

Design and Implementation of the TRIPS EDGE Architecture

TRIPS Tutorial
Originally Presented at ISCA-32
June 4, 2005



Department of Computer Sciences
The University of Texas at Austin

Tutorial Outline









- **Part I: Overview**
 - Introduction (*Doug Burger/Steve Keckler*)
 - EDGE architectures and the TRIPS ISA (*Doug Burger*)
 - TRIPS microarchitecture and prototype overview (*Steve Keckler*)
 - TRIPS code examples (*Robert McDonald*)
- **Part II: TRIPS Front End**
 - Global control (*Ramdas Nagarajan*)
 - Control-flow prediction (*Nitya Ranganathan*)
 - Instruction fetch (*Haining Liu*)
 - Register accesses and operand routing (*Karu Sankaralingam*)
- **Part III: TRIPS Execution Core & Memory System**
 - Issue logic and execution (*Premkishore Shivakumar*)
 - Primary memory system (*Simha Sethumadhavan*)
 - Secondary memory system (*Changkyu Kim*)
 - Chip- and system-level networks and I/O (*Paul Gratz*)
- **Part IV: TRIPS Compiler**
 - Compiler overview (*Kathryn McKinley*)
 - Forming TRIPS Blocks (*Aaron Smith*)
 - Code optimization (*Kathryn McKinley*)
- **Part V: Conclusions**

Names in italics refer to both the presenters at ISCA-32 and the main developers of each section's slide deck.

Copyright Information

- The material in this tutorial is Copyright © 2005-2006 by The University of Texas at Austin. All rights reserved.
- This material was developed as a part of the TRIPS project in the Computer Architecture and Technology Laboratory, Department of Computer Sciences, at The University of Texas at Austin. Please contact Doug Burger (doug@cs.utexas.edu) or Steve Keckler (skeckler@cs.utexas.edu) for information about redistribution or usage of these materials in other presentations.
- Portions of the TRIPS technology described in this tutorial are patent pending. Commercial parties interested in licensing or using TRIPS technology for profit should contact the project principal investigators listed above.

Acknowledgment of TRIPS Sponsors

-  DARPA
 - Polymorphous Computing Architectures (2001-2006)
 - Contracts F33615-01-C-1892 and F44615-03-C-4106
 - Special thanks to Bob Graybill
-  Air Force Research Laboratory
-  Intel Research Council
 - 2 Research Grants (2000-2004) and equipment donations (2000-2005)
-  IBM
 - Faculty Partnership Awards (1999-2004) and SUR grants (1999, 2003)
-  Sun Microsystems
 - Research Grant (2003)
-  National Science Foundation
 - CAREER and Research Infrastructure Grants (1999-2008)
-  Peter O'Donnell Foundation
-  UT-Austin College of Natural Sciences
 - Matching funds for industrial fellows, infrastructure grants (1999-2008)

Other Members of the TRIPS Team

Research Scientists

- **Bill Yoder**
 - Chief developer of the TRIPS toolchain
- **Jim Burrill**
 - Chief engineer of the Scale compiler
- **Nick Nethercote**
 - TRIPS compiler performance leader

Graduate Students

- **Xia Chen**
 - PowerPC to TRIPS translation and emulation
- **Raj Desikan**
 - Initial data tile design
- **Saurabh Drolia**
 - Prototype DMA controller, system simulator, parallel software
- **Madhu Sibi Govindan**
 - Prototype external bus and clock controller
- **Divya Gulati**
 - Execution tile design, processor core verification
- **Heather Hanson**
 - Operand router design
- **Sundeeep Kushwaha**
 - Back-end instruction placement
- **Bert Maher**
 - Hyperblock construction and formation
- **Suriya Narayanan**
 - Compiler/memory system optimization
- **Sadia Sharif**
 - TRIPS system software

Relevant TRIPS/EDGE Publications

- "The Design and Implementation of the TRIPS Prototype Chip"
 - HotChips 17, 2005.
 - Contains details about the prototype ASIC.
- "Dynamic Placement, Static Issue (SPDI) Scheduling for EDGE Architectures"
 - Intl. Conference on Parallel Architectures and Compilation Techniques (PACT), 2004.
 - Specifies compiler algorithm for scheduling instructions on the TRIPS architecture.
- "Scaling to the End of Silicon with EDGE Architectures"
 - IEEE Computer, July, 2004
 - Provides an overview of EDGE architectures using the TRIPS architecture as a case study.
- "Exploiting ILP, TLP, and DLP with the Polymorphous TRIPS Architecture"
 - Intl. Symposium on Computer Architecture (ISCA-30), 2003.
 - Details the flexibility of the TRIPS architecture for exploiting different types of parallelism.
- "An Adaptive, Non-Uniform Cache Structure for Wire-Dominated On-Chip Caches"
 - Intl. Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-X), 2002.
 - The first paper describing NUCA caches and their design tradeoffs.
- "A Design Space Evaluation of Grid Processor Architectures"
 - Intl. Symposium on Microarchitecture (MICRO-34), 2001.
 - An early study that formed the basis for the TRIPS architecture.

Principal Related Work

- Transport-Triggered Architectures [Supercomputing 1991]
 - Hans Mulder and Henk Corporaal
 - MOVE Architecture had direct instruction-to-instruction communication bypassing registers
- WaveScalar [Micro 2003]
 - Mark Oskin, Steve Swanson, Ken Michelson, and Andrew Schwerin
 - Imperative/dataflow EDGE ISA
 - 3 most significant differences with TRIPS:
 - Dynamic instruction placement
 - All-path execution
 - Two-level microarchitectural execution hierarchy
- ASH/CASH/Pegasus IR [CGO-03, ASPLOS-04, ISPASS-05]
 - Mihai Budiu and Seth Goldstein
 - Similar goals and related compiler techniques
 - More of a focus on compiling to ASICs or reconfigurable substrates
- RAW [IEEE Computer-97, ISCA-04]
 - Pioneering tiled architecture tackling concurrency and wire delays
 - Leading work on Scalar Operand Networks
 - Direct instruction-to-instruction communication through network switch 1-stream

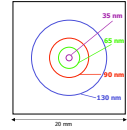
June 4, 2005

UT-Austin TRIPS Tutorial

7

Key Trends

- Power
 - Now a budget, just like area
 - What performance can you get for a given power and area budget?
 - Transferring work from the HW to the SW (compiler, programmer, etc.) is power efficient
 - Leakage trends must be addressed (not underlying goal of TRIPS)
- Slowing (stopping?) frequency increases
 - Pipelining has reached depth limits
 - Device speed scaling may stop tracking feature size reduction
 - May result in decreasing main memory latencies
 - Must exploit more concurrency
 - Key issue: how to expose this concurrency? Force the programmer?
- Wire Delays
 - Will force growing partitioning of future chips
 - Works against reduced power and improved concurrency
- Reliability
 - Do everything twice (or thrice) at some level of abstraction
 - Works against power limits and exploitation of concurrency

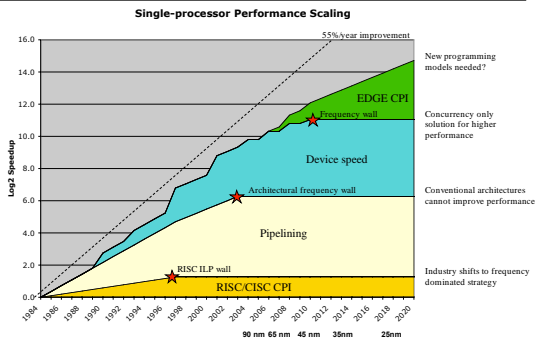


June 4, 2005

UT-Austin TRIPS Tutorial

8

Performance Scaling and Technology Challenges

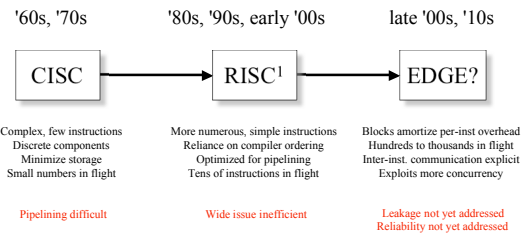


June 4, 2005

UT-Austin TRIPS Tutorial

9

EDGE Architectures: Can New ISAs Help?



¹RISC concepts implemented in both ISA and HW

June 4, 2005

UT-Austin TRIPS Tutorial

10

The TRIPS EDGE ISA

ISCA-32 Tutorial

Doug Burger



Department of Computer Sciences
The University of Texas at Austin

June 4, 2005

UT-Austin TRIPS Tutorial

11

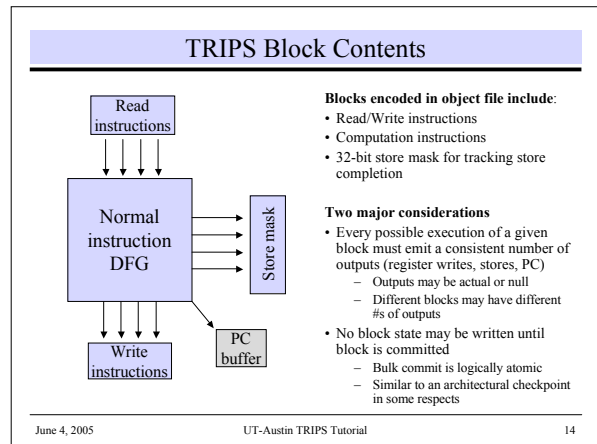
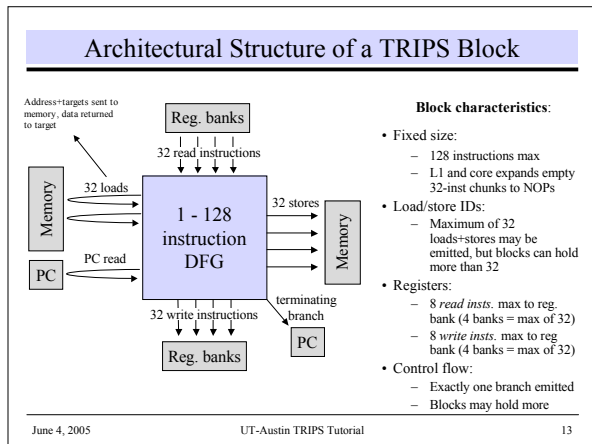
What is an EDGE Architecture?

- **Explicit Data Graph Execution**
 - Defined by two key features
- 1. Program graph is broken into sequences of *blocks*
 - Basic blocks, hyperblocks, or something else
 - Blocks commit atomically or not - a block never partially executes
- 2. Within a block, ISA support for direct producer-to-consumer communication
 - No shared named registers within a block (point-to-point dataflow edges only)
 - Caveat: memory is still a shared namespace
 - The block's dataflow graph (DFG) is explicit in the architecture
- What are design alternatives for different architectures?
 - Single path through program block graph (TRIPS)
 - All paths through program block graph (WaveScalar)
 - Mechanism for specifying instruction-to-instruction links in the ISA
 - Block constraints (composition, fixed size vs. variable size, etc.)

