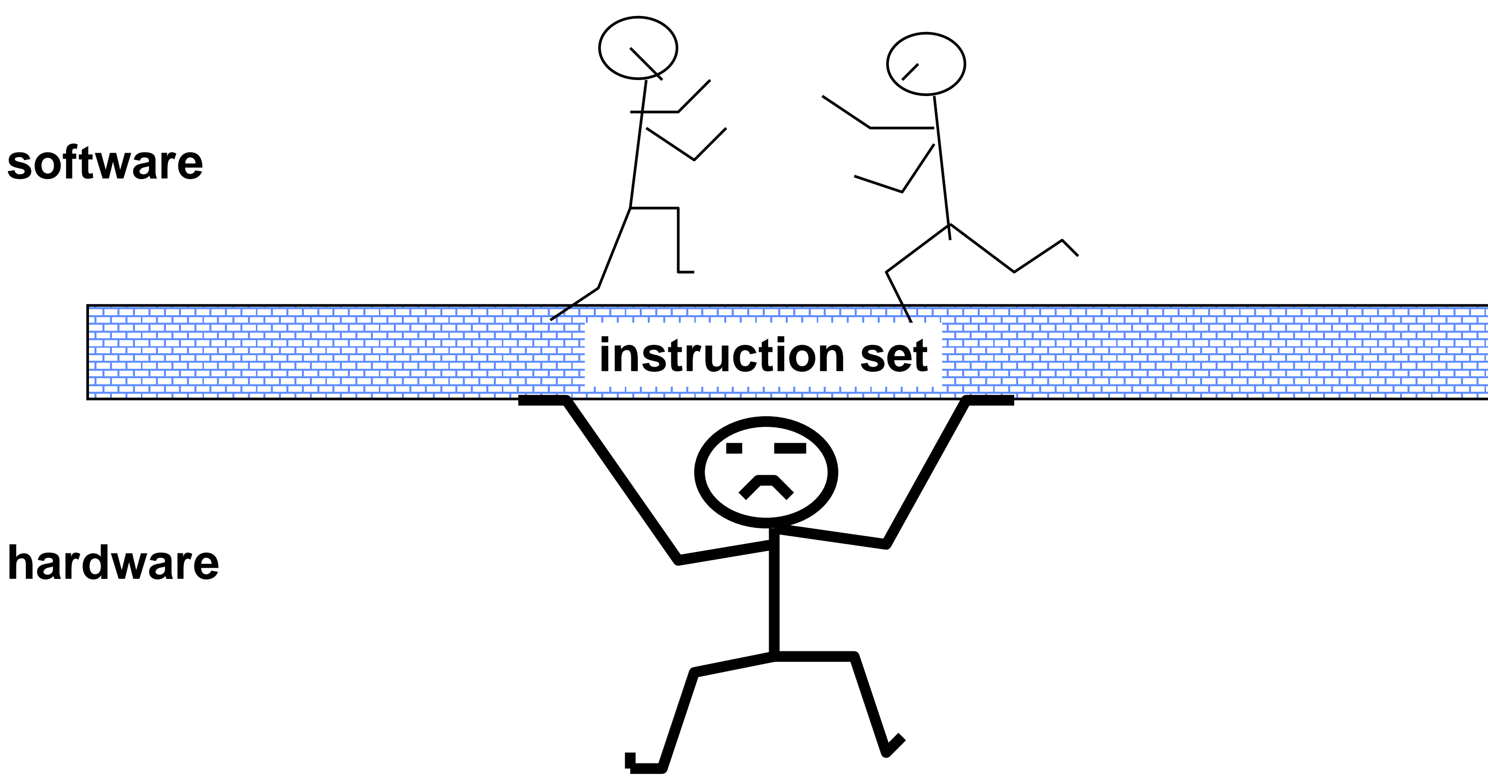
- Introduction
  - Motivation
  - Computer Architecture in 10 Minutes Flat
- Processor Modeling
- Use of Processor Modeling in Embedded Systems
- Conclusions

What is "Computer Architecture"?



- Coordination of many *levels of abstraction*
- Under a rapidly changing set of forces
- Design, Measurement, *and* Evaluation

The Instruction Set: a Critical Interface



Levels of Representation (61C Review)

High Level Language Program

Compiler

Assembly Language Program

Assembler

Machine Language Program

Machine Interpretation

Control Signal Specification

°

°

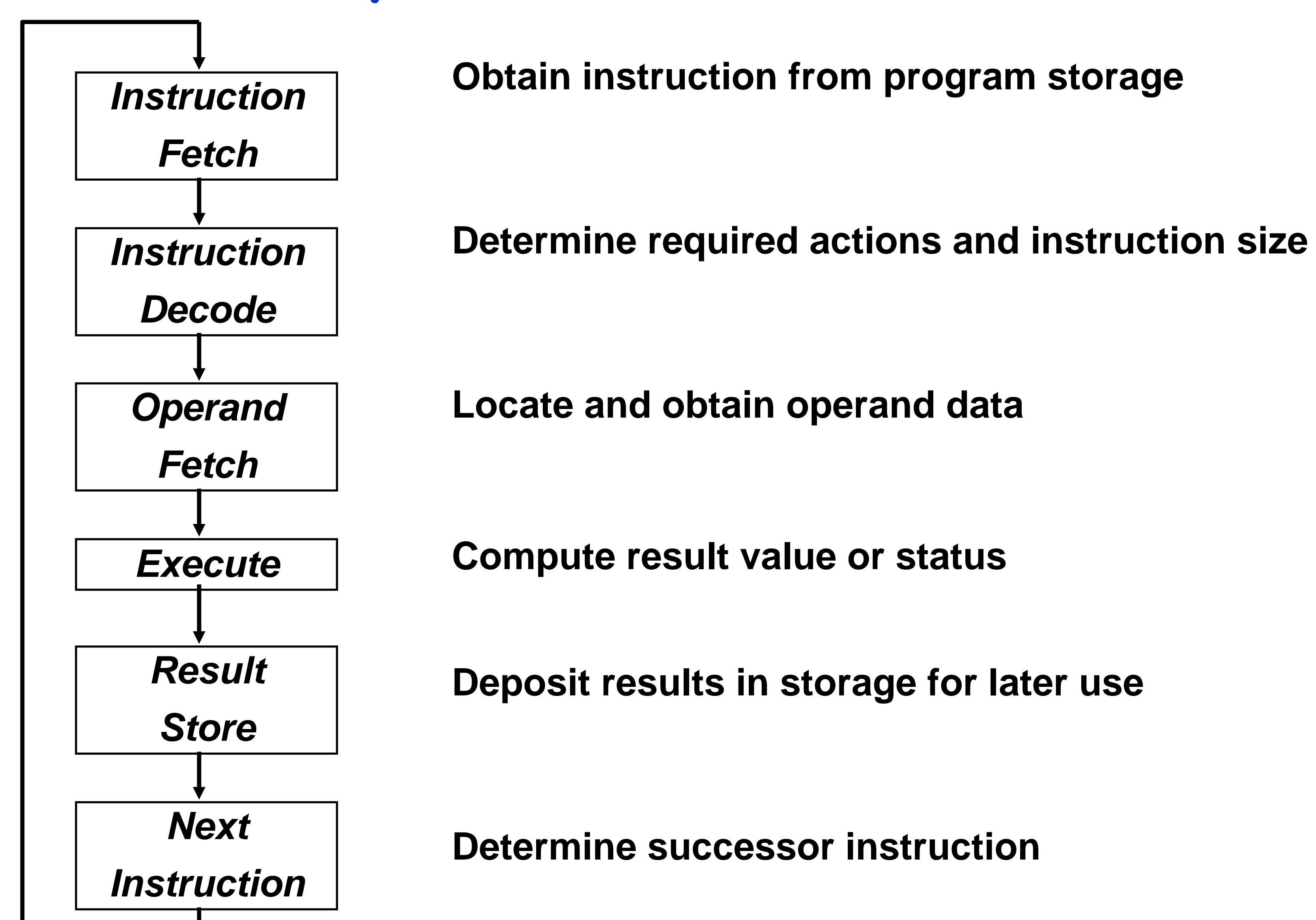
```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

```
lw  $15, 0($2)
lw  $16, 4($2)
sw  $16, 0($2)
sw  $15, 4($2)
```

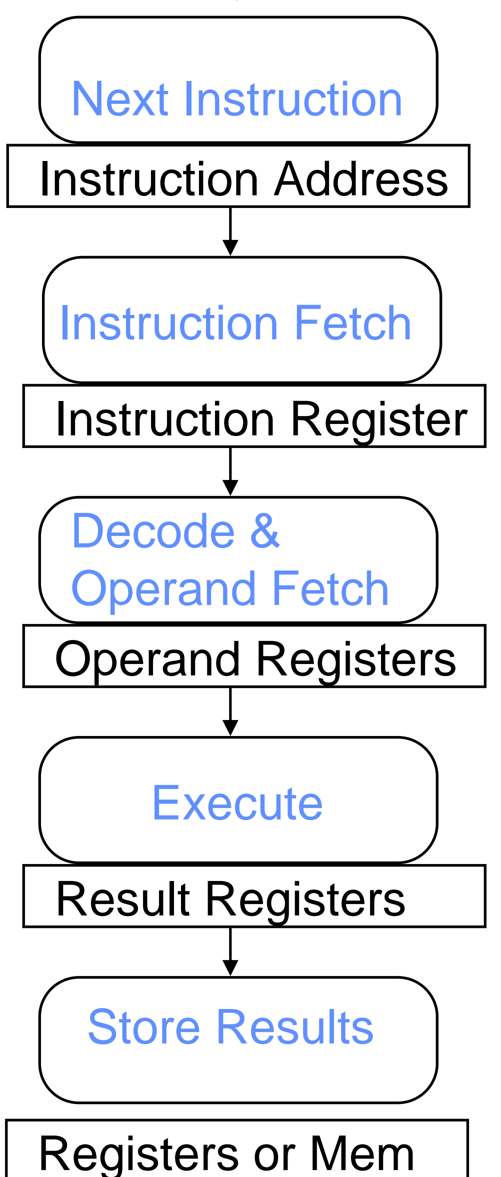
```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```