June 4, 2005

UT-Austin TRIPS Tutorial

12



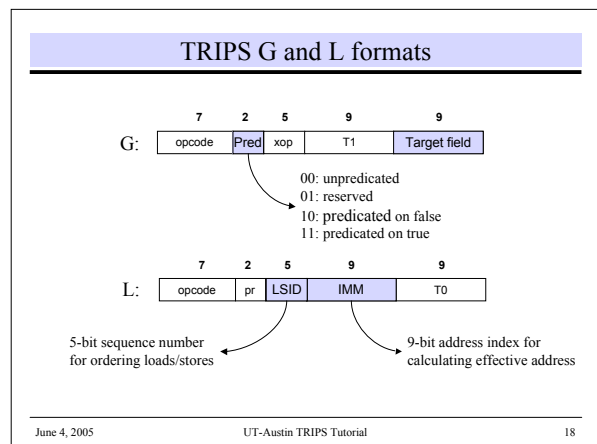
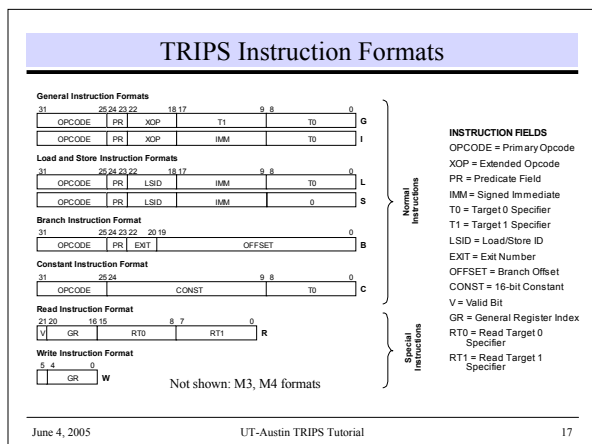
TRIPS Block Example

RISC code	TIL (operand format)	TASL (target format)
<pre>ld R3, 4(R2) add R4, R1, R3 st R4, 4(R2) addi R5, R4, #2 beqz R4, block3</pre>	<pre>.bbegin block1 read \$t1, \$g1 read \$t2, \$g2 ld \$t3, 4(\$t2) add \$t4, \$t1, \$t3 st \$t4, 4(\$t2) addi \$t5, \$t4, 2 teqz \$t6, \$t4 b_t<\$t6> block3 b_f<\$t6> block2 write \$g5, \$t5 .bend block1 .bbegin block2 ...</pre>	<pre>[R1] \$g1 [2] [R2] \$g2 [1] [4] [1] ld L[1] 4 [2] [2] add [3] [4] [3] mov [5] [6] [4] st S[2] 4 [5] addi 2 [w1] [6] teqz [7] [8] [7] b_t block3 [8] b_f block2 [w1] \$g5</pre>

- Read target format
- Predicated instructions
- LD/ST sequence numbers
- Target fanout with movs
- Block outputs fixed (3 in this example)

June 4, 2005 UT-Austin TRIPS Tutorial 15

- ### Key Features of TRIPS ISA
- Fixed instruction lengths - all instructions 32 bits
 - Read and write instructions
 - Contained in block header for managing DFG/register file communication
 - Target format (T0, T1)
 - Output destinations specified with 9-bit targets
 - Predication (PR)
 - Nearly every instruction has a predicate field, encoded for efficient dataflow predication
 - Load/store IDs (LSID)
 - Used to maintain sequential memory semantics despite EDGE ISA
 - Exit bits (EXIT)
 - Used to improve block exit control prediction
- June 4, 2005 UT-Austin TRIPS Tutorial 16



Target (Operand) Formats

- 00: No target (or write inst.)
- 01: Targets predicate field
- 10: Targets left operand
- 11: Targets right operand

- Names one of 128 target instructions
- Location of instruction is microarchitecture dependent

- Names one of 32 write instructions

01 5 bits

June 4, 2005 UT-Austin TRIPS Tutorial 19

Object Code Example

TASL (target format)

```

[R1] $g1 [2]
[R2] $g2 [1] [4]
[1] ld L[1] 4
[2] add [3] [4]
[3] mov [5] [6]
[4] st S[2] 4
[5] addi 2 [W1]
[6] teqz [7] [8]
[7] b_t block3
[8] b_f block2
[W1] $g5
        
```

Object Code

v	reg	T1	T0	
v	reg	T1	T0	
opcode	pr	LSID	imm	T0
opcode	pr	xop	T1	T0
opcode	pr	xop	T1	T0
opcode	pr	LSID	imm	0
opcode	pr	xop	imm	T0
opcode	pr	xop	T1	T0
opcode	pr	exit	offset	
opcode	pr	exit	offset	
v	reg			

June 4, 2005 UT-Austin TRIPS Tutorial 20

Object Code Example

TASL (target format)

```

[R1] $g1 [2]
[R2] $g2 [1] [4]
[1] ld L[1] 4
[2] add [3] [4]
[3] mov [5] [6]
[4] st S[2] 4
[5] addi 2 [W1]
[6] teqz [7] [8]
[7] b_t block3
[8] b_f block2
[W1] $g5
        
```

Object Code

1	00001	[2]	----	
1	00010	[1]	[4]	
ld	00	00001	4	[2]
add	00	---	[3]	[4]
mov	00	---	[5]	[6]
st	00	00010	4	---
addi	00	---	2	[W1]
teqz	00	---	[7]	[8]
b	11 (t)	E0	block3 - PC	
b	10 (f)	E1	block2 - PC	
1	00101			

Store mask: 00000000000000000000000000000000

June 4, 2005 UT-Austin TRIPS Tutorial 21

TRIPS Block Format

PC →

Header Chunk	128 Bytes
Instruction Chunk 0	128 Bytes
Instruction Chunk 1	128 Bytes
Instruction Chunk 2	128 Bytes
Instruction Chunk 3	128 Bytes

- Each block is formed from two to five 128-byte program "chunks"
- Blocks with fewer than five chunks are expanded to five chunks in the L1 I-cache
- The header chunk includes a block header (execution flags plus a store mask) and register read/write instructions
- Each instruction chunk holds 32 4-byte instructions (including NOPs)
- A maximally sized block contains 128 regular instructions, 32 read instructions, and 32 write instructions

June 4, 2005 UT-Austin TRIPS Tutorial 22

Example of Predicate Operation

- Test instructions output low-order bit of *I* or *O* (*T* or *F*)
- Predicated instructions either match the arriving predicate or not
 - An ADD with PR=10 is predicated on false, an arriving 0 predicate will match and enable the ADD
 - Predicated instructions must receive all operands and a matching predicate before firing
 - Instructions may receive multiple predicate operands but at most one matching predicate
 - Despite predicates, blocks must produce fixed # of outputs.

```

if (p==0)
  z = a * 2 + 3;
else
  z = b * 3 + 4;
        
```

June 4, 2005 UT-Austin TRIPS Tutorial 23

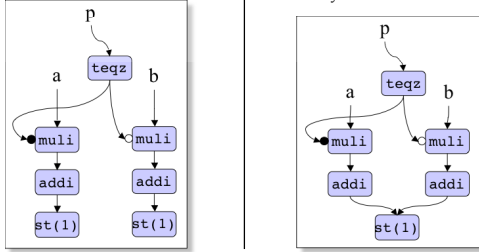
Predicate Optimization and Fanout Reduction

- Predicate dependence heads
 - Reduce instructions executed
 - Reduces dynamic energy by using less speculation
- Predicate dependence tails
 - Tolerate predicate latencies with speculatively executed insts.
 - Cannot speculate on block outputs

June 4, 2005 UT-Austin TRIPS Tutorial 24

Instruction (Store) Merging

- Old: two copies of stores down distinct predicate paths
- New: merge redundant insts.
 - Speculative copies unneeded
 - Only one data source will fire



June 4, 2005

UT-Austin TRIPS Tutorial

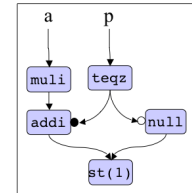
25

Nullified Outputs

- Each transmitted operand tagged with “null” and “exception” tokens.
 - Nullified operands arriving at insts. produce nullified outputs
 - Nulls used to indicate that a block won't produce a certain output (writes and stores)

Block with store down one conditional path:

```
...
if (p==0)
  z = a * 2 + 3;
...
```



June 4, 2005

UT-Austin TRIPS Tutorial

26

Exception Handling

- TRIPS exception types:
 - fetch, breakpoint, timeout, execute, store, system call, reset, interrupt
- TRIPS exception model demands
 - Must not raise exceptions for speculative instructions
- Key concept: propagate exception tokens within dataflow graph
 - Instructions with exception-set operands do not execute, but propagate exception token to their targets
 - Exceptions are detected at commit time if one or more block output (write, store, branch) has exception bit set
 - Exceptions serviced between blocks
 - “Block-precise” exception model

June 4, 2005

UT-Austin TRIPS Tutorial

27

Other Architecturally Supported Features

- Execution mode via Thread Control Register
 - Types of speculation (load, branch)
 - Number of blocks in flight
 - Breakpoints before/after blocks execute
- T-Morph via Processor Control Register
 - Can place processors into single or 4-threaded SMT modes
- Per-bank cache/scratchpad configurations
- TLBs managed by software
- On-chip network routing tables programmed by software
- 40-bit global system memory address space
 - Divided into SRF, cached memory, configuration quadrants

June 4, 2005

UT-Austin TRIPS Tutorial

28

TRIPS Microarchitecture and Implementation

ISCA-32 Tutorial

Steve Keckler



Department of Computer Sciences
The University of Texas at Austin

June 4, 2005

UT-Austin TRIPS Tutorial

29

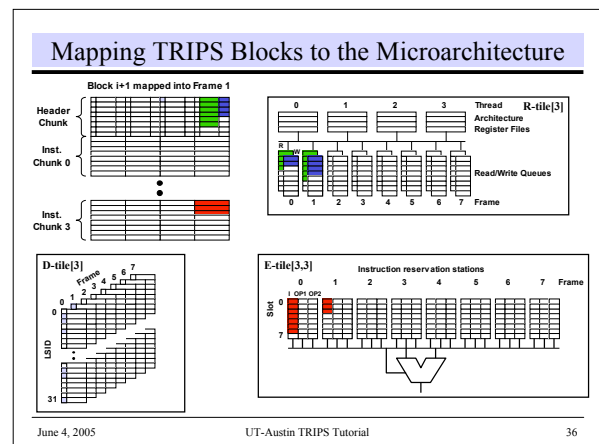
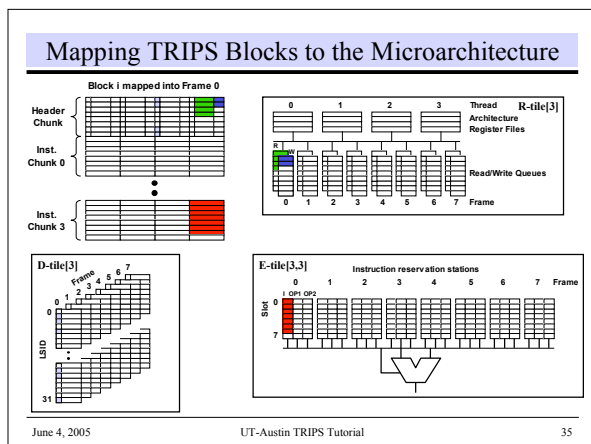
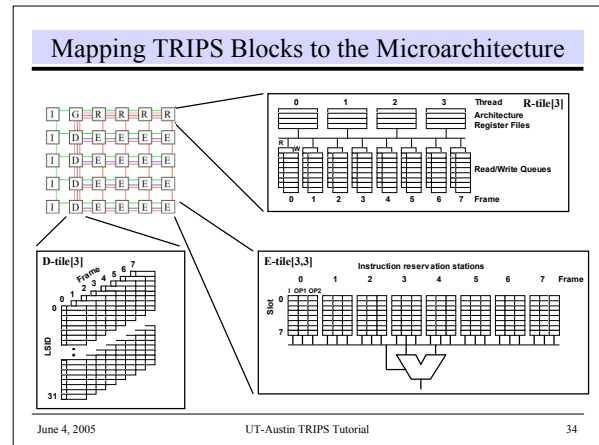
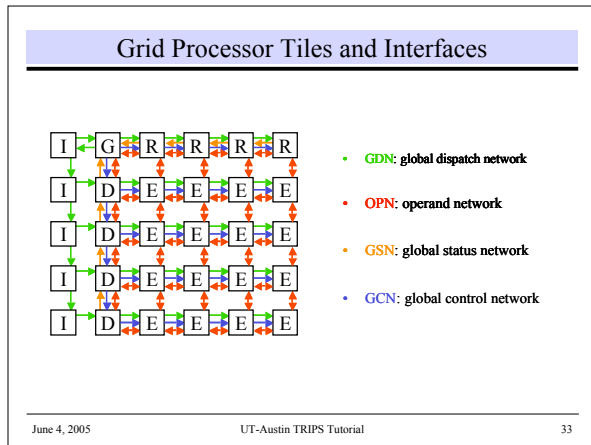
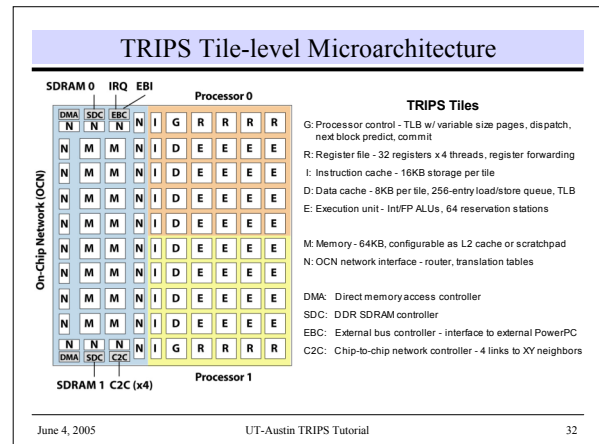
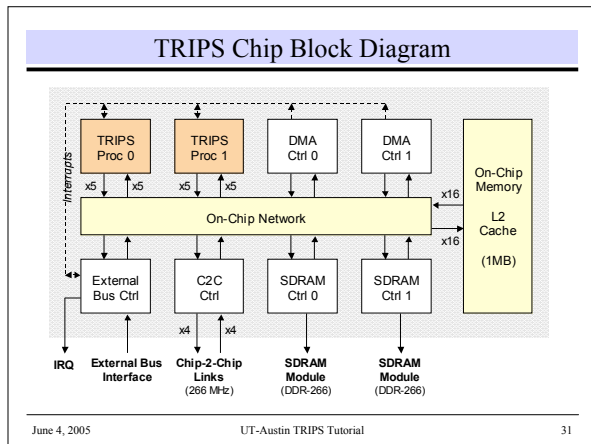
TRIPS Microarchitecture Principles

- Limit wire lengths
 - Architecture is partitioned and distributed
 - Microarchitecture composed of different types of tiles
 - No centralized resources
 - Local wires are short
 - Tiles are small (2-5 mm² per tile is typical)
 - No global wires
 - Networks connect only nearest neighbors
- Design for scalability
 - Design productivity by replicating tiles (design reuse)
 - Tiles interact through well-defined control and data networks
 - Networks are extensible, even late in the design cycle
 - Opportunity for asynchronous interfaces
 - Prototype: employs communication latencies of 35nm technology, even though prototype is implemented in 130nm

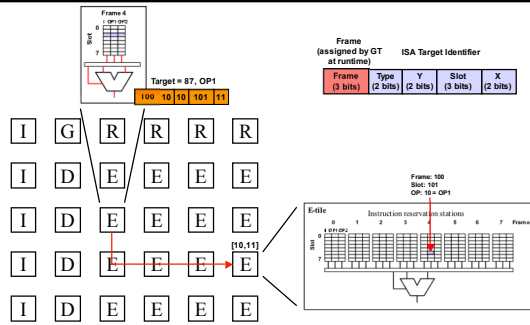
June 4, 2005

UT-Austin TRIPS Tutorial

30



Mapping Target Identifiers to Reservation Stations

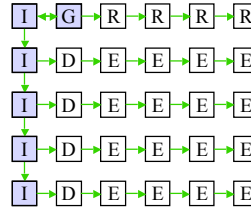


June 4, 2005

UT-Austin TRIPS Tutorial

37

Processor Core Tiles and Interfaces



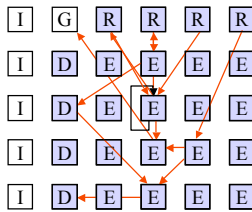
- Fetch
 - G-tile examines IS tags
 - Sends dispatch command to I-tiles
 - I-tiles fetch instructions, distribute to R-tiles, E-tiles

June 4, 2005

UT-Austin TRIPS Tutorial

38

Processor Core Tiles and Interfaces



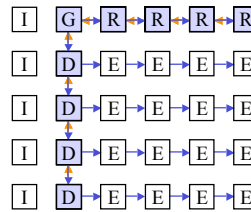
- Execute
 - R-tiles inject register values
 - E-tiles execute block instructions
 - compute, load/store
 - E-tiles delivers outputs
 - Results to R-tiles/D-tiles
 - Branch to G-tile

June 4, 2005

UT-Austin TRIPS Tutorial

39

Processor Core Tiles and Interfaces



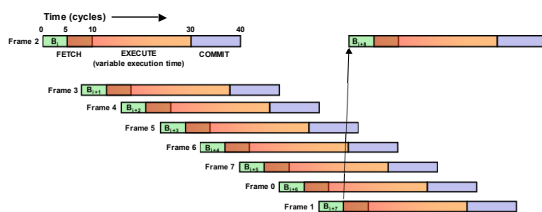
- Commit
 - R-tiles/D-tiles accumulate outputs
 - Send completion messages to G-tile
 - G-tile sends commit message
 - R-tiles/D-tiles respond with commit acknowledge
 - R-tiles commit state to register files
 - D-tiles commit state to data cache

June 4, 2005

UT-Austin TRIPS Tutorial

40

Block Execution Timeline



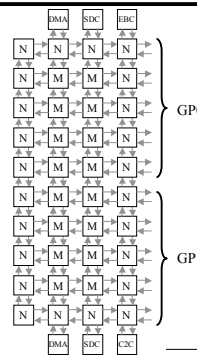
- Execute/commit overlapped across multiple blocks
- G-tile manages frames as a circular buffer
 - D-morph: 1 thread, 8 frames
 - T-morph: up to 4 threads, 2 frames each

June 4, 2005

UT-Austin TRIPS Tutorial

41

On-Chip Network



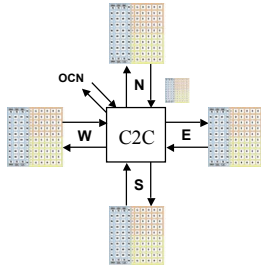
- Flexible/fast on-chip network fabric
 - Programmable translation tables
 - Configurable 1 MB memory system
 - 128-bit wide, 533 MHz
 - 6.8GB/sec per link
- M-tile (on-chip memory)
 - 64 KB SRAM capacity
 - Configurable tag array (on/off)
- N-tile with translation tables
 - System address \Rightarrow OCN address
- EBC - external bus controller
 - Connection to master control processor
- SDC - DDR1 SDRAM controller
 - 1.1 GB/sec per controller
- DMA controller
 - Linear, scatter-gather, chaining

June 4, 2005

UT-Austin TRIPS Tutorial

42

Chip-to-Chip (C2C) Network



- Glueless interchip network
 - Extension of OCN
 - 2D mesh router
 - 64-bits wide
 - 266MHz, 1.7GB/sec per link

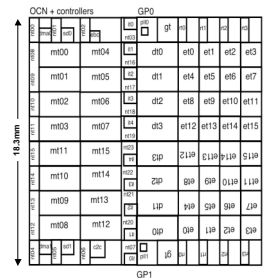
June 4, 2005

UT-Austin TRIPS Tutorial

43

Chip Implementation

- Chip Specs
 - 130 nm 7LM IBM ASIC process
 - 18.3mm x 18.3mm die
 - 853 signal I/Os
 - 568 for C2C
 - 216 for SDCs
 - ~70 misc I/Os
- Schedule
 - 5/99: Seed of TRIPS ideas
 - 12/01: Publish TRIPS concept architecture
 - 10/02: Publish NUCA concept architecture
 - 6/03: Start high-level prototype design
 - 10/03: Start RTL design
 - 2/05: RTL 90% complete
 - 7/05: Design 100% complete
 - 9/05: Tapeout
 - 12/05: Start system evaluation in lab



TRIPS Working Floorplan (to scale)

June 4, 2005

UT-Austin TRIPS Tutorial

44

Prototype Simplifications

- Architecture simplifications
 - No hardware cache coherence mechanisms
 - No floating-point divider
 - No native exception handling
 - Simplified exception reporting
 - Limited support for variable-sized blocks
- Microarchitecture simplifications
 - No I-cache prefetching or block predict for I-cache refill
 - No variable resource allocation (for smaller blocks)
 - One cycle per hop on every network
 - No clock-gating
 - Direct-mapped and replicated LSQ (non-associative, oversized)
 - No selective re-execution
 - No fast block refetch

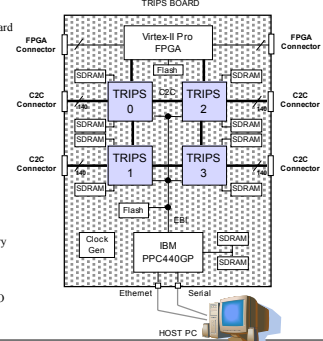
June 4, 2005

UT-Austin TRIPS Tutorial

45

TRIPS Prototype System Board

- TRIPS chip network
 - 4 TRIPS chips (8 processors) per board
 - Connected using a 2x2 chip-to-chip network
 - Each chip provides local and remote access to 2GB SDRAM
- PPC440GP
 - Configures and controls the TRIPS chips
 - Connects to a Host PC for user interface (via serial port or ethernet)
 - Runs an embedded Linux OS
 - Includes a variety of integrated peripherals
- SDRAM DIMMs
 - 8 slots for up to 8 GB TRIPS memory
 - 2 slots for up to 1 GB PPC memory
- FPGA - usage TBD but may include
 - Protocol translator for high-speed I/O devices
 - Programmable acceleration engine

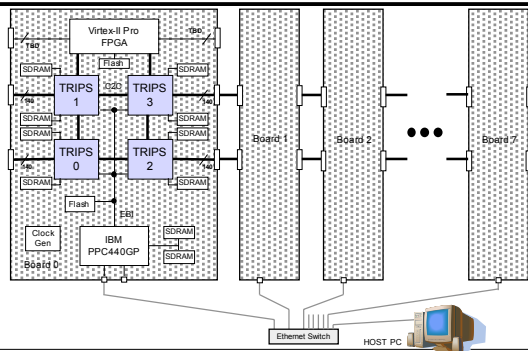


June 4, 2005

UT-Austin TRIPS Tutorial

46

Maximum Size TRIPS System



June 4, 2005

UT-Austin TRIPS Tutorial

47

TRIPS Prototype System Capabilities

- Clock speeds
 - 533MHz internal clock
 - 266MHz C2C clock
 - 133/266 SDRAM clock
- Single chip
 - 2 processors per chip each with
 - 64 KB L1-IS, 32KB L1-DS
 - 8.5 Gops or GFlops/processor
 - 17 Gops or GFlops/chip peak
 - Up to 8 threads
 - 1 MB on-chip memory
 - L2 cache or scratchpad
 - 2 GB SDRAM
 - 2.2 GB/sec DRAM bandwidth
- Full system (8 boards)
 - 32 chips, 64 processors
 - 1024 64-bit FPUs
 - 545 Gops/Gflops
 - Up to 256 threads
 - 64GB SDRAM
 - 70 GB/s aggregate DRAM bandwidth
 - 6.8 GB/sec C2C bisection bandwidth

June 4, 2005

UT-Austin TRIPS Tutorial

48

Code Examples

ISCA-32 Tutorial

Robert McDonald



Department of Computer Sciences
The University of Texas at Austin

June 4, 2005

UT-Austin TRIPS Tutorial

49

Two Examples

- Vector Add Example
 - A simple for loop
 - Basic unrolling
 - TRIPS Intermediate Language (TIL)
 - Block Dataflow Graph
 - Placed Instructions
 - TRIPS Assembly Language (TASL)
 - Sample Execution
- Linked List Example
 - A while loop with pointers
 - TRIPS predication
 - Predicated loop unrolling