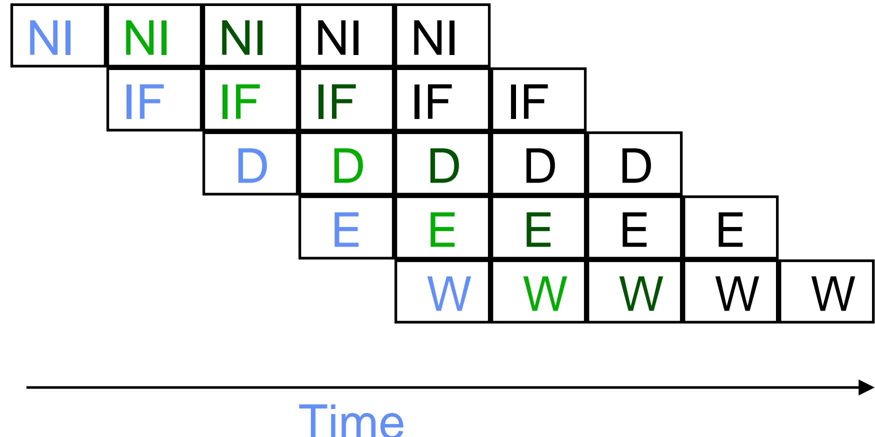
ALUOP[0:3] <= InstReg[9:11] & MASK

Execution Cycle



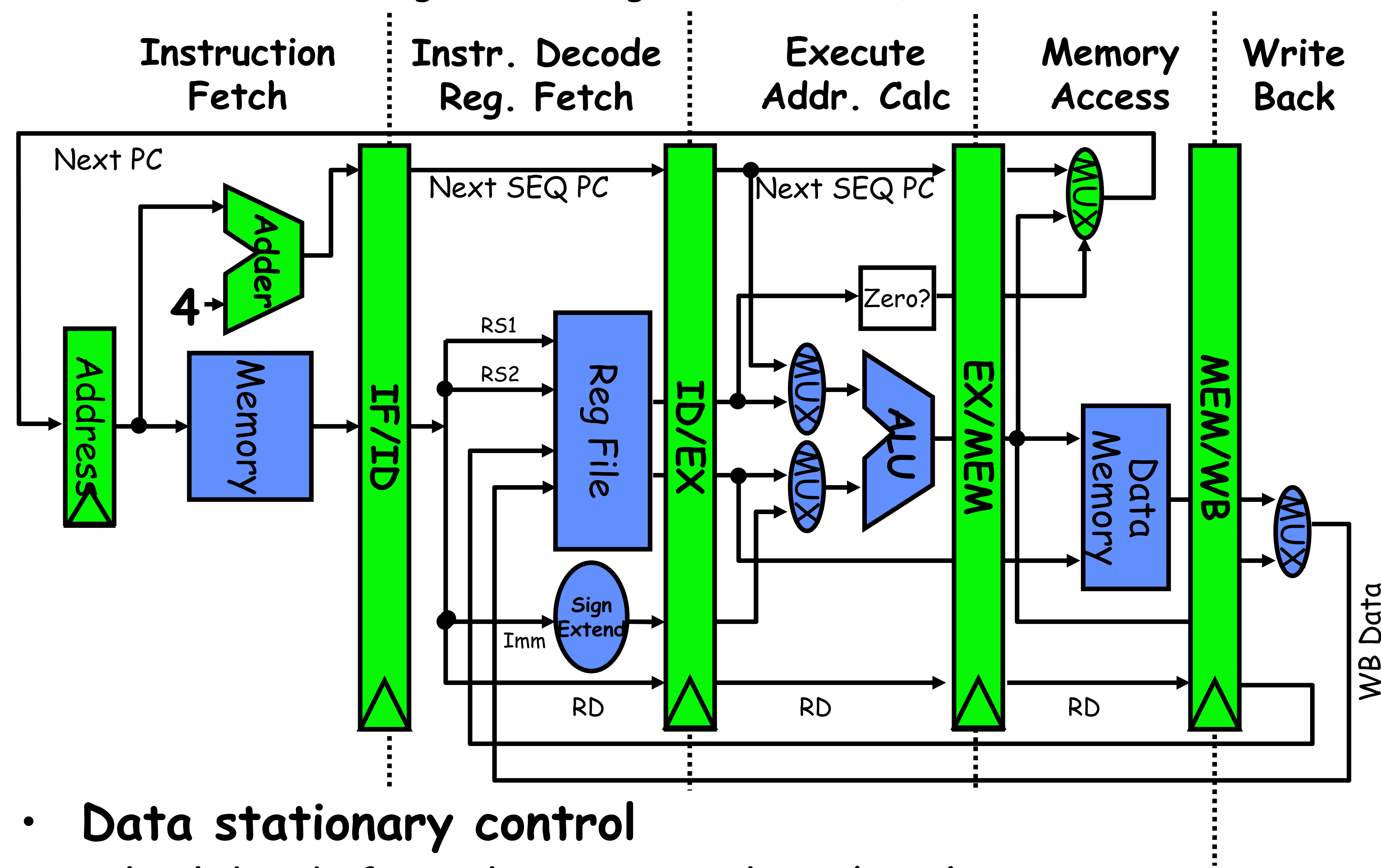
Fast, Pipelined Instruction Interpretation





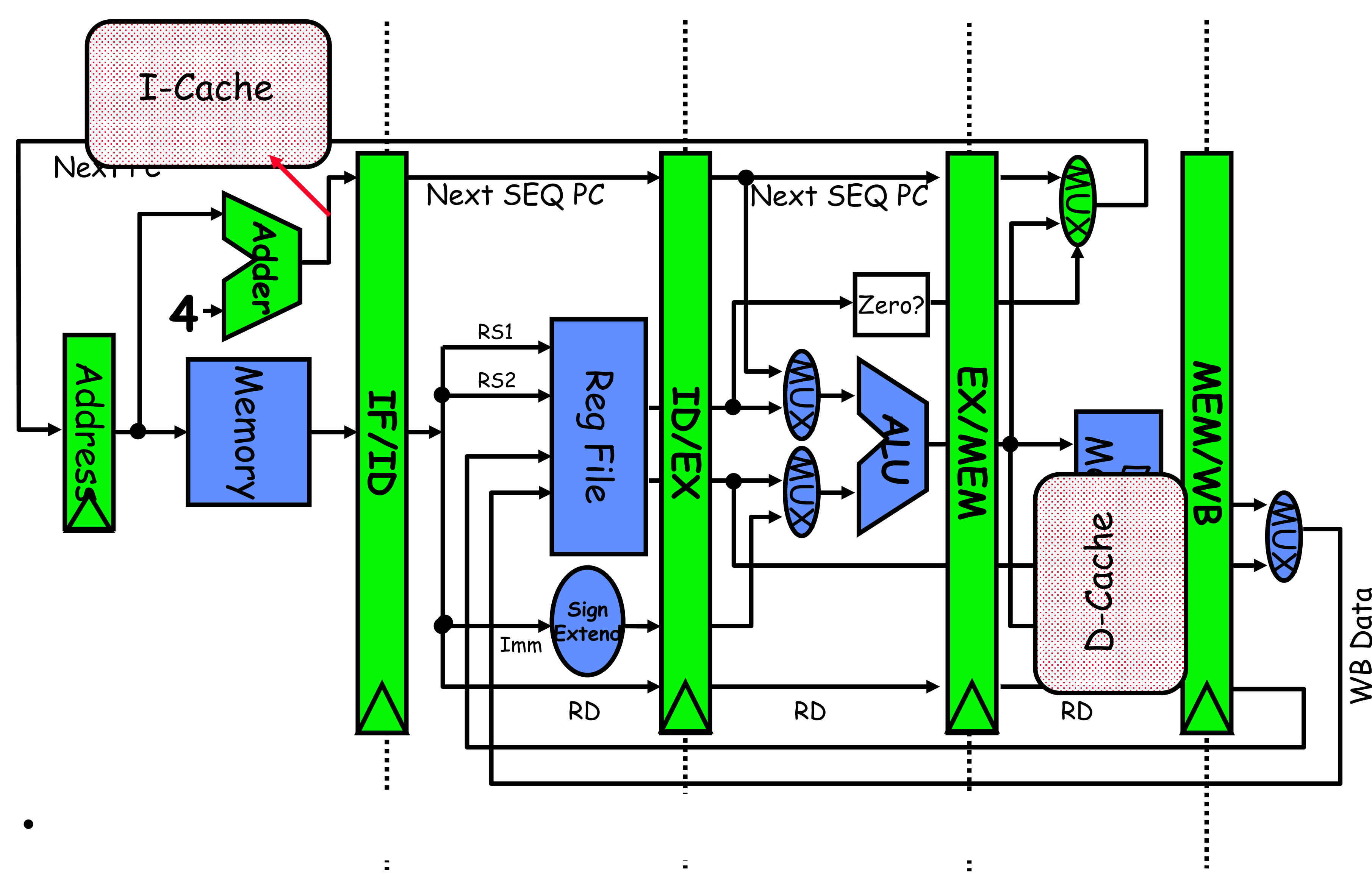
5 Steps of MIPS Datapath

Figure 3.4, Page 134 , CA:AQA 2e



- **Data stationary control**
  - local decode for each instruction phase / pipeline stage

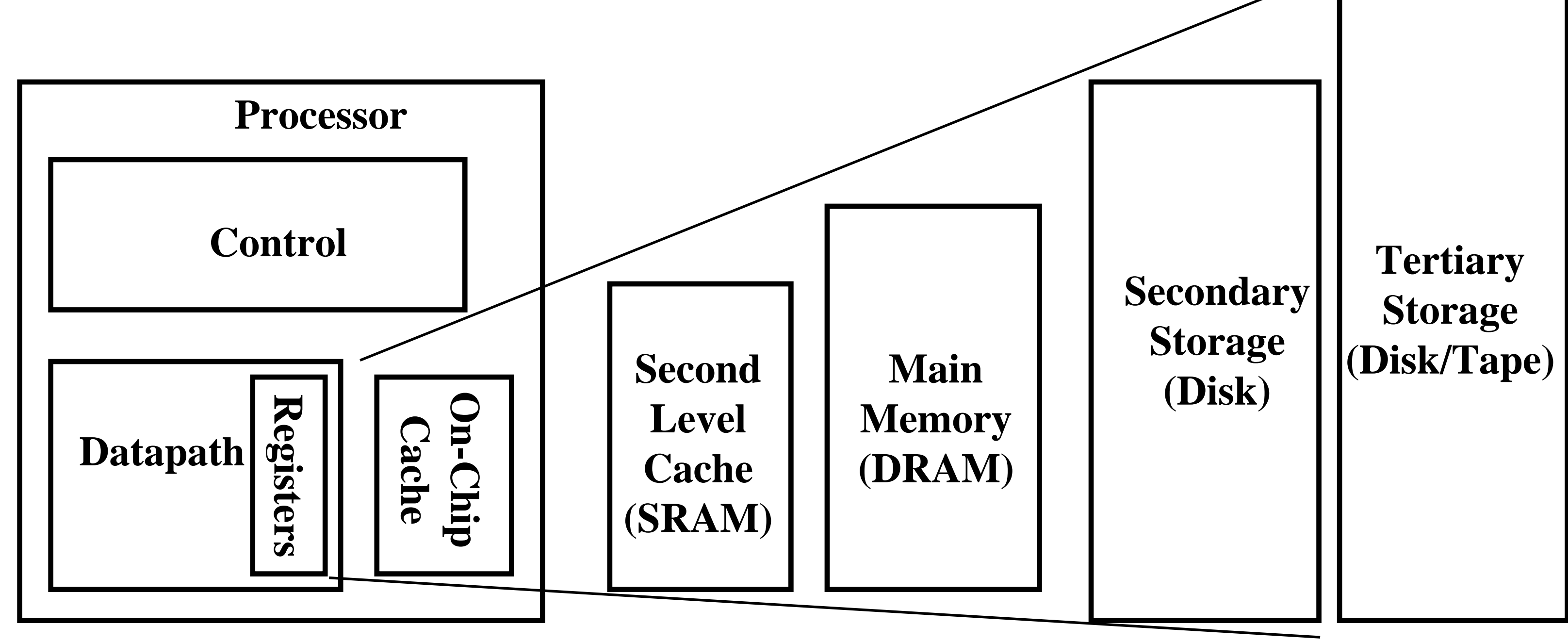
Relationship of Caching and Pipelining



•

A Modern Memory Hierarchy

- By taking advantage of the principle of locality:
  - Present the user with as much memory as is available in the cheapest technology.
  - Provide access at the speed offered by the fastest technology.
- Requires servicing faults on the processor



Speed (ns):	1s	10s	100s	10,000,000s (10s ms)	10,000,000,000s (10s sec)
Size (bytes):	100s	Ks	Ms	Gs	Ts

The Other 90% of Architecture

- Longer Pipelines
  - The Prescott Pentium 4 CPU has a 31 stage pipeline
- Wider Pipelines and Speculation
  - Superscalar – Multi-issue
  - Speculate with branch prediction
  - Out of Order Execution
- Caches and Buffers
  - Up to 3 levels of caches + specialized caches
  - Memory Buffers and Reservation Stations
- Multiple Everything
  - Multithreading
  - Multiprocessor System On Chip

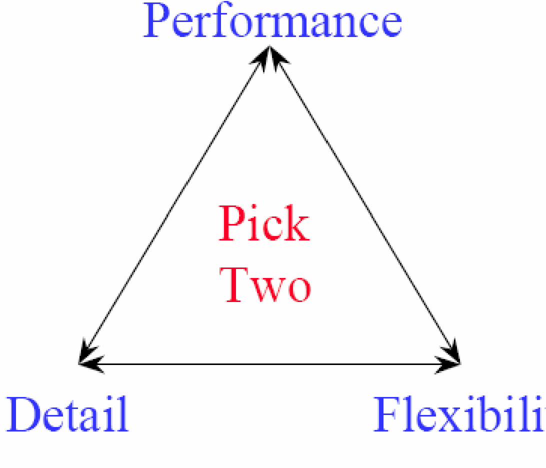
Outline

- Introduction
- Processor Modeling
  - SimpleScalar
  - Liberty Simulation Environment
  - Metropolis Processor Modeling
- Use of Processor Modeling in Embedded Systems
- Conclusions

SimpleScalar Overview

- The Standard for Microarchitectural Simulation
  - First Released in 1996
  - Developed by Todd Austin and Doug Burger
  - Multiple Levels of Models for Accuracy
  - Written in low-level high-performance sequential C-code
- Supports a Variety of Instruction Sets
  - Alpha, ARM, PowerPC, (x86)
- Supports a Variety of Microarchitectural Features

The Zen of Simulator Design



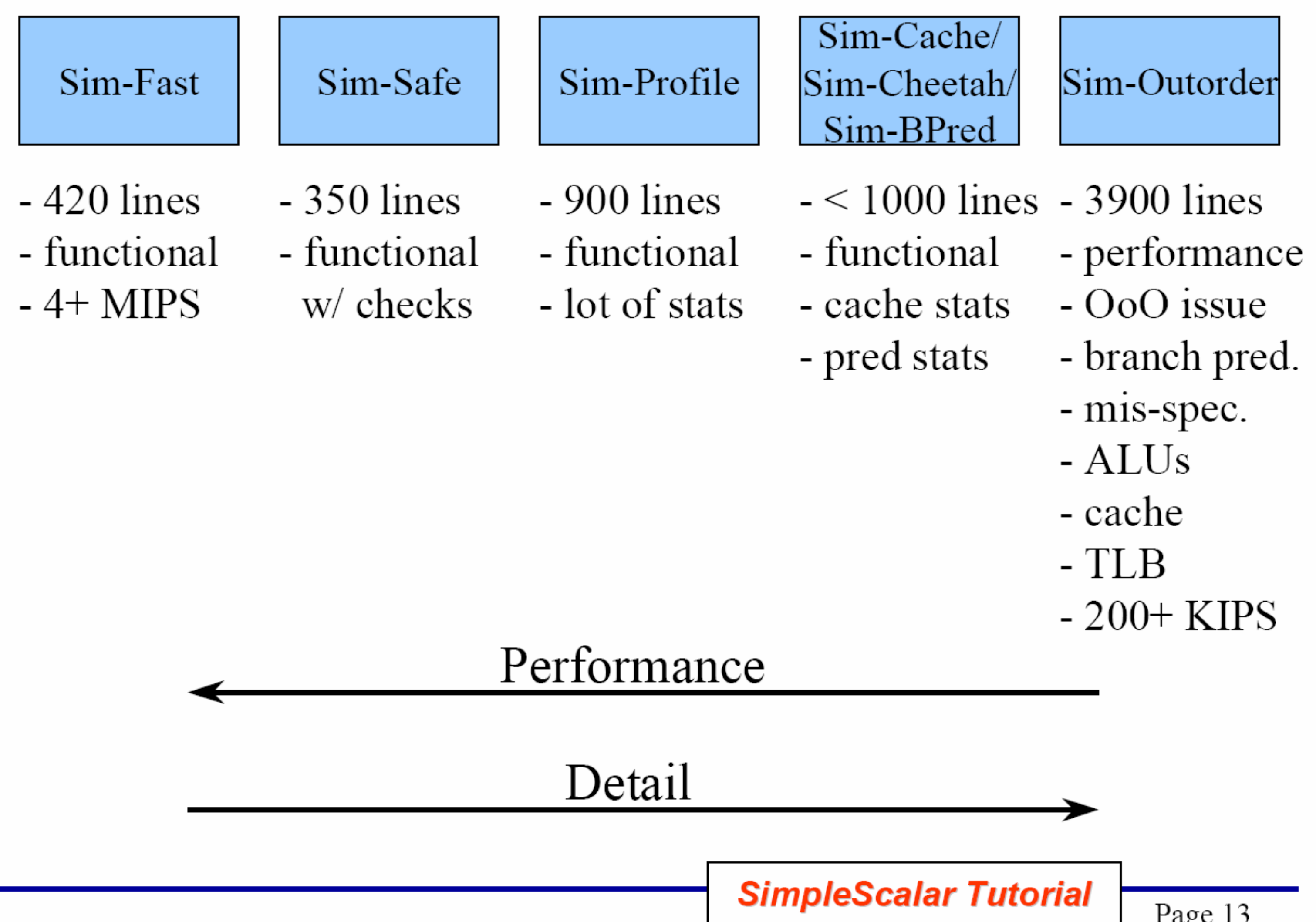
Performance: speeds design cycle

Flexibility: maximizes design scope

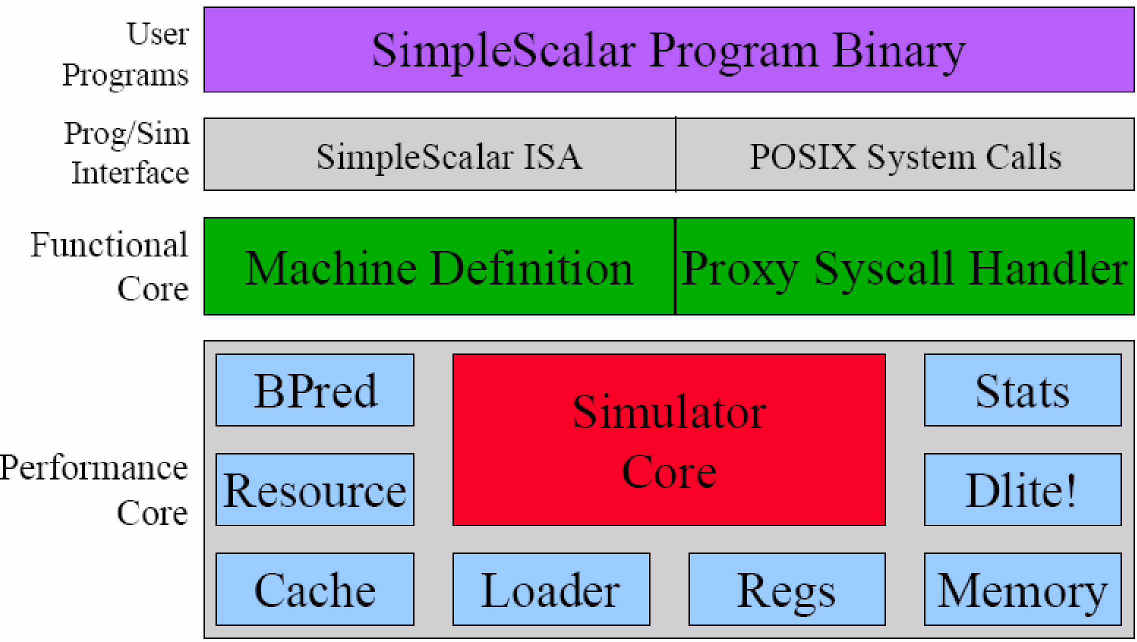
Detail: minimizes risk

- *design goals* will drive which aspects are optimized
- the SimpleScalar Tool Set
  - optimizes performance and flexibility
  - in addition, provides portability and varied detail