June 4, 2005

UT-Austin TRIPS Tutorial

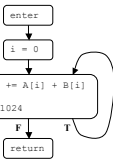
50

Vector Add – C Code

```

#define VSIZE 1024
void vadd(double* A, double* B,
double* C)
{
    int i;
    for (i = 0; i < VSIZE; i++)
        C[i] += (A[i] + B[i]);
}
    
```

- Consider this simple C routine
- The routine does an add and accumulate for fixed-size vectors
- An initial control flow graph is shown

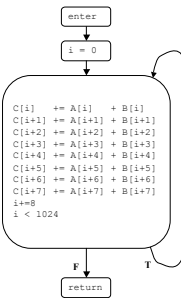


June 4, 2005

UT-Austin TRIPS Tutorial

51

Vector Add – Unrolled



- This loop wants to be unrolled
- Unrolling reduces the overhead per loop iteration
- It reduces the number of conditional branches that must be executed

June 4, 2005

UT-Austin TRIPS Tutorial

52

Vector Add – TIL Code

```

.begin vadd$1
read $r0, $r7
read $r1, $r7
read $r2, $r7
read $r3, $r7
ld $r4, ($r1, $r0)
ld $r5, ($r1, $r1)
ld $r6, ($r1, $r2)
fadd $r7, $r4, $r5
fadd $r8, $r6, $r7
ad ($r1), $r8, $r3
ld $r9, ($r1, $r3)
ld $r10, ($r1, $r4)
ld $r11, ($r1, $r5)
fadd $r12, $r9, $r10
fadd $r13, $r11, $r12
ad $r14, $r13, $r7
ld $r15, ($r1, $r6)
ld $r16, ($r1, $r7)
fadd $r17, $r15, $r16
fadd $r18, $r17, $r13
ad $r19, $r18, $r11
ld $r20, ($r1, $r7)
ld $r21, ($r1, $r8)
ld $r22, ($r1, $r9)
ld $r23, ($r1, $r10)
fadd $r24, $r22, $r21
fadd $r25, $r23, $r24
ad $r26, $r25, $r19
ld $r27, ($r1, $r8)
ld $r28, ($r1, $r9)
ld $r29, ($r1, $r10)
ld $r30, ($r1, $r11)
fadd $r31, $r29, $r28
fadd $r32, $r30, $r31
ad $r33, $r32, $r26
ld $r34, ($r1, $r9)
ld $r35, ($r1, $r10)
ld $r36, ($r1, $r11)
ld $r37, ($r1, $r12)
fadd $r38, $r36, $r35
fadd $r39, $r37, $r38
ad $r40, $r39, $r33
ld $r41, ($r1, $r10)
ld $r42, ($r1, $r11)
ld $r43, ($r1, $r12)
ld $r44, ($r1, $r13)
fadd $r45, $r43, $r42
fadd $r46, $r44, $r45
ad $r47, $r46, $r40
ad $r48, $r47, $r40
ad $r49, $r48, $r40
ad $r50, $r49, $r40
l1e $r54, $r40, $r52
br0, 0, $r54, vadd$2
write $r70, $r47
write $r71, $r47
write $r72, $r49
write $r73, $r49
.end
    
```

```

ld $r29, 40($r1) |i 20|
ld $r30, 40($r1) |i 21|
ld $r31, 40($r1) |i 22|
fadd $r32, $r30, $r31
fadd $r33, $r32, $r30
ad $r34, 40($r1), $r33 |i 23|
ld $r35, 40($r1) |i 24|
ld $r36, 40($r1) |i 25|
ld $r37, 40($r1) |i 26|
fadd $r38, $r35, $r36
fadd $r39, $r37, $r38
ad $r40, $r39, $r34
ad $r41, $r40, $r34
ad $r42, $r41, $r34
ad $r43, $r42, $r34
ad $r44, $r43, $r34
ad $r45, $r44, $r34
ad $r46, $r45, $r34
ad $r47, $r46, $r34
ad $r48, $r47, $r34
ad $r49, $r48, $r34
ad $r50, $r49, $r34
l1e $r54, $r40, $r52
br0, 0, $r54, vadd$2
write $r70, $r47
write $r71, $r47
write $r72, $r49
write $r73, $r49
.end
    
```

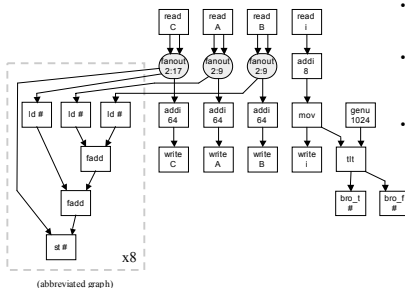
- The compiler produces TRIPS Intermediate Language (TIL) files
- Each file defines one or more program blocks
- Each block includes instructions, normally listed in program order
- Instructions are defined using a familiar syntax (name, target, sources)
- Read and write instructions access registers
- All other instructions deal with temporaries
- Notice the Load/Store IDs
- Notice the predicated branches

June 4, 2005

UT-Austin TRIPS Tutorial

53

Vector Add – Block Dataflow Graph



- Read and load instructions gather block inputs
- Write, store, and branch instructions produce block outputs
- Address fanout can present an interesting challenge

June 4, 2005

UT-Austin TRIPS Tutorial

54

Vector Add – Placed Instructions

ET0: ld, mov3, mov3, mov, mov, mov	ET1: mov3, ld, mov, mov, mov, mov	ET2: mov, mov3, ld, addi, addi, mov	ET3: mov, mov, mov, mov3, ld
ET4: ld, ld, ld, mov3, addi	ET5: ld, mov3, mov3, fadd, sd	ET6: ld, fadd, ld, ld, ld	ET7: ld, genu, addi
ET8: ld, ld, ld, mov3, fadd, sd	ET9: ld, ld, ld, ld, fadd	ET10: ld, fadd, fadd, sd	ET11: fadd, fadd, sd, th, bro_f
ET12: ld, fadd, fadd, sd	ET13: ld, fadd, fadd, sd	ET14: fadd, fadd, sd, ld	ET15: fadd, fadd, sd, bro_f

- The scheduler analyzes each block dataflow graph
- It inserts fanout instructions, as needed
- It places the instructions within the block (they don't have to be in program order)
- It produces assembly language files
- An instruction fires when its operands become available (regardless of position)

June 4, 2005 UT-Austin TRIPS Tutorial 55

Vector Add – TASL Code

```

.begin vaddsl
read $t0, $g70 ; key
read $t1, $g71 ; p
BB0
teql $p0, $t1, 0
ld $t3, ($t1)
teq_f <$p0> $p1, $t3, $t0
ld $t4, 16($t1) ; p->next
BB2
ld $t10, 8($t1) ; p->data
null_i <$p0> $t11
addi_t <$p1> $t11, $t10, 1
null_f <$p1> $t11
sd 8($t1), $t11 ; p->data
BB3
null_i <$p0, $p1> $t99
mov_f <$p1> $t99, $t4
write $g71, $t99 ; p
branches
bro_t <$p0, $p1> block2
bro_f <$p1> block1
.bend

```

```

97: ld $t0, 8($t0) ; key
98: read $t0, $g70 ; key
99: read $t1, $g71 ; p
A0: fadd $t0, $t0, $t1
A1: add $t0, $t0, 8
A2: fadd $t0, $t0, $t1
A3: ld $t0, 16($t0) ; p->next
A4: ld $t0, 8($t0) ; p->data
A5: fadd $t0, $t0, $t1
A6: add $t0, $t0, 8
A7: fadd $t0, $t0, $t1
A8: ld $t0, 16($t0) ; p->next
A9: ld $t0, 8($t0) ; p->data
AA: fadd $t0, $t0, $t1
AB: add $t0, $t0, 8
AC: fadd $t0, $t0, $t1
AD: ld $t0, 16($t0) ; p->next
AE: ld $t0, 8($t0) ; p->data
AF: fadd $t0, $t0, $t1
AG: add $t0, $t0, 8
AH: fadd $t0, $t0, $t1
AI: ld $t0, 16($t0) ; p->next
AJ: ld $t0, 8($t0) ; p->data
AK: fadd $t0, $t0, $t1
AL: add $t0, $t0, 8
AM: fadd $t0, $t0, $t1
AN: ld $t0, 16($t0) ; p->next
AO: ld $t0, 8($t0) ; p->data
AP: fadd $t0, $t0, $t1
AQ: add $t0, $t0, 8
AR: fadd $t0, $t0, $t1
AS: ld $t0, 16($t0) ; p->next
AT: ld $t0, 8($t0) ; p->data
AU: fadd $t0, $t0, $t1
AV: add $t0, $t0, 8
AW: fadd $t0, $t0, $t1
AX: ld $t0, 16($t0) ; p->next
AY: ld $t0, 8($t0) ; p->data
AZ: fadd $t0, $t0, $t1
BA: add $t0, $t0, 8
BB: fadd $t0, $t0, $t1
BC: ld $t0, 16($t0) ; p->next
BD: ld $t0, 8($t0) ; p->data
BE: fadd $t0, $t0, $t1
BF: add $t0, $t0, 8
BG: fadd $t0, $t0, $t1
BH: ld $t0, 16($t0) ; p->next
BI: ld $t0, 8($t0) ; p->data
BJ: fadd $t0, $t0, $t1
BK: add $t0, $t0, 8
BL: fadd $t0, $t0, $t1
BM: ld $t0, 16($t0) ; p->next
BN: ld $t0, 8($t0) ; p->data
BO: fadd $t0, $t0, $t1
BP: add $t0, $t0, 8
BQ: fadd $t0, $t0, $t1
BR: ld $t0, 16($t0) ; p->next
BS: ld $t0, 8($t0) ; p->data
BT: fadd $t0, $t0, $t1
BU: add $t0, $t0, 8
BV: fadd $t0, $t0, $t1
BV: write $g70
BV: write $g71
BV: write $g72
BV: write $g73
.bend

```

- TRIPS Assembly Language (TASL) files are fed to the assembler
- Instructions are described using a dataflow notation

June 4, 2005 UT-Austin TRIPS Tutorial 56

Vector Add – Block-Level Execution

- The TRIPS processor can execute up to 8 blocks concurrently
- This diagram shows block latency and throughput for an optimal vector add loop
- It achieves ~7.4 IPC, ~3.2 loads & stores per cycle

June 4, 2005 UT-Austin TRIPS Tutorial 57

Linked List – C Code

```

// type info
struct foo
{
    long tag;
    long value;
    struct foo * next;
};
struct foo * head;

void list_add( long key )
{
    struct foo * p = head;

    // search list and increment node
    while (p)
    {
        if (p->tag == key)
        {
            p->value++;
            break;
        }
        p = p->next;
    }
    // ...
}

```

- For a second example, let's examine something more complex
- We'll be focusing upon the code in red
- A linked list operation is performed using a while loop and search pointer
- The number of while loop iterations can vary

June 4, 2005 UT-Austin TRIPS Tutorial 58

Linked List – CFG

- The while loop's control flow graph is shown here
- It consists of four rather small basic blocks
- Its hard to get anywhere fast with all of those branches
- We'd like to use predication to form a larger program block (or hyperblock)

June 4, 2005 UT-Austin TRIPS Tutorial 59

Linked List – Hyperblock #1

```

.begin block1
read $t0, $g70 ; key
read $t1, $g71 ; p
BB0
teql $p0, $t1, 0
ld $t3, ($t1)
teq_f <$p0> $p1, $t3, $t0
ld $t4, 16($t1) ; p->next
BB2
ld $t10, 8($t1) ; p->data
null_i <$p0> $t11
addi_t <$p1> $t11, $t10, 1
null_f <$p1> $t11
sd 8($t1), $t11 ; p->data
BB3
null_i <$p0, $p1> $t99
mov_f <$p1> $t99, $t4
write $g71, $t99 ; p
branches
bro_t <$p0, $p1> block2
bro_f <$p1> block1
.bend