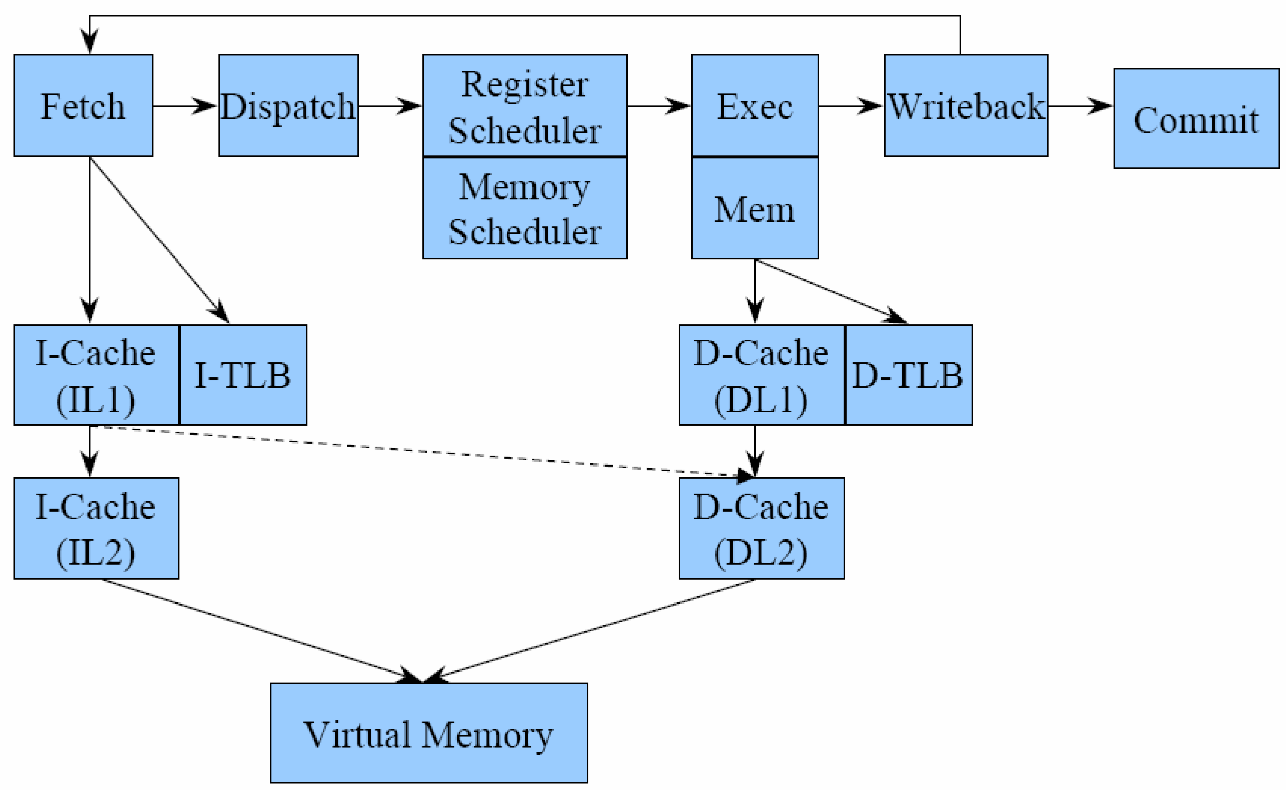
Taken From: http://www.simplescalar.com



Taken From: http://www.simplescalar.com



Taken From: http://www.simplescalar.com



Taken From: http://www.simplescalar.com

Main Simulation Loop

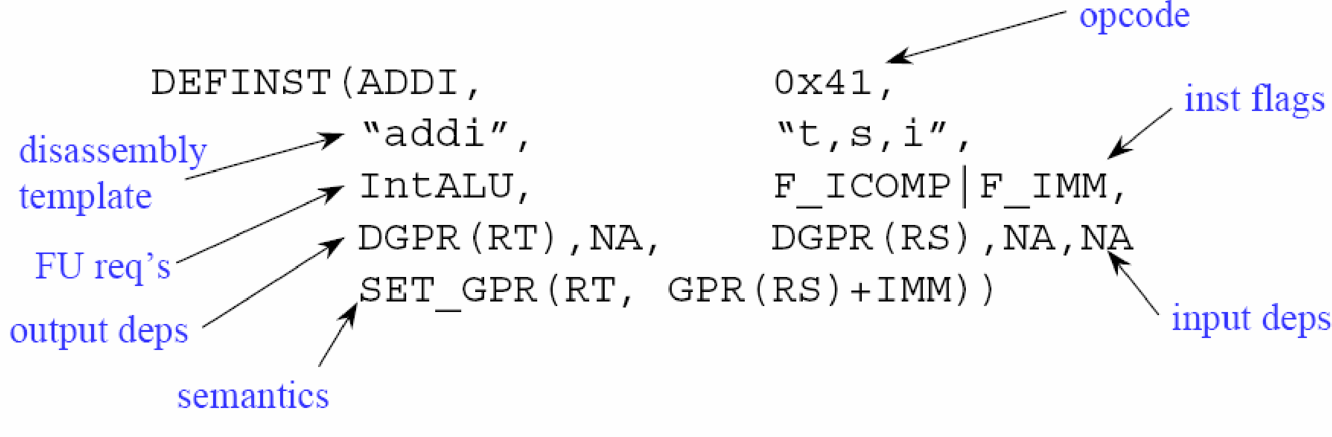
```
for (;;) {
  ruu_commit();
  ruu_writeback();
  lsq_refresh();
  ruu_issue();
  ruu_dispatch();
  ruu_fetch();
}
```

- main simulator loop is implemented in `sim_main()`
- walks pipeline from Commit to Fetch
  - backward pipeline traversal eliminates relaxation problems, e.g., provides correct inter-stage latch synchronization
- loop is exited via a `longjmp()` to `main()` when simulated program executes an `exit()` system call

Taken From: http://www.simplescalar.com

Machine Definition File (ss.def)

- a single file describes all aspects of the architecture
  - used to generate decoders, dependency analyzers, functional components, disassemblers, appendices, etc.
  - e.g., machine definition + ~30 line main = functional sim
  - generates fast and reliable codes with minimum effort
- instruction definition example:



Taken From: http://www.simplescalar.com

SimpleScalar Conclusions

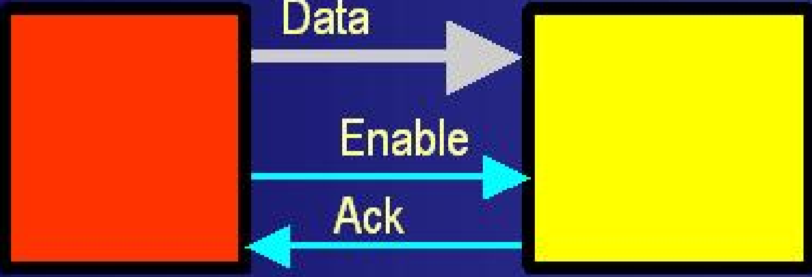
- Solid Framework for Microarchitectural Research
  - Used for ~33% of all Computer Architecture Papers
  - Good for examining new microarchitectural features
  - Fast and Reliabile
- But…
  - Difficult to Retarget and Modify
  - Monolithic – hard to use with other tools…
  - Core Execution Semantics hard to modify
  - Purely Sequential MOC.

Liberty Simulation Environment

- A Next-Generation Microarchitectural Environment
  - From Prof. David August's Princeton Research Group
  - Compiler and Simulator Framework
  - Advanced Language Features
- Structural Specification
  - Extended Polymorphism
- Custom Model of Computation
  - Composability
  - Potential for Optimized Simulation

Liberty Simulation Environment

- Simulator construction system for high reuse
- Two-tiered specifications
  - Leaf module templates in C
  - Netlisting language for instantiation and customization
- Three-signal standard communications contract with overrides (control functions)

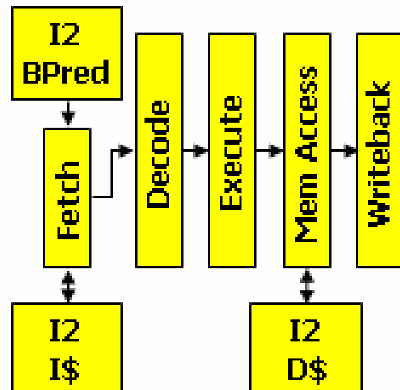


- Code is generated

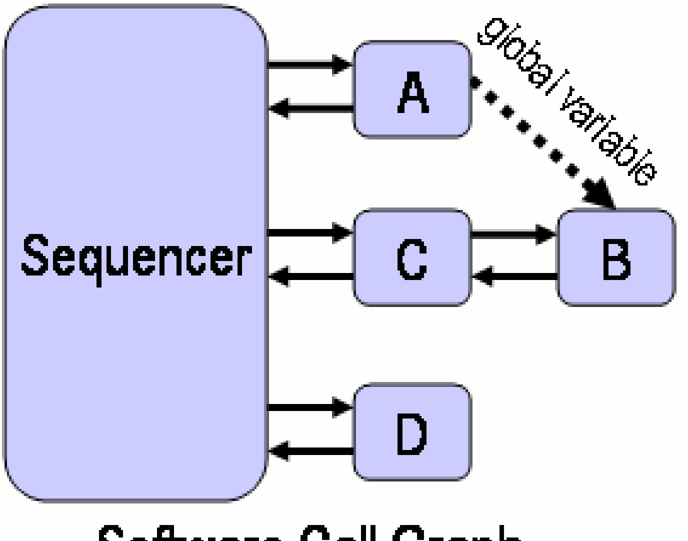
Taken From: http://liberty.cs.princeton.edu

LSS – A Natural Specification Language

- Modularize the model like HW
- Basic components
  - Concurrent computation
  - Communication through ports
- Called structural modeling
- Cornerstone of LSS design



- Compare to C/C++ approach
  - Function encapsulation ≠ hardware block encapsulation
  - Requires re-divide and re-conquer
  - Leads to [MICRO-35]
    - Inaccuracies
    - Long development times

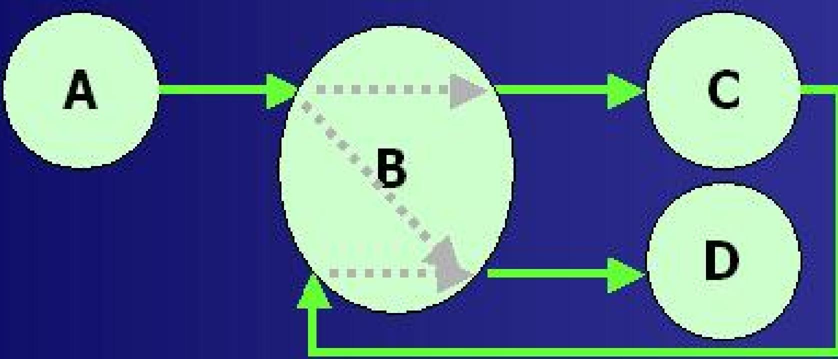


Software Call Graph

Taken From: http://liberty.cs.princeton.edu

Models of Computation

- System C uses Discrete Event (DE)
- LSE uses Heterogenous Synchronous Reactive (HSR)
  - Edwards (1997)
  - Unparsed code blocks (black boxes)
  - Values begin *unresolved* and resolve monotonically
  - Chaotic scheduling

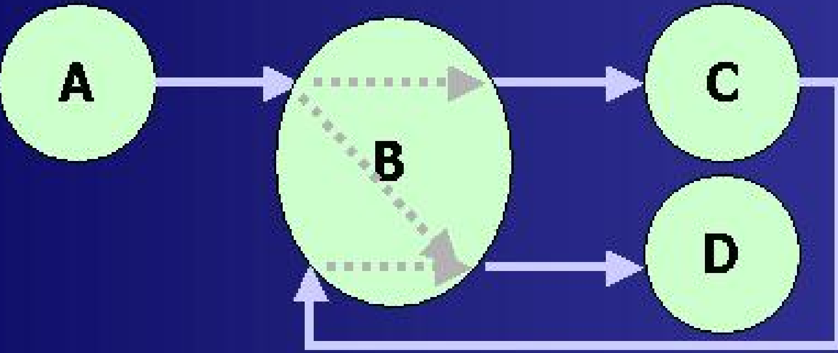


7

Taken From: http://liberty.cs.princeton.edu

Creating Static Schedules

- Edwards' algorithm (1997)
  - Construct a signal dependency graph
  - Break into strongly-connected components (SCC). Schedule in topological order
  - Partition each SCC into a head and tail
  - Schedule tail recursively, then repeat head (any order) and tail's schedule
  - Coalesce



12

Taken From: http://liberty.cs.princeton.edu

Reuse Penalty Revisited

Model	Cycles/sec	Speedup	Build time (s)
Custom SystemC	53722	-	49.1
Custom LSE	155111	2.88	15.4
Reusable LSE w/o optimization	40649	0.76	33.9
Reusable LSE with optimization	57046	1.06	34.4

- Reuse penalty mitigated in part

Reusable LSE model 6% faster than custom SystemC

Taken From: http://liberty.cs.princeton.edu

Component Flexibility

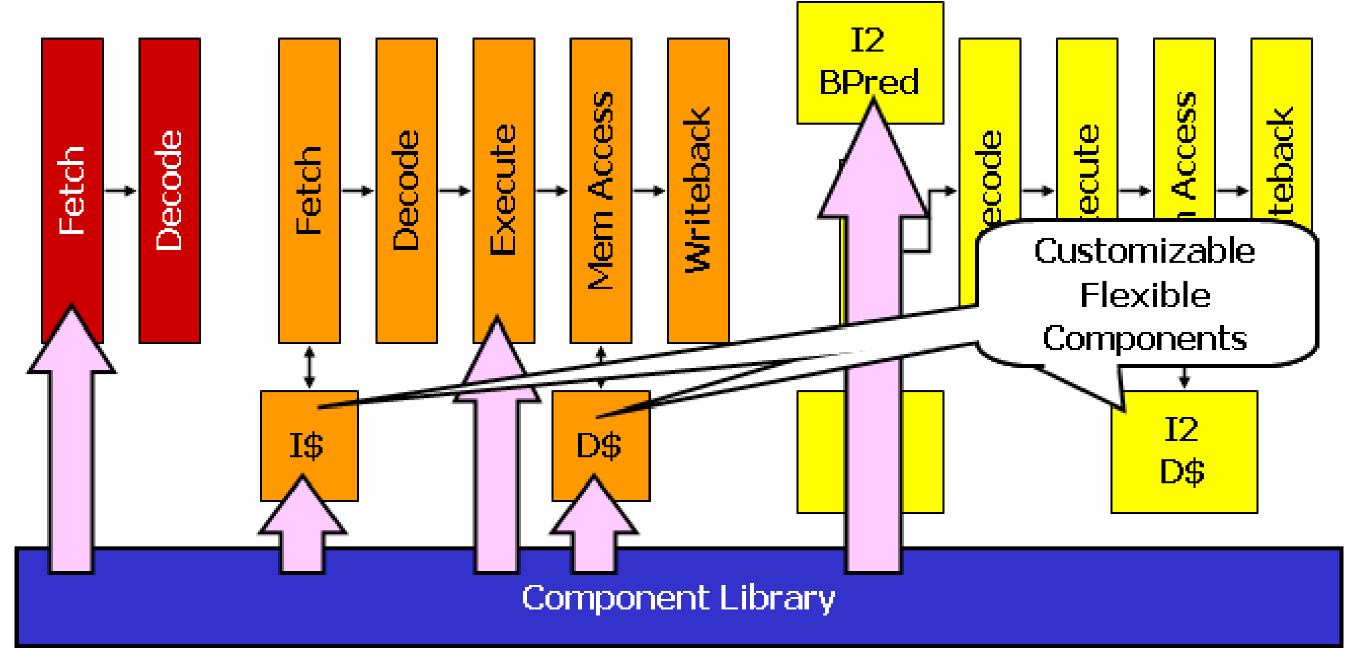
Polymorphism

- Many state and routing components
  - Share identical functionality
  - Store different data types
- Polymorphism allows components to adapt
- But, polymorphism forces over 101 type instantiations for the Itanium 2

Taken From: http://liberty.cs.princeton.edu

The Liberty Structural Specification Language

Accelerate Modeling with Reuse



- Reuse components to amortize costs [Charest '02, Emer '02, Koegst '98]
- 80% of I2 model's 183 components from library of 22
  - Similar level of reuse for other non-Itanium processor models
- Low-overhead customizability critical for this reuse [Radetzki '98]

Taken From: http://liberty.cs.princeton.edu

Liberty Conclusions

- This improves upon SimpleScalar in that…
  - Modular and Structural
  - Domain Specific Language
  - Simulator and Compiler Generation
- But…
  - Large learning curve
    - Arcane entry languages
    - Complex communication protocol
  - Retargeting capabilities are unclear
  - Still a monolithic environment

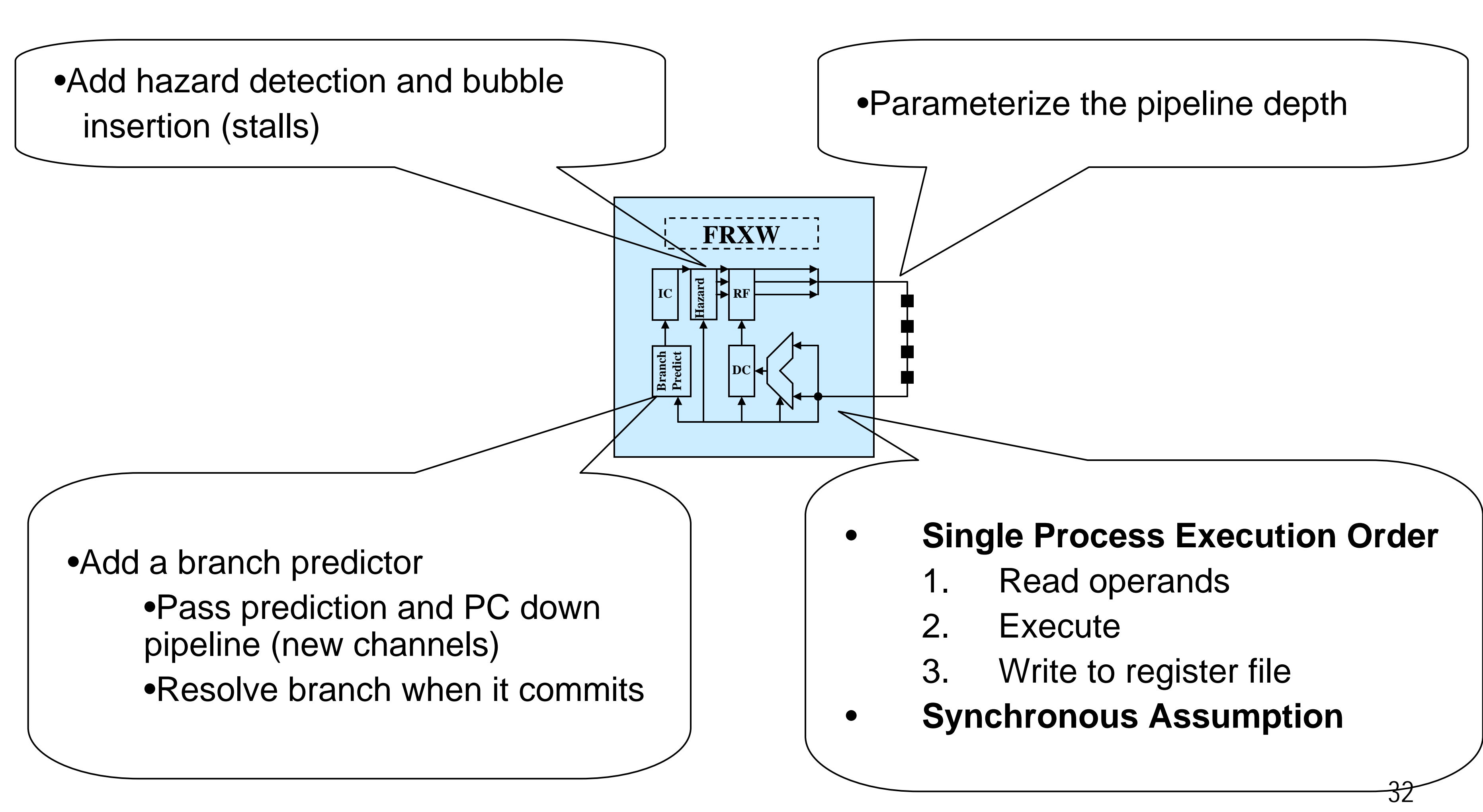
Modeling Microprocessors in Metropolis

- Focus on Microarchitectural Design Space Exploration in the context of a System-Level Design framework
  - Intuitive MOC and Simplified Modeling Methodology
  - Connectivity to other tools
  - Retargetability
- Outline
  - Modeling using Kahn Process Networks
  - ARM Processor Modeling
  - Instruction Set Retargeting

Modeling with YAPI + KPN

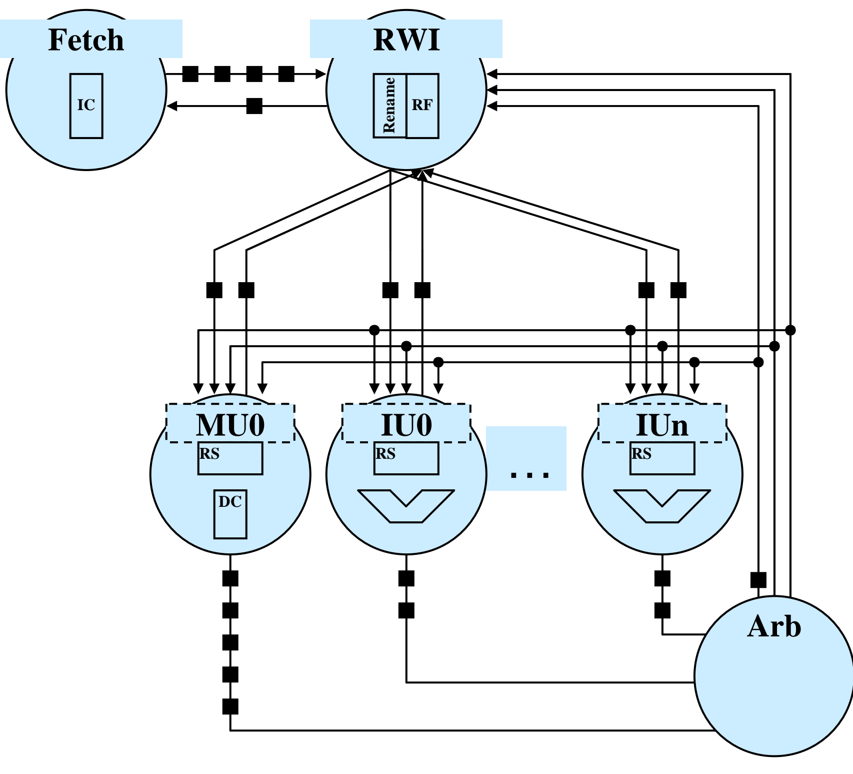
- Kahn Process Networks
  - Processes communicating via unbounded FIFO's
  - Blocking Reads / Unblocking Writes
  - Fully deterministic
  - No notion of time
- YAPI
  - Extension of KPN
  - Non-deterministic select
  - Refinement to bounded FIFO's
- Our Work
  - Characteristics
    - Synchronous assumption
    - Keeps FIFO lengths fixed
    - Separation of function and timing
  - Microarchitectural Models
    - Single Process Model
    - Out of Order Execution Model
    - 2 Process ARM Models
      - XScale + Strongarm
    - Abstract Speculative OOE Model

Single Process YAPI Model*



* Collaboration With: Sam Williams

Out-of-Order Architectures

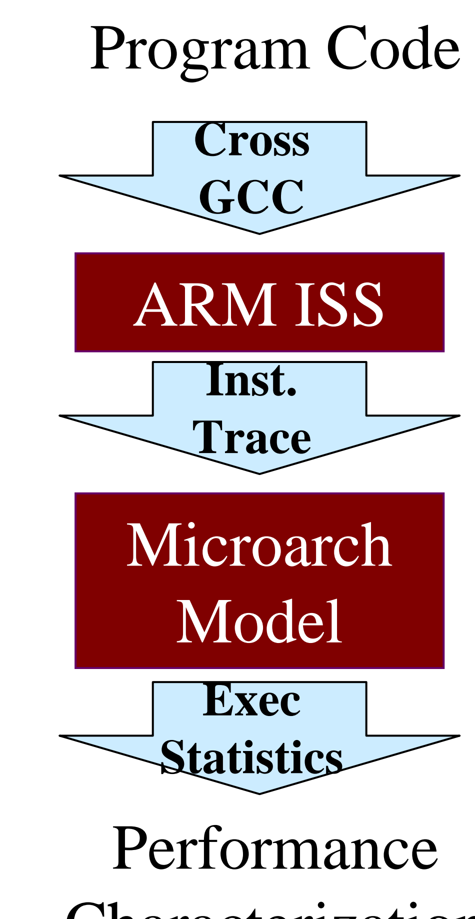


- Tomasulo style register renaming
- Highly Parameterizable
  - #ports, # integer units, depth, etc.
- Broadcast nature handled with multiple copies of each channel
- ReadWriteIssue must maintain knowledge of which instructions can be issued to which functional units
- All execution units are derived from the Station class
- N-way Super scalar processor was realized by changing the depth of the instruction channel (depth can be treated as width) from Fetch