```

- Here is a TIL representation of the hyperblock
- Predicate p0 represents the test to determine whether p is null
- Predicate p1 represents the test to determine whether the tag matches the key
- The test instruction that produces p1 is itself predicated upon p0 being false
- That means that there are only three possible predicate outcomes:
 - p0 true, p1 unresolved
 - p0 false, p1 false
 - p0 false, p1 true
- This example demonstrates "predicate-oring", which allows multiple exclusive predicates to be OR'd at the target instruction
- This example demonstrates write and store nullification, which allows block outputs to be cancelled
- Loads are allowed to execute speculatively because exceptions will be predicated downstream

June 4, 2005 UT-Austin TRIPS Tutorial 60

Linked List – Predicated Loop Unrolling

- With predication, it's also possible (and helpful) to unroll this loop
- Many variations are possible
- This diagram shows just two iterations within the hyperblock
- There number of block exits and block outputs remains the same as before
- The compiler can use static heuristics or profile data to choose an unrolling factor
- Can sometimes use instruction merging to combine multiple statements into one (in yellow)

June 4, 2005 UT-Austin TRIPS Tutorial 61

Linked List – Hyperblock #2

```

.bbegin block1
read $t0, $g70      ; key
read $t1, $g71      ; p
treql $p0, $t1, 0   ; p->tag
ld $t3, ($t1)       ; p->tag
teq_f <$p0> $t1, $t3, $t0 ; p->tag
ld $t4, 16($t1)     ; p->next
treql_f <$p1> $p2, $t4, 0 ; p->next
ld $t5, ($t4)       ; p->next->tag
teq_f <$p2> $p3, $t5, $t0 ; p->next->tag
ld $t6, 16($t4)     ; p->next->next
; conditional p update
null_t <$p0,$p1> $t99, $t6
mov_t <$p2,$p3> $t99, $t4
write $g71, $t99    ; p
; conditional p->data update
ld_t <$p1> $t10, 8($t1)
ld_t <$p3> $t10, 8($t4)
null_t <$p0,$p3> $t11
addl_t <$p1,$p1> $t11, $t10, 1
null_f <$p3> $t11
sd 8($t11), $t11    ; p->data
; branches
bro_t <$p0,$p1,$p2,$p3> block2
bro_f <$p3> block1
.bend

```

- Here is some TIL for the unrolled while loop
- The changes from the first version are highlighted (in red)
- Notice the use of predicate-ORing, which allows the block to finish executing early in some cases
- There are now four predicates and five possible outcomes

p0	p1	p2	p3
T	-	-	-
F	T	-	-
F	F	T	-
F	F	F	T
F	F	F	F

June 4, 2005 UT-Austin TRIPS Tutorial 62

Conclusion

- The EDGE ISA allows instructions to be both *fetch*ed and *executed* out-of-order with minimal book-keeping
- It's possible to use loop unrolling and predication to form large blocks
- The TRIPS predication model allows control flow to be converted into an efficient dataflow representation
- Operand fanout is more explicit in an EDGE ISA and costs extra instructions

June 4, 2005 UT-Austin TRIPS Tutorial 63

Global Control

ISCA-32 Tutorial

Ramadass Nagarajan

Department of Computer Sciences
The University of Texas at Austin

June 4, 2005 UT-Austin TRIPS Tutorial 64

G-Tile Location and Connections

June 4, 2005 UT-Austin TRIPS Tutorial 65

Major G-Tile Responsibilities

- Orchestrates global control for the core
 - Maps blocks to execution resources
- Instruction fetch control
 - Determines I-cache hits/misses, ITLB hits/misses
 - Initiates I-cache refills
 - Initiates block fetches
- Next-block prediction
 - Generates speculative block addresses
- Block completion detection and commit initiation
- Flush detection and initiation
 - Branch mispredictions, Ld/St dependence violations, exceptions
- Exception reporting

June 4, 2005 UT-Austin TRIPS Tutorial 66

Networks Used by G-Tile

- **OPN** - Operand Network
 - Receive and send branch addresses from/to E-tiles
- **OCN** - On-Chip Network
 - Receive block header from adjacent I-tile
- **GDN** - Global Dispatch Network
 - Send inst. fetch commands to I-tiles and read/write instructions to R-tiles
- **GCN** - Global Control Network
 - Send commit and flush commands to D-tiles, R-tiles and E-tiles
- **GRN** - Global Refill Network
 - Send I-cache refill commands to I-tile slave cache banks
- **GSN** - Global Status Network
 - Receive refill, store, and register write completion status and commit acknowledgements from I-tiles, D-tiles, and R-tiles
- **ESN** - External Store Network
 - Receive external store pending and error status from D-tiles

June 4, 2005 UT-Austin TRIPS Tutorial 67

Global Dispatch Network (GDN)

The diagram shows a horizontal chain of tiles: I → G → R → R → R → R. Vertical arrows point from the first I tile to four D tiles below it. Each D tile is connected to an E tile to its right, forming a grid of D and E tiles.

Interface Ports

CMD: None/Fetch/Dispatch
 ADDR: Address of block
 SMASK: Store mask in block
 FLAGS: Block execution flags
 ID: Frame identifier
 INST: Instruction bits

- Delivers instruction fetch commands to I-tiles
- Dispatches instructions from I-Tiles to R-Tiles/E-Tiles

June 4, 2005 UT-Austin TRIPS Tutorial 68

Global Refill Network (GRN)

The diagram shows a horizontal chain of tiles: I ← G → R → R → R → R. Vertical arrows point from the first I tile to four D tiles below it. Each D tile is connected to an E tile to its right, forming a grid of D and E tiles.

Interface Ports

CMD: None/Refill
 RID: Refill buffer identifier
 ADDR: Address of block to refill

Delivers refill commands to I-Tiles

June 4, 2005 UT-Austin TRIPS Tutorial 69

Global Control Network (GCN)

The diagram shows a horizontal chain of tiles: I → G → R → R → R → R. Vertical arrows point from the G tile to four D tiles below it. Each D tile is connected to an E tile to its right, forming a grid of D and E tiles.

Interface Ports

CMD: None/Commit/Flush
 MASK: Mask of frames for commit/flush

Delivers commit and flush commands to R-Tiles, D-Tiles and E-Tiles

June 4, 2005 UT-Austin TRIPS Tutorial 70

Global Status Network (GSN)

The diagram shows a horizontal chain of tiles: I → G ← R ← R ← R → R. Vertical arrows point from the first I tile to four D tiles below it. Each D tile is connected to an E tile to its right, forming a grid of D and E tiles.

Interface Ports

CMD: None/Completed/Committed
 ID: Frame/Refill buffer identifier
 STATUS: Exception Status

I-tiles: Completion status for pending refills
 R-tiles: Completion status register writes, acknowledgement of register commits
 D-tiles: Completion status for stores, acknowledgement of store commits

June 4, 2005 UT-Austin TRIPS Tutorial 71

External Store Network (ESN)

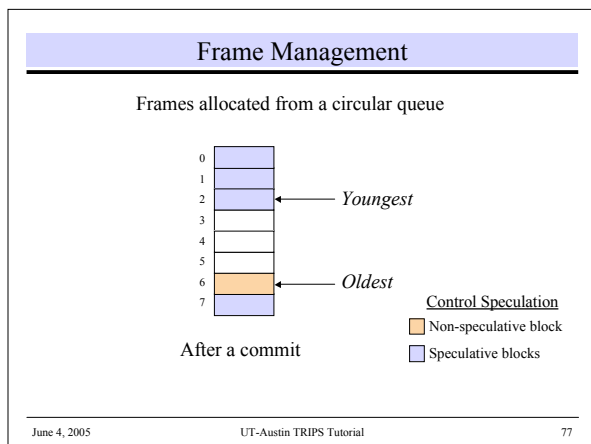
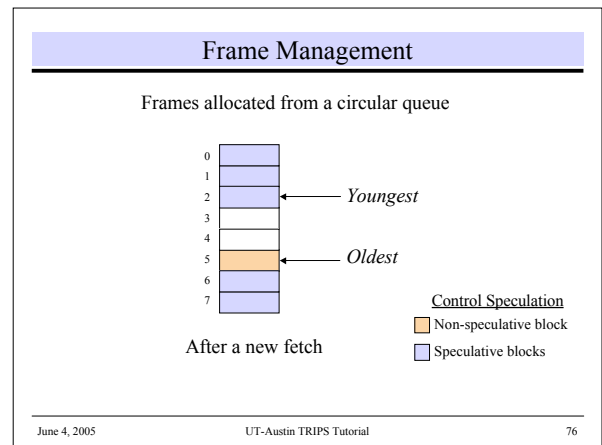
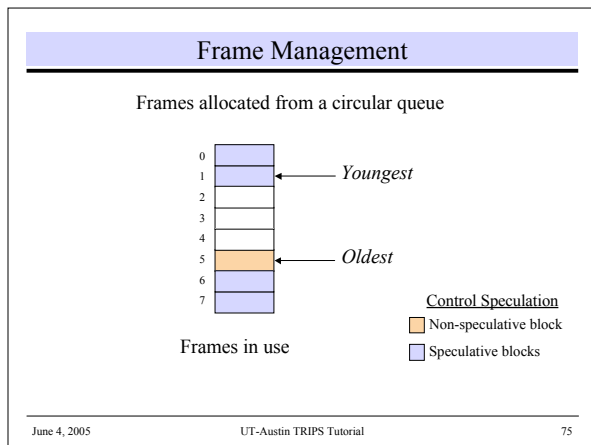
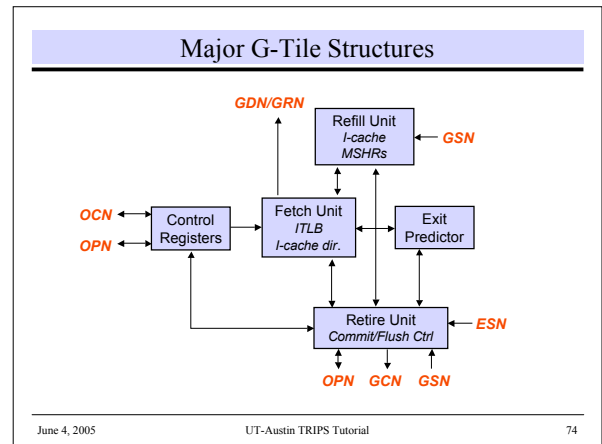
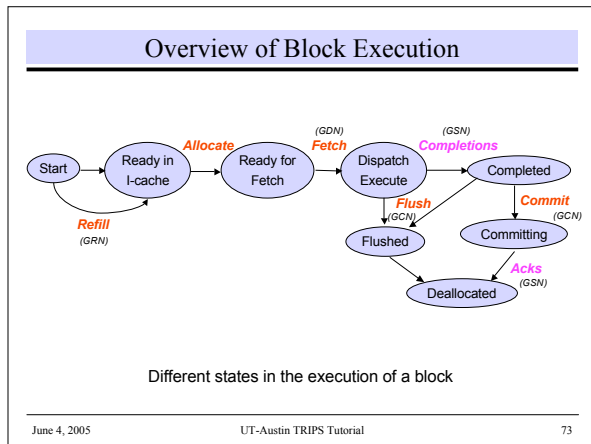
The diagram shows a horizontal chain of tiles: I → G → R → R → R → R. Vertical arrows point from the G tile to four D tiles below it. Each D tile is connected to an E tile to its right, forming a grid of D and E tiles.