* Collaboration With: Sam Williams

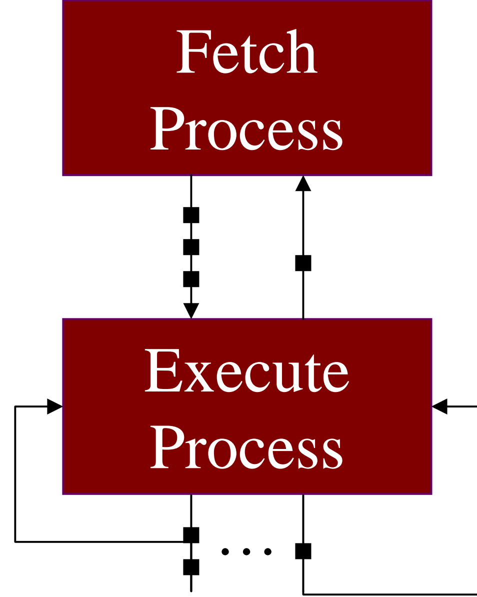
ARM Modeling Overview

- Separate between Microarchitectural Performance Model and Program Execution
- We Only Need to Model
  - Operand and Condition Code Dependencies
  - Branch Results
  - Execution Latencies
  - Forwarding Latencies
- Trace Contains
  - Every Instruction
    - Program Counter
    - Read + Write Operands (including cond. codes)
    - Instruction Type
  - (Optional) Data Addresses Accessed
- Advantages
  - Higher execution speed
  - Simplified, reusable microarchitectural modeling



Double Process Model

- Needed For:
  - Modeling Forwarding
  - Modeling Variable Instruction Latencies
- Leverages FIFO's for modeling delays
  - Preexecution Delay
    - Fetch, Decode, etc.
  - Execution Delay
    - Multiple Latencies, Forwarding
  - Synchronization
  - Stalls
    - Issue Stalls
      - Branch Misprediction
      - ICache Misses
    - Result Stalls
      - Operand Dependencies

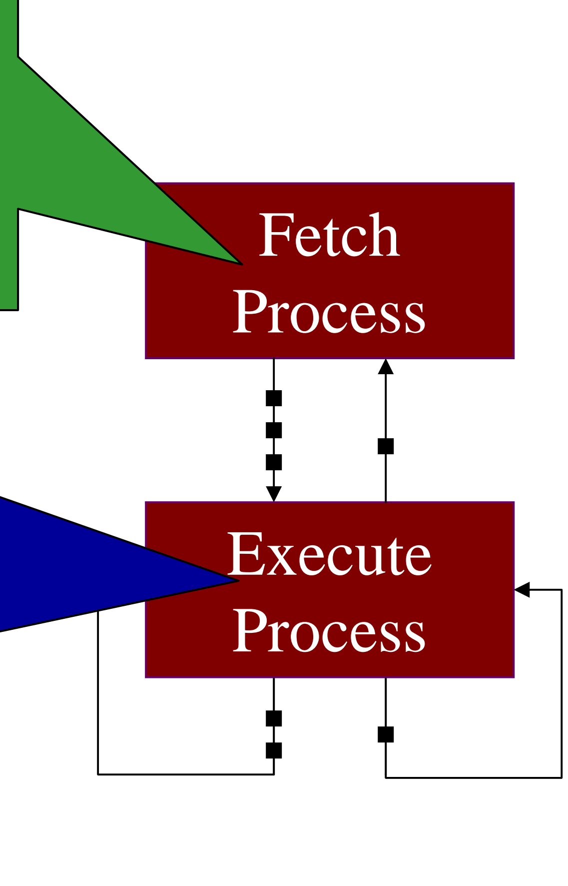


Double Process

- Needed For:
  - Modeling Forwarding
  - Modeling Variable Ins
- Leverages FIFO's for m
  - Preexecution Delay
    - Fetch, Decode, etc.
  - Execution Delay
    - Supports Multiple Latencies, Forwarding
  - Synchroniz

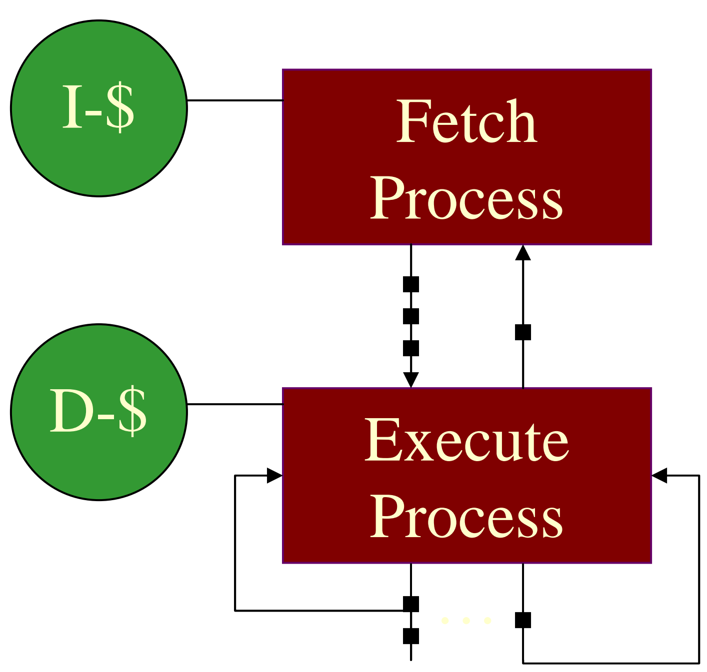
```
Preload_fifo();
While(true) {
    stall = stall_in.read();
    check_mispredict(stall);
    if (!stall) {
      inst = fetch(inst_num);
      if (inst.type == branch)
          branch_pred(inst);
      inst_out.write(inst);
 }  }
```

```
Preload_each_results_fifo();
While(true) {
  read_results();
  if (stall == 0)
     ReadInst = FetchedInst.read();
   stall = check_stall();
   compute_memory();
   DoStall.write(stall);
   write_results();
   cycle_count++;
}  }
```