Interface Ports

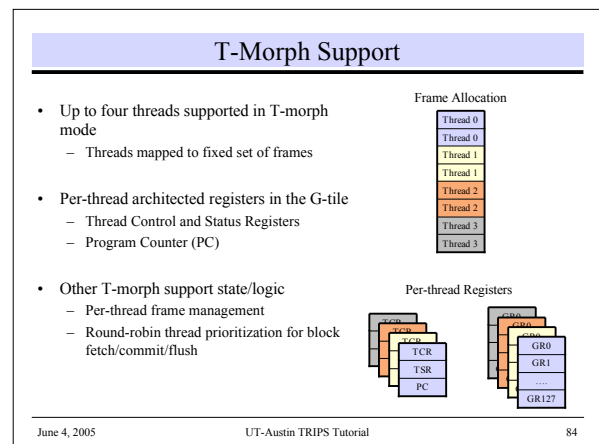
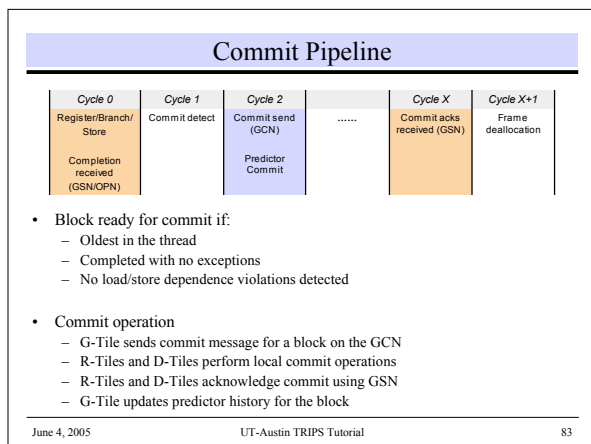
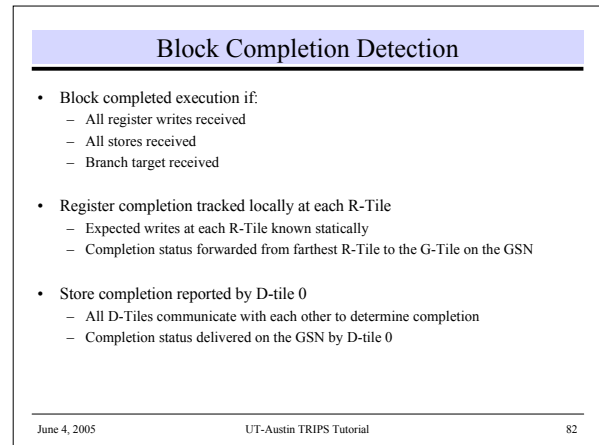
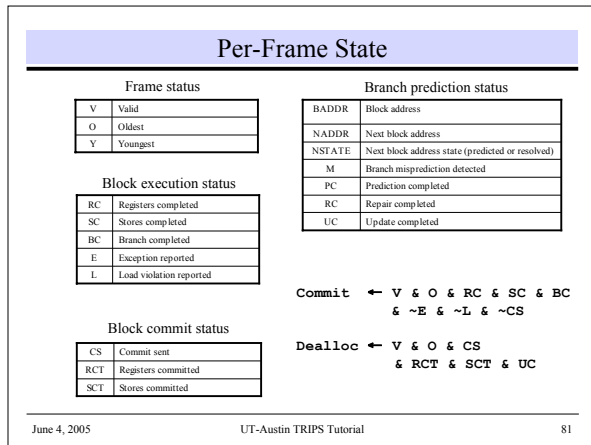
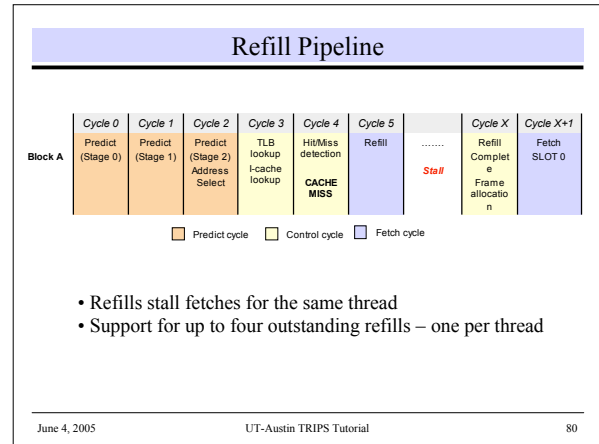
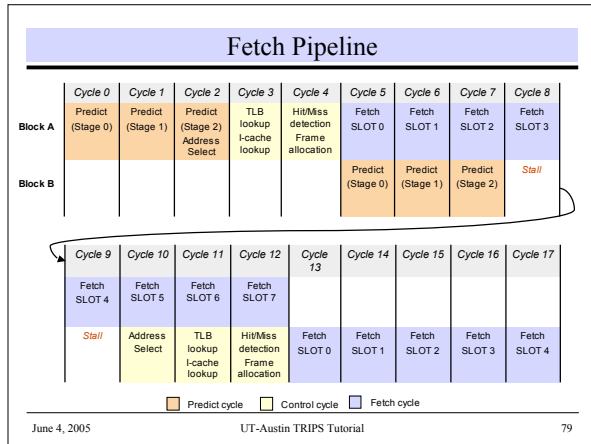
PENDING: Pending status of external store requests and spills in each thread
 ERROR: Errors encountered for external store requests and spills in each thread

Delivers external store pending and error status from D-tiles

June 4, 2005 UT-Austin TRIPS Tutorial 72



- ### Fetch Unit
- Instruction TLB
 - 16 entries
 - 64KB - 1TB supported page size
 - I-cache directory
 - 128 entries
 - Each entry corresponds to one cached block
 - Entry Fields
 - Physical tag for the block
 - Store mask for the block
 - Execution flags for the block
 - 2-way set associative
 - LRU replacement
 - Virtually indexed, physically tagged
- June 4, 2005 UT-Austin TRIPS Tutorial 78



Exception Reporting

- Exceptions reported to the G-tile
 - GSN: refill exceptions, execute exceptions delivered to stores, register outputs
 - OPN: execute exceptions delivered to branch targets
 - ESN: errors in external stores
- G-tile halts processor whenever exception occurs
 - Speculative blocks in all threads flushed
 - Oldest blocks in all threads completed and committed
 - All outstanding refills, stores and spills completed
 - Signals EBC to cause external interrupt
- Exception type reported in PSR and TSRs
 - Multiple exceptions may be reported

Breakpoint
Timeout
System Call
Reset
Fetch
Interrupt

Execute
Store
Fetch

Exceptions detected at G-tile (left), Exceptions detected at other tiles (right)

June 4, 2005 UT-Austin TRIPS Tutorial 85

Control Flow Prediction

ISCA-32 Tutorial

Nitya Ranganathan

Department of Computer Sciences
The University of Texas at Austin

June 4, 2005 UT-Austin TRIPS Tutorial 86

Predictor Responsibilities

- Predict next block address for fetch
 - Predict block exit and exit branch type (branches not seen by predictor)
 - Predict target of exit based on predicted branch type
 - Predictions amortized over each large block
 - Maximum necessary prediction rate is one every 8 cycles
- Speculative update for most-recent histories
 - Keep safe copies around to recover from mispredictions
- Repair predictor state upon a misprediction
- Update the predictor upon block commit
 - Update exits, targets and branch types with retired block state

June 4, 2005 UT-Austin TRIPS Tutorial 87

Example Hyperblock from Linked List Code

- Each hyperblock may have several predicted exit branches
 - Exit e0 → block H2
 - Exit e1 → block H1
- Only one branch will fire
- Predictor uses exit IDs to make prediction
 - 8 exit IDs supported
 - e0 → "000"
 - e1 → "001"
- Predictor predicts the ID of the exit that will fire
 - implicitly predict several predicates in a path
- Exit history formed by concatenating sequence of exit IDs

June 4, 2005 UT-Austin TRIPS Tutorial 88

High-level Predictor Design

- Index exit predictor with current block address
- Use exit history to predict 1 of 8 exits
- Predict branch type (call/return/branch) for chosen exit
- Index multiple target predictors with block address and predicted exit
- Choose target address based on branch type

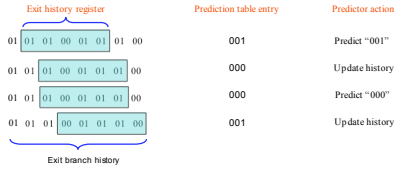
June 4, 2005 UT-Austin TRIPS Tutorial 89

Predictor Components

June 4, 2005 UT-Austin TRIPS Tutorial 90

Prediction using Exit History (linked list example)

- Dynamic block sequence H1 H1 H1 H2 H1 H1 H1 H2
- Local exit history for block H1 e1 e1 e1 e0 e1 e1 e1 e0
- Local exit history encoding (2 bits) 01 01 01 00 01 01 01 00
 - Use only 2 bits from each 3-bit exit to permit longer histories
- History table entry holds 5 exit IDs
- Prediction table contains exit IDs and hysteresis bits

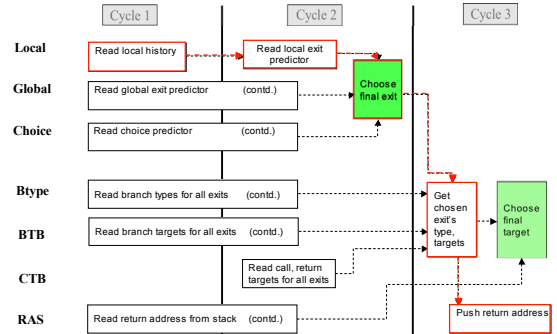


June 4, 2005

UT-Austin TRIPS Tutorial

91

Prediction Timing



June 4, 2005

UT-Austin TRIPS Tutorial

92

More Predictor Features

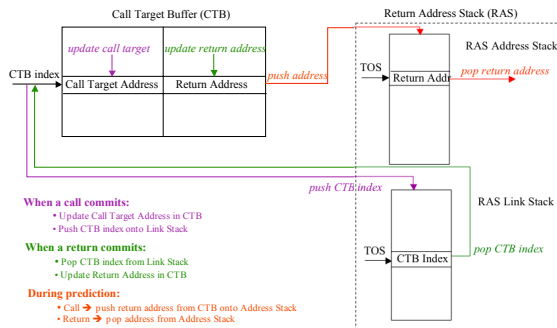
- Speculative state buffering
 - Latest global histories in global and choice history registers
 - Old snapshots of histories stored in history file for repair
 - Latest local history maintained in future file
 - Local exit history table holds committed block history
 - Return address stack state buffering
 - Save top of stack pointer and value before every prediction
 - Repair stack on address and branch type mispredictions
- Call-return
 - Call/return instructions enforce proper control flow
 - Predictor learns to predict return addresses using call-return link stack
 - Update stores return address in link stack corresponding to call stored in CTB

June 4, 2005

UT-Austin TRIPS Tutorial

93

Learning Call-Return Relationships

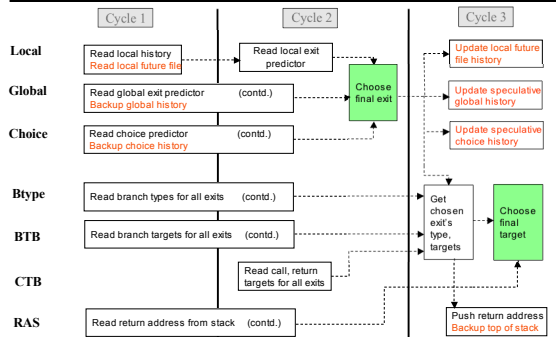


June 4, 2005

UT-Austin TRIPS Tutorial

94

Prediction and Speculative Update Timing



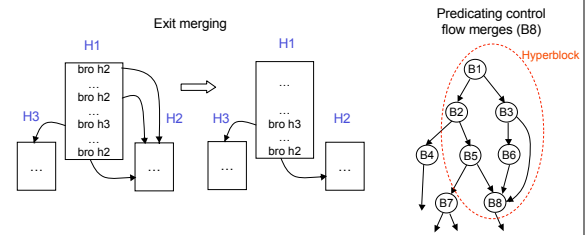
June 4, 2005

UT-Austin TRIPS Tutorial

95

Optimizations for Better Prediction

- Exit prediction more difficult than branch prediction
 - 1 of 8 vs. 1 of 2
- Fewer exits in block \rightarrow better predictability



June 4, 2005

UT-Austin TRIPS Tutorial

96

Instruction Fetch

ISCA-32 Tutorial

Haiming Liu

Department of Computer Sciences
 The University of Texas at Austin

June 4, 2005
UT-Austin TRIPS Tutorial
97

I-Tile Location and Connections

June 4, 2005 UT-Austin TRIPS Tutorial 98

I-Tile Major Responsibilities

- **Instruction Fetch**
 - Cache instructions for one row of R-Tiles/E-Tiles
 - Dispatch instructions to R-Tiles/E-Tiles through GDN
- **Instruction Refill**
 - Receive refill commands from G-Tile on GRN
 - Submit memory read requests to the N-Tile on OCN
 - Keep track of completion status of each ongoing refill
 - Report completion status to G-Tile on the GSN when refill completes
- **OCN Access Control**
 - Share N-Tile connection between I-Tile and D-Tile/G-Tile
 - Forward OCN transactions between GT/NT or DT/NT

June 4, 2005
UT-Austin TRIPS Tutorial
99

Major I-Tile Structures

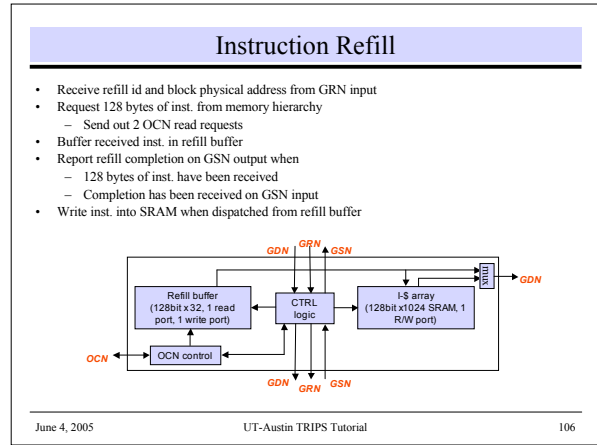
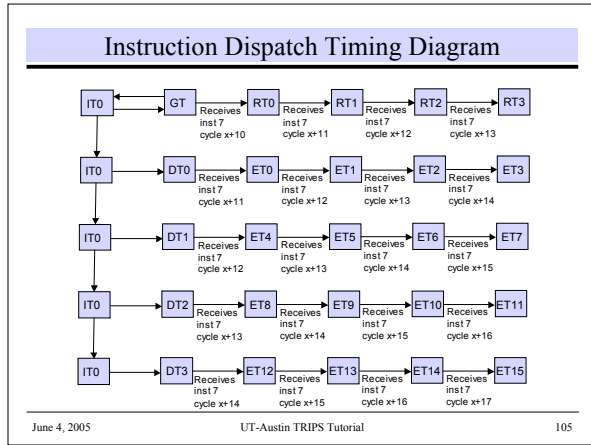
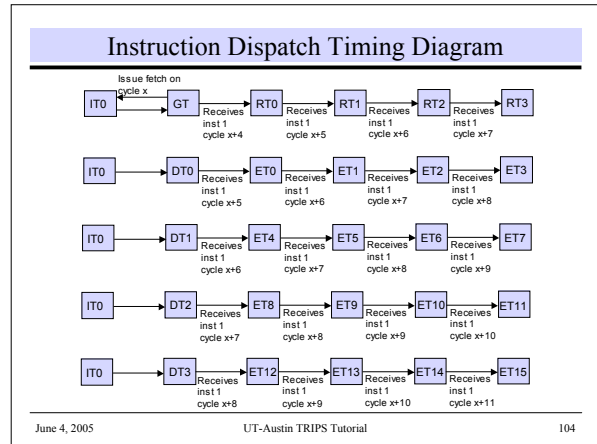
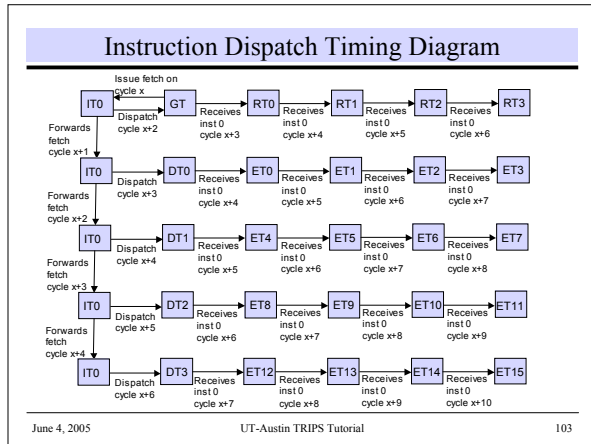
June 4, 2005 UT-Austin TRIPS Tutorial 100

Instruction Layout in a Maximal Block

June 4, 2005 UT-Austin TRIPS Tutorial 101

Instruction Fetch Pipeline

June 4, 2005 UT-Austin TRIPS Tutorial 102

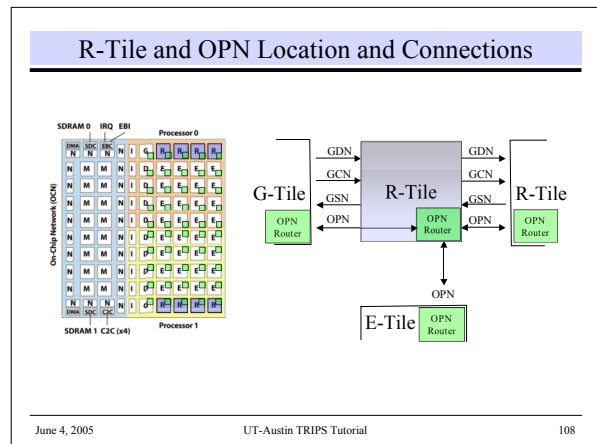


Register Accesses and Operand Routing

ISCA-32 Tutorial
Karu Sankaralingam

Department of Computer Sciences
The University of Texas at Austin

June 4, 2005 UT-Austin TRIPS Tutorial 107



Major R-Tile Responsibilities

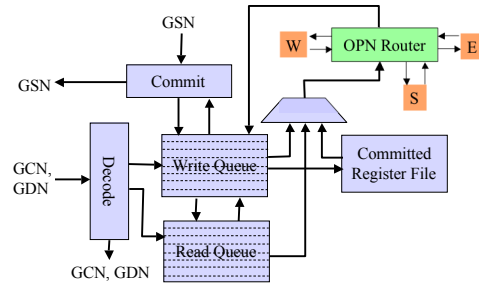
- Maintain architecture state, commit register writes
 - 512 registers total
 - 128 registers per thread, interleaved across 4 tiles
 - 32 registers/thread * 4 threads = 128 registers per tile
- Process read/write instructions for each block
 - 64 entry Read Queue (RQ), 8 blocks, 8 reads in each
 - 64 entry Write Queue (WQ), 8 blocks, 8 writes in each
- Forward inter-block register values
- Detect per-block and per-tile register write completion
- Detect exceptions that flow to register writes

June 4, 2005

UT-Austin TRIPS Tutorial

109

Major Structures in the R-tile

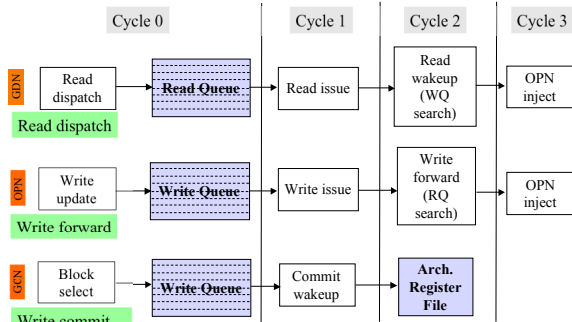


June 4, 2005

UT-Austin TRIPS Tutorial

110

Pipeline diagrams: Register Read and Write

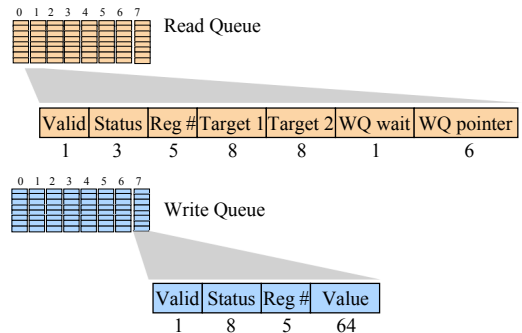


June 4, 2005

UT-Austin TRIPS Tutorial

111

Register Forwarding: RQ and WQ contents

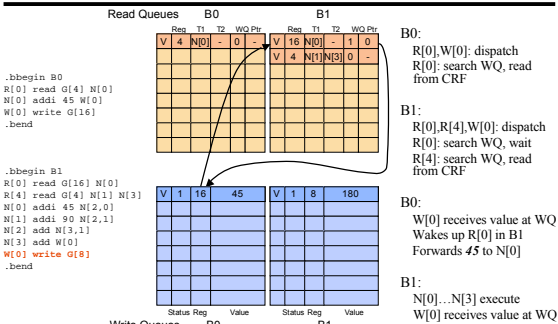


June 4, 2005

UT-Austin TRIPS Tutorial

112

Register Forwarding Example

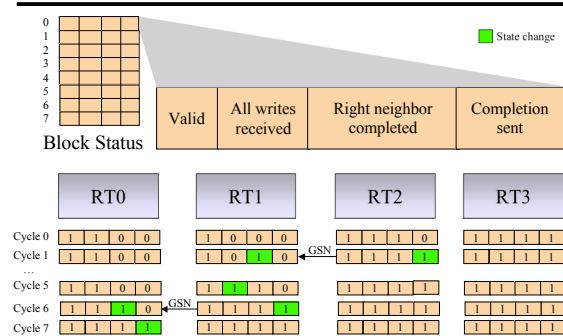


June 4, 2005

UT-Austin TRIPS Tutorial

113

Block completion: Commit unit

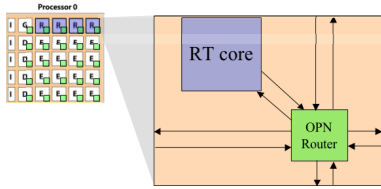


June 4, 2005

UT-Austin TRIPS Tutorial

114

Major OPN Responsibilities and Attributes



- Wormhole-routed network connecting 25 tiles
- 4 entry FIFO flit buffers in each direction
- Fixed packet length; 2 flits
- On-off flow control: 1-bit hold signal
- Dimension order routing (Y-X)

June 4, 2005

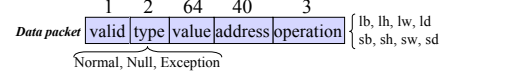
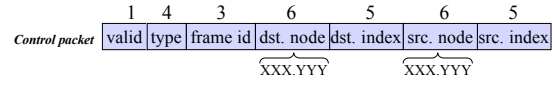
UT-Austin TRIPS Tutorial

115

OPN Packet Format

- One control packet followed by one data packet in next cycle
- Dedicated wires for control/data
- Control packet
 - Contains routing information
 - Enables early wakeup and fast bypass in ET

Control Packet Types
 0 - Generic transfer or reply
 1 - Load
 2 - Store
 4 - PC read
 5 - PC write
 14 - Special register read
 15 - Special register write

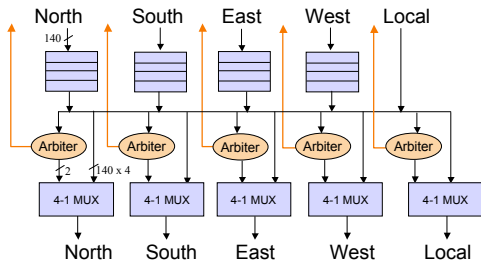


June 4, 2005

UT-Austin TRIPS Tutorial

116

OPN Router Schematic



June 4, 2005

UT-Austin TRIPS Tutorial

117

Operand Network Properties

- Directly integrated into processor pipeline
 - Hold signal stalls tile pipelines (back pressure flow control)
 - FIFOs flushed with processor flush wave
- Deadlock avoidance
 - On-off flow control
 - Dimension order routing
 - Guaranteed buffer location for every packet
- Tailored for operands
 - [Taylor et al. 2003, Sankaralingam et al. 2003]