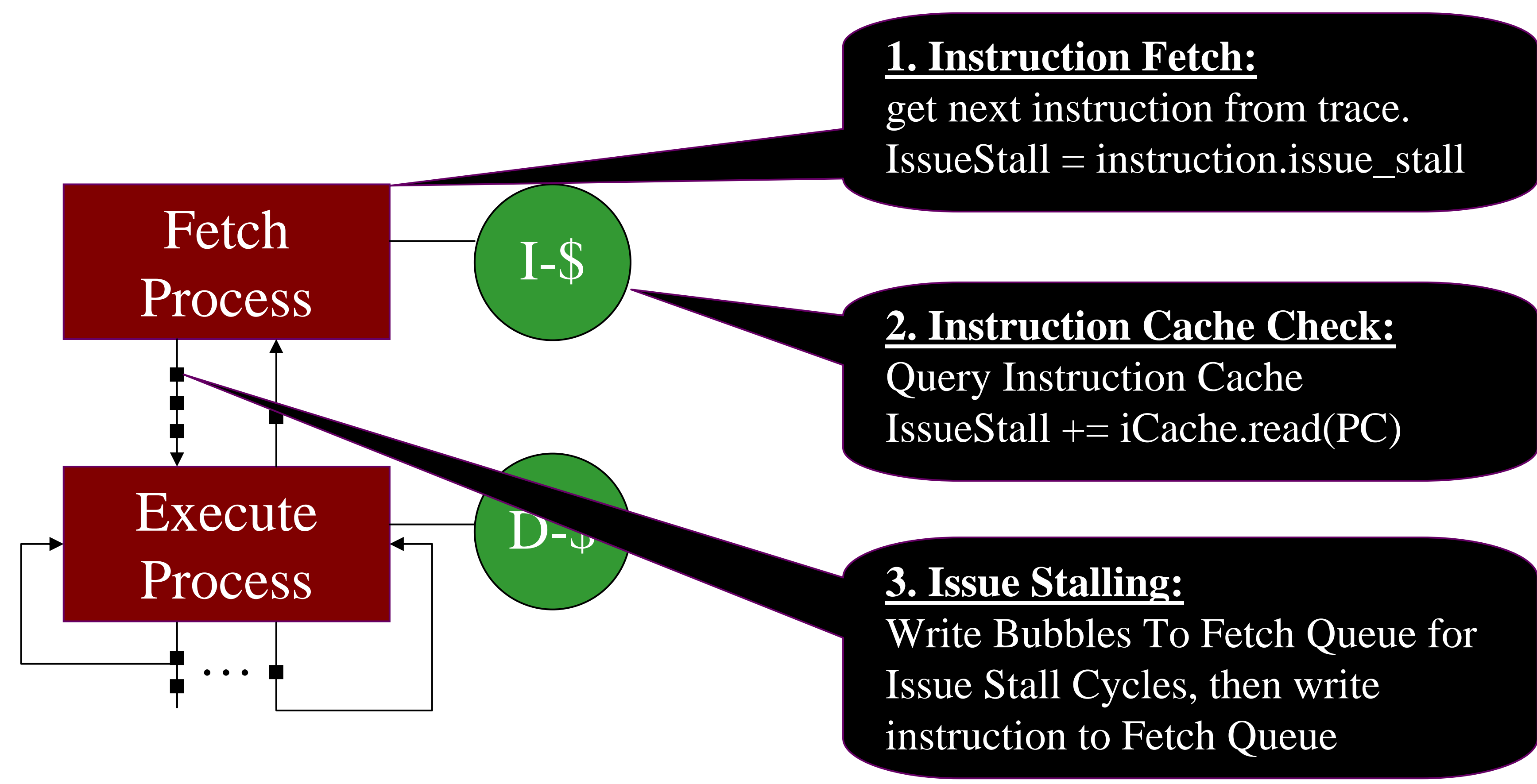


Models with Memory

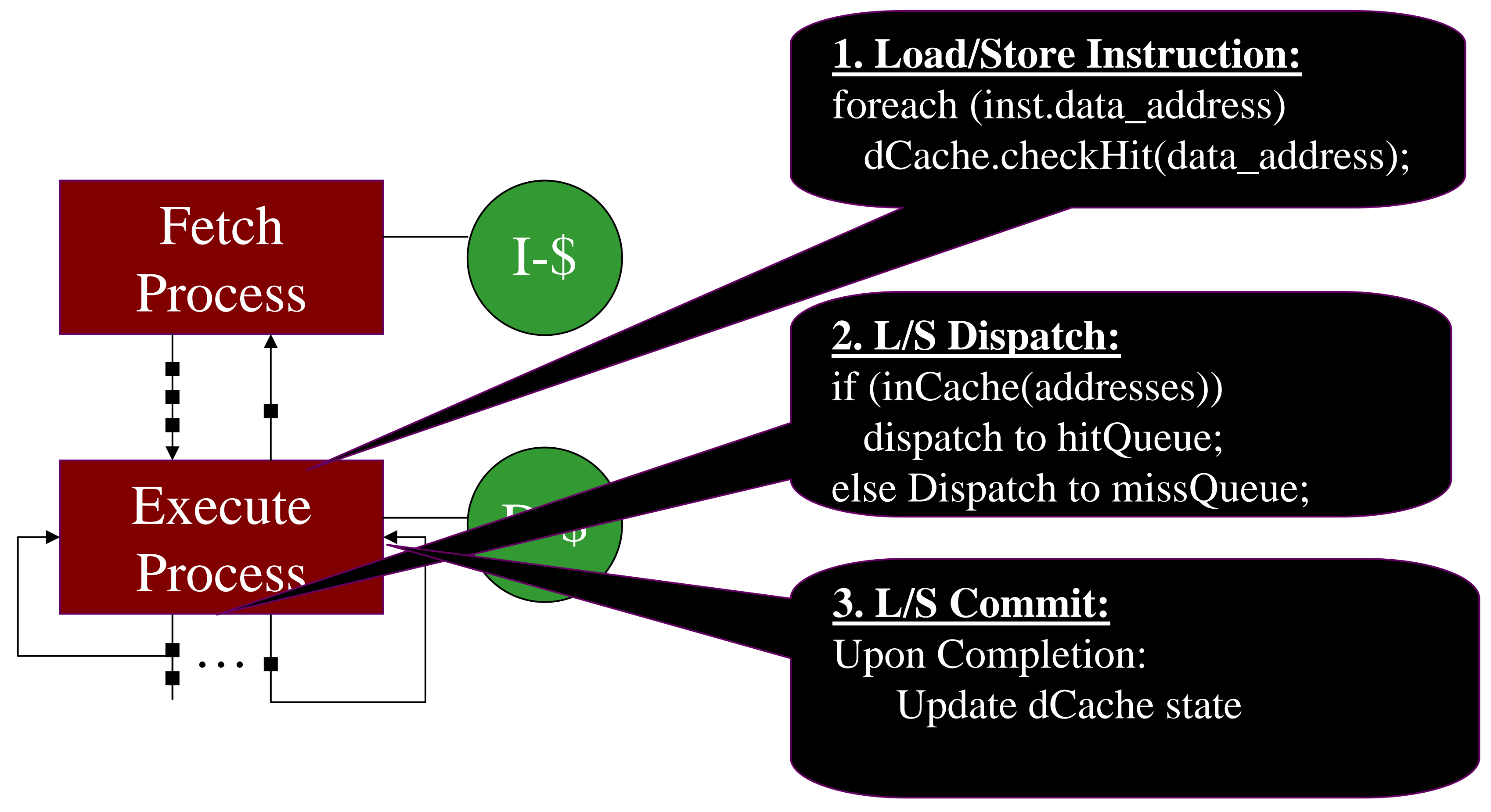
- Features
  - Cache Models
    - Associative
    - Perfect
    - Statistical
  - Translation Lookaside Buffers
  - Data Cache Write Buffers
  - Shared Bus between Caches
- Close to Simplescalar Models ARM
  - 15% for XScale
  - 25% for Strongarm
- Still Missing
  - Instruction Buffer



ICache Usage

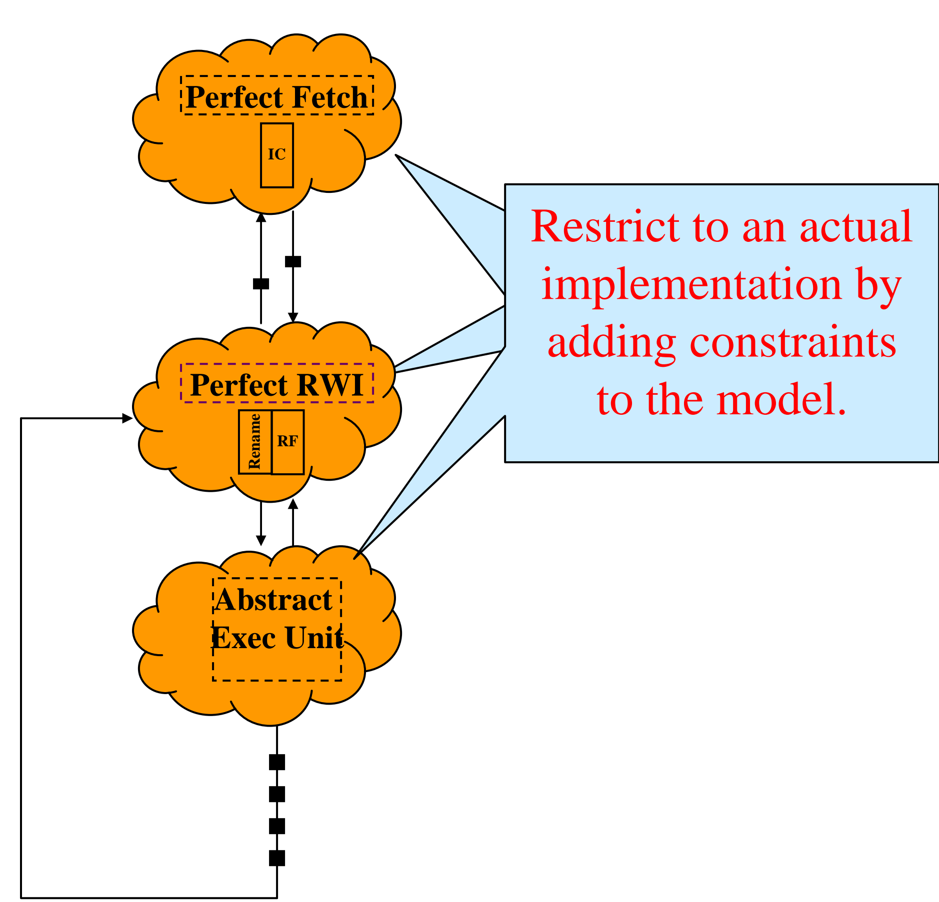


DCache Usage



An Abstract Speculative Model

- Currently under development in collaboration with Haibo Zeng and Qi Zhu {zenghb, zhuqi}@eecs.berkeley.edu
- Adding in Speculation + OOE as an afterthoughts can be difficult
  - Why not begin with it and then constrain to a real implementation?
- Assume Perfect Model (for a given fetch width)
  - Branch prediction
  - Perfect Memory and Register Files
  - Unlimited Execution Resources and Forwarding
- Analyze Performance for Different Applications
  - Parallelism
  - Resource Usage
  - Etc.



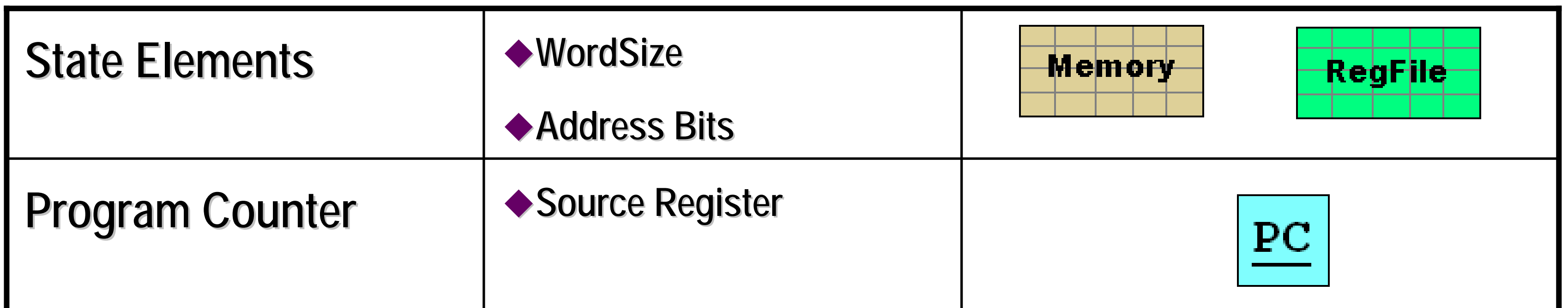
ISA_ML Overview

- Main Parts
  - A Visual Instruction Set Description Language
    - Currently one describes the encoding of instructions
    - Written using GME*, a UML-based environment for constructing domain specific modeling environments
  - Generates a C++-based disassembler and trace-interface code for the given model ISA description
- Key Features:
  - Two high level models
    - ISA State: Register Files, Memories, Program Counter, etc.
    - Instructions: Encoding and operand fields of each instruction
  - Intuitive Visual Interface
  - Leverages Hierarchy + Compact Representation
  - Extensive Error Checking
  - Easy to Retarget to Output Other Formats (e.g. verilog, nML, etc)

Results	MIPS Integer Subset	PowerPC Integer Subset	ARM (approximate)
Base Instructions	10	12	6
Actual Instructions	55	80	26
Hours to Enter (appox.)	8	6	5
Header File (# lines)	1357	2134	759

To Appear in 2004 OOPSLA Workshop on Domain Specific Modeling, October 24th, 2004
Title: A Visual Language for Describing Instruction Sets and Generating Decoders
Authors: T. Meyerowitz, J. Sprinkle, A. Sangiovanni-Vincentelli

*GME website: http://www.isis.vanderbilt.edu/projects/gme/



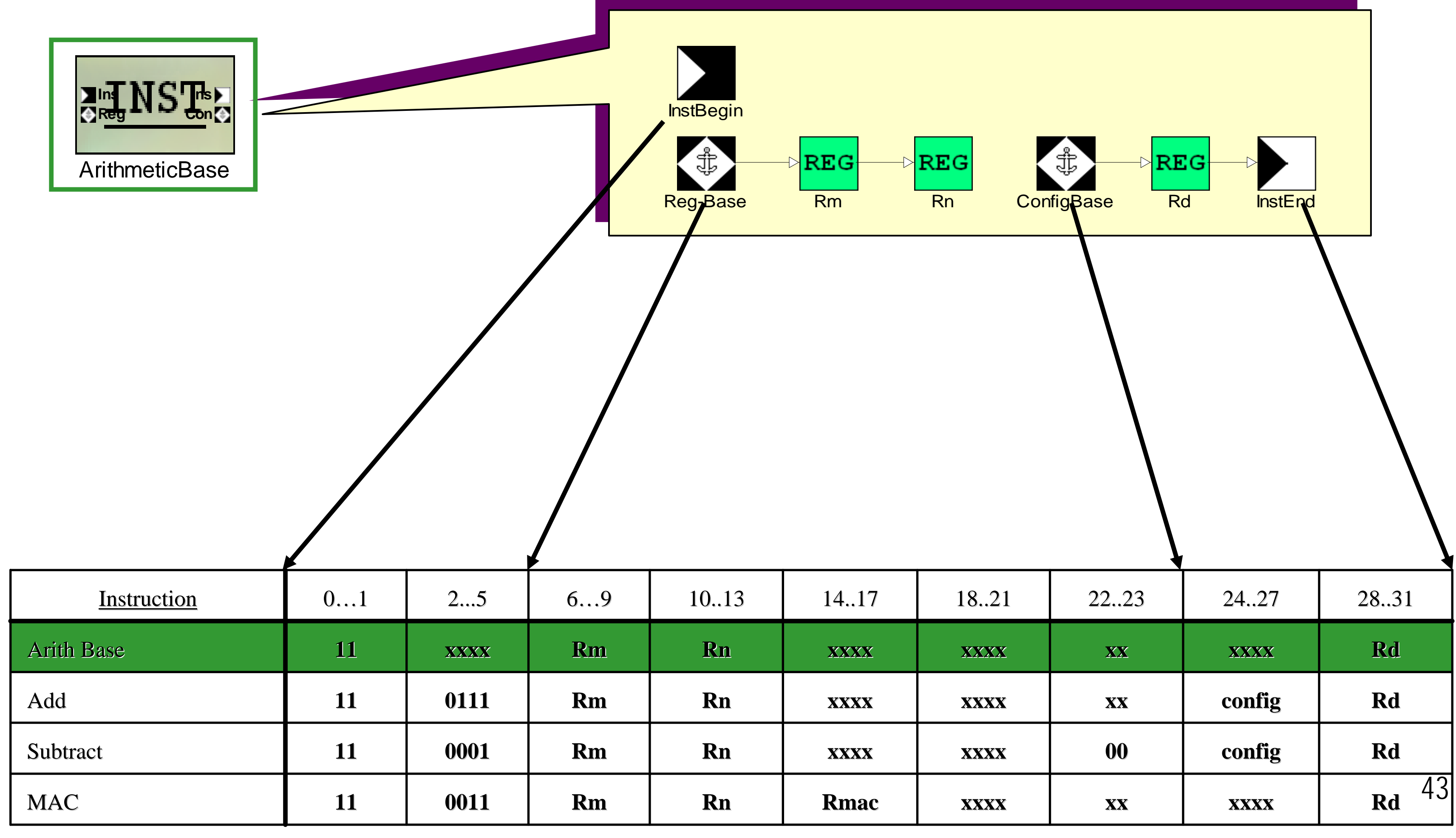
State Elements	◆WordSize ◆Address Bits	Memory RegFile
Program Counter	◆Source Register	PC

ISA_ML State Elements

Bitfield Operands	◆NumBits ◆Encoding ◆SingleEncoding	INST Instruction, MEM Memory Ref, IMM Immediate, CON Constant, REG Register Ref
Anchors	◆AnchorPoint	Begin Anchor, End Anchor, Custom Anchor
Connection	Specifies the ordering of bitfields	Ordering Connection

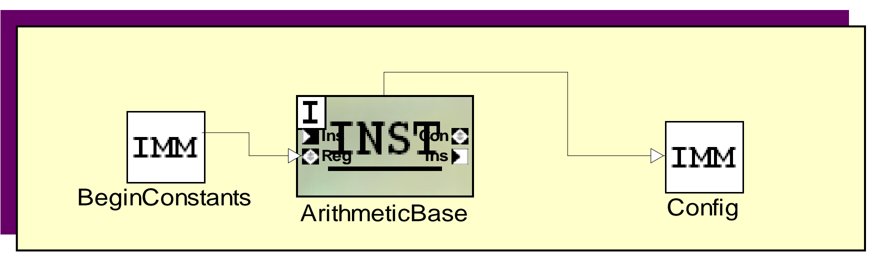
ISA_ML Instruction Elements

Sample Instructions: Base Instruction

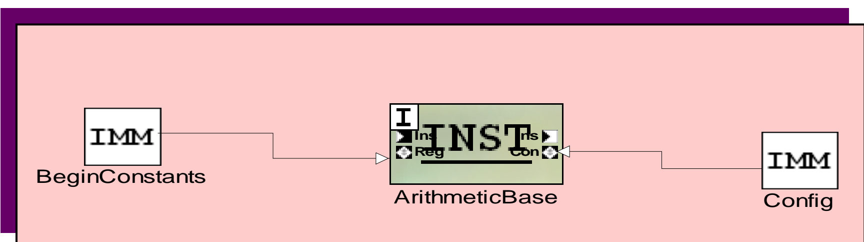


Instruction	0…1	2...5	6…9	10..13	14..17	18..21	22..23	24..27	28..31
Arith Base	**11**	**xxxx**	**Rm**	**Rn**	**xxxx**	**xxxx**	**xx**	**xxxx**	**Rd**
Add	**11**	**0111**	**Rm**	**Rn**	**xxxx**	**xxxx**	**xx**	**config**	**Rd**
Subtract	**11**	**0001**	**Rm**	**Rn**	**xxxx**	**xxxx**	**00**	**config**	**Rd**
MAC	**11**	**0011**	**Rm**	**Rn**	**Rmac**	**xxxx**	**xx**	**xxxx**	**Rd**

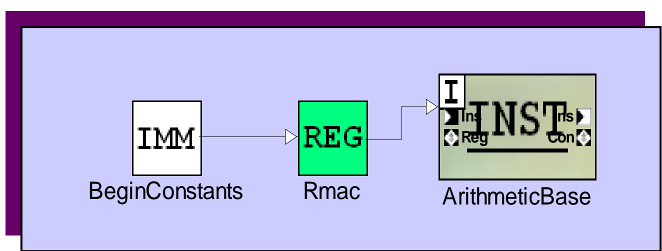
Sample Instructions: Other Instructions



Addition



Subtraction



Multiply Accumulate

Instruction	0…1	2…5	6…9	10..13	14..17	18..21	22..23	24..27	28..31
Arith Base	**11**	**xxxx**	**Rm**	**Rn**	**xxxx**	**xxxx**	**xx**	**xxxx**	**Rd**
Addition	**11**	**0111**	**Rm**	**Rn**	**xxxx**	**xxxx**	**xx**	**config**	**Rd**
Subtractraction	**11**	**0001**	**Rm**	**Rn**	**xxxx**	**xxxx**	**00**	**config**	**Rd**
Multiply Accumulate	**11**	**0011**	**Rm**	**Rn**	**Rmac**	**xxxx**	**xx**	**xxxx**	**Rd**

Modeling Microprocessors in Metropolis: Conclusions

- Improvements on Prior Approaches
  - More Abstract
  - More Retargetable and Modular
  - Methodology for Refinement and DSE
- But…
  - Ongoing accuracy comparison with other tools
  - Performance needs to improve
  - Currently requires an external ISS to drive it

Outline

- Introduction
- Processor Modeling
- Use of Processor Modeling in Embedded Systems
  - Different Levels of Modeling
  - Co-Simulation
  - Back-Annotation
- Conclusions

Accuracy vs Performance vs Cost

	Accuracy	Speed	$$$*
Hardware Emulation	+++	+-	---
Cycle accurate model	++	--	--
Cycle counting ISS	++	+	-
Dynamic estimation	+	++	++
Static spreadsheet	-	+++	+++

*$$$ = NRE + per model + per design

Traditional Cosimulation

Software Programs

Microprocessor Simulator

Interfacing Simulator

Hardware Descriptions

Hardware Simulator

- **Advantages**
  - Allows prototyping without actual hardware
  - Consistency between HW and SW models
- **Disadvantages**
  - Overhead for having 2+ simulators
  - Often requires custom microprocessor models
  - Doesn't scale well for multiprocessor systems

Co-Simulation in Metropolis

Application | **Mapping** | **Architecture**

Program Code, Sync Points, and Data Arguments

Performance Information

Application Process

Other Application Processes

uArch Model

Functional (ISS) Model

Timing Model

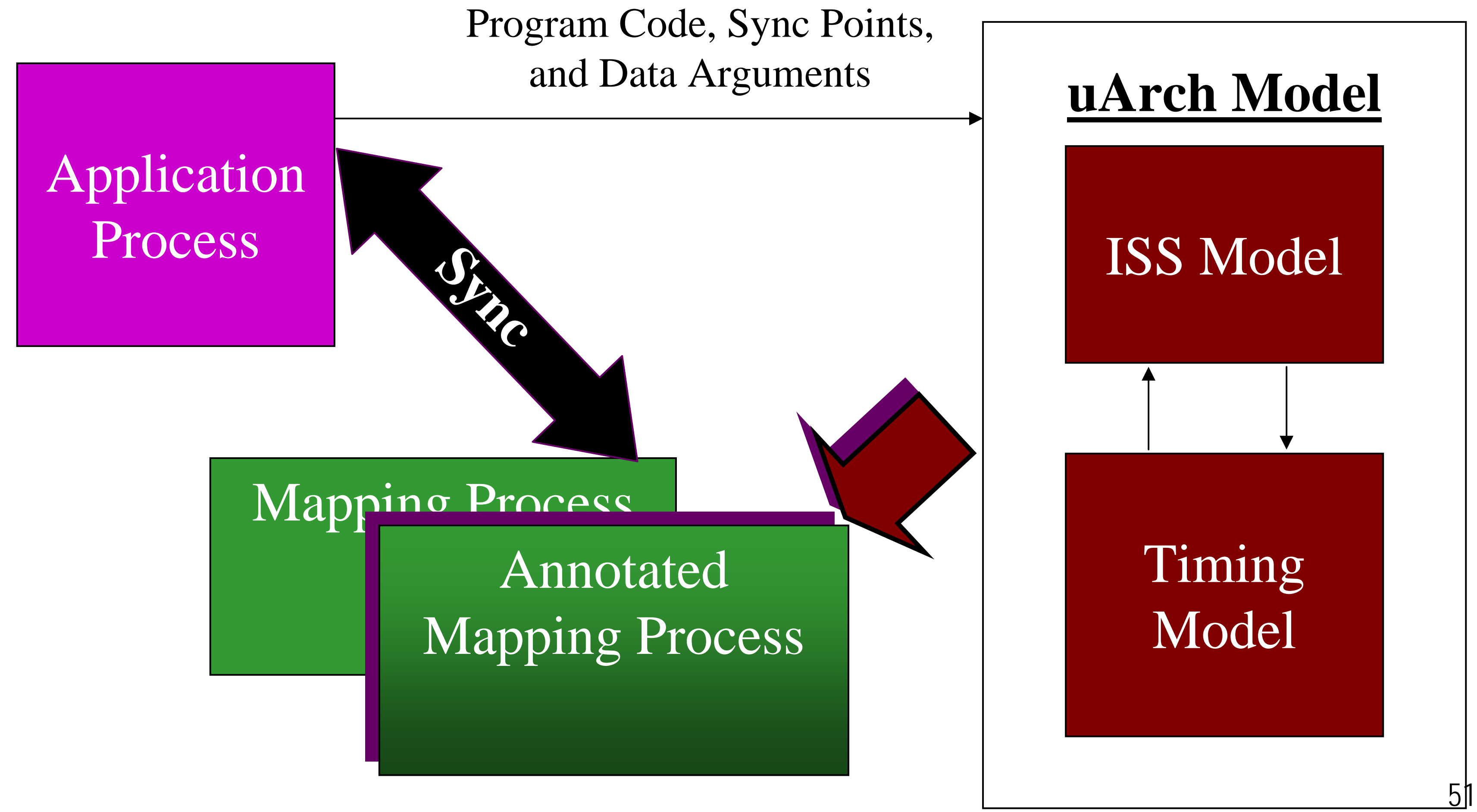
Other Architecture Components

Backwards Annotation

- Back Annotation Requirements
  - User-specified level of granularity
  - Flexibility for handling non-trivial interactions
    - RTOS's, Interrupts, Pipelining, Intra-process variation
  - Natural + Flexible Syntax
  - Function with Metamodel and Native Code

- Our Proposed Solution
  - Two functions to annotate the model code
    - CPU.BackAnnotate(begin_label, end_label, atomicity, (arguments))
    - CPU.BB_BackAnnotate(begin_label, end_label, atomicity, (arguments))
  - Handle complicated features at the system-level

Back Annotation: Overall Picture



Back Annotation: Example

Application Process

Mapping Process

```
In_data = InPort.ReadInputs();
Out_data = do_processing(In_data);
OutPort.WriteOutPuts()
```

```
CPU.read(IN_DATA_SIZE);
CPU.execute(CPU.back_annotate(
        do_processing.begin,
        do_processing.end, true));
CPU.write(OUT_DATA_SIZE)
```

```
CPU.execute(EXEC_TIME);
```

uArch Model
and
Back Annotator

CPU

Final Words

- Software is a key component in Embedded Systems
  - Fast and Accurate Modeling is Key
  - Time isn't the only factor to consider
    - Power, Memory Usage, Communication Usage, etc.

- Traditional Microarchitectural Environments are Unsuitable
  - Monolithic designs
  - Retargeting and integration issues

- We're developing an integrated approach within Metropolis

- What we haven't covered
  - Software Performance Estimation (Coming Soon…)
    - Estimate based on application, computation, and communication
  - Architecture Description Languages
  - Commercial Offerings
    - Mentor Graphics - Seamless
    - CoWare - ConvergenceSC + LISAtek
    - VaST Systems
    - Et al.