June 4, 2005

UT-Austin TRIPS Tutorial

118

Issue Logic and Execution

ISCA-32 Tutorial

Premkishore Shivakumar



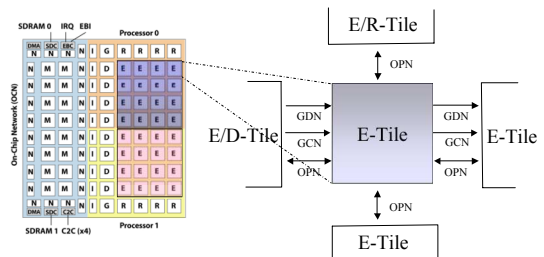
Department of Computer Sciences
The University of Texas at Austin

June 4, 2005

UT-Austin TRIPS Tutorial

119

E-Tile Location and Connections



June 4, 2005

UT-Austin TRIPS Tutorial

120

Networks Used by E-Tile

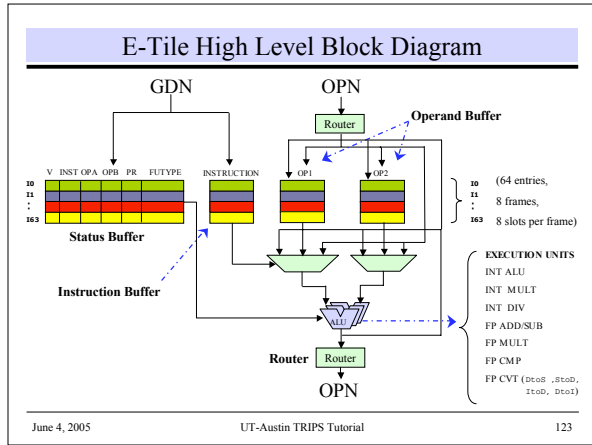
- **OPN** - Operand Network
 - Transmit register values from/to R-tiles
 - Transmit operands from/to other E-tiles
 - Transmit load/store packets from/to D-tiles
- **GDN** - Global Dispatch Network
 - Receive instructions from I-tile, and the dispatch commands from G-tile
 - Through a D-tile or a neighboring E-tile
- **GCN** - Global Control Network
 - Receive commit, flush commands originating at G-tile
 - From a D-tile or a neighboring E-tile

June 4, 2005 UT-Austin TRIPS Tutorial 121

Major E-Tile Responsibilities

- Buffer Instructions and Operands
 - Local reservation stations for instruction and operand management
- **Instruction Wakeup and Select**
- **Operand Bypass**
 - Local operand bypass to enable back-to-back local dependent instruction execution
 - Remote operand bypass with one extra cycle of OPN forwarding latency per hop
- **Instruction Target Delivery**
 - Instructions with multiple local and/or remote targets
 - Successive local or remote targets in consecutive cycles
 - Concurrently deliver a remote and a local target
- **Predication Support**
 - Predicate-based conditional execution
 - Hardware predicate OR-ing
- **Exception and Null**
 - Exception generation: divide by zero exception, NULL instruction
 - Exception and null tokens propagated in dataflow manner to register writes, stores
- **Flush and Commit**
 - Clear block state (pipeline registers, status buffer etc.)

June 4, 2005 UT-Austin TRIPS Tutorial 122



Functional Unit Latencies in the E-Tile

Functional Unit	Latency	Implementation
Integer ALU	1	Synopsys IP
Integer Multiplier	3 (Pipelined)	Synopsys IP
Integer Divider	24 (Iterative)	Synopsys IP
FP Add / Sub	4 (Pipelined)	Synopsys IP
FP Multiplier	4 (Pipelined)	Synopsys IP
FP Comparator	2 (Pipelined)	Synopsys IP
FP DtoI Convert	2 (Pipelined)	Synopsys IP
FP ItoD Convert	3 (Pipelined)	Synopsys IP
FP StoD Convert	2 (Pipelined)	UT TRIPS IP
FP DtoS Convert	3 (Pipelined)	UT TRIPS IP

June 4, 2005 UT-Austin TRIPS Tutorial 124

Issue Prioritization Among Ready Instructions

Reservation stations at one E-Tile

The diagram shows reservation stations B0 through B7. B0 is non-spec, while B1 through B7 are spec. The instructions are shown in a queue, with B0 being the oldest and B7 being the newest. The reservation stations are shown as a stack of colored blocks.

- Instructions can execute out-of-order
- Issue prioritization among ready instructions
- Scheduler assigns slots for block instructions
- Slot based instruction priority within a block

June 4, 2005 UT-Austin TRIPS Tutorial 125

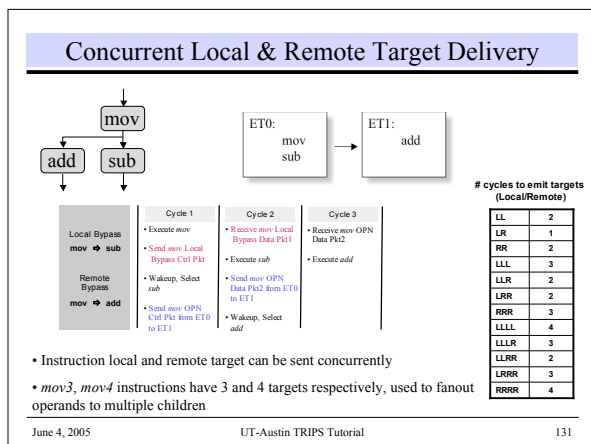
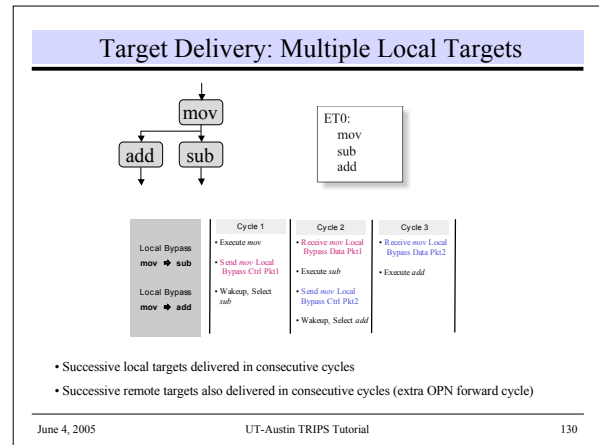
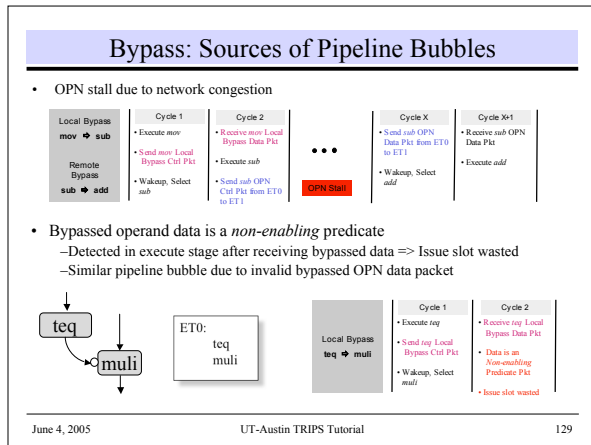
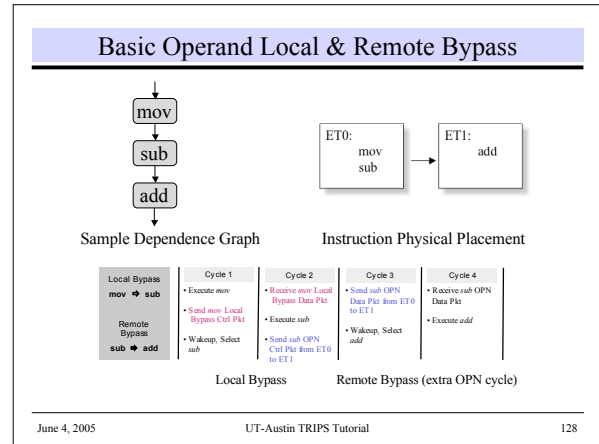
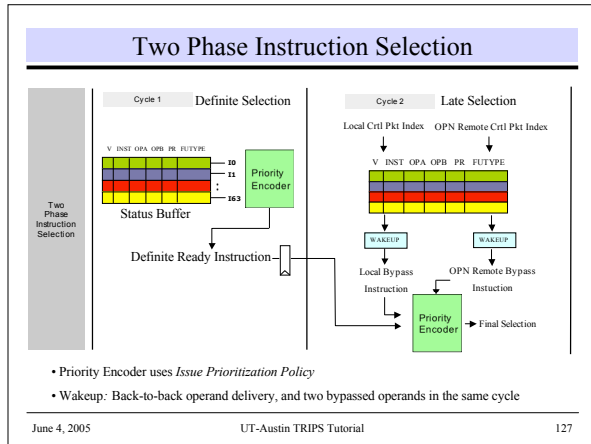
Issue Prioritization Among Ready Instructions

Reservation stations at one E-Tile

The diagram illustrates three prioritization rules: Older Block First (B1 before B2), Smaller Instruction Slot First (B3.10 before B3.12), and Three-level Select Priority (Round robin priority across threads, Older block first priority in a thread, Smaller instruction slot first within a block).

- Round robin priority across threads
- Older block first priority in a thread
- Smaller instruction slot first within a block

June 4, 2005 UT-Austin TRIPS Tutorial 126



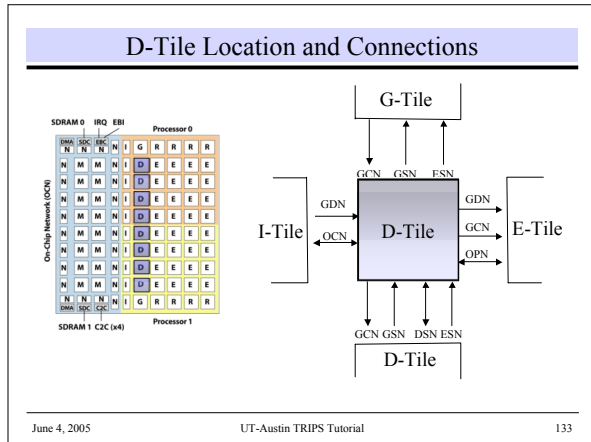
Primary Memory System

ISCA-32 Tutorial

Simha Sethumadhavan

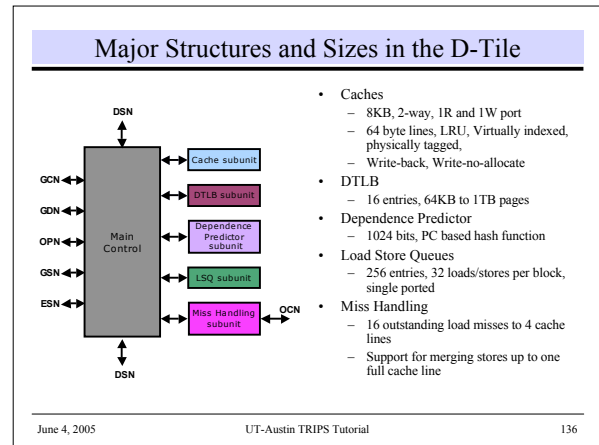
Department of Computer Sciences
The University of Texas at Austin

June 4, 2005 UT-Austin TRIPS Tutorial 132



- ### Major D-Tile Responsibilities
- Provide D-cache access for arriving loads and stores
 - Perform address translation with DTLB
 - Handle cache misses with MSHRs
 - Perform dynamic memory disambiguation with load/store queues
 - Perform dependence prediction for aggressive load/store issue
 - Detect per-block store completion
 - Write stores to caches/memory upon commit
 - Store merging on L1 cache misses
- June 4, 2005 UT-Austin TRIPS Tutorial 134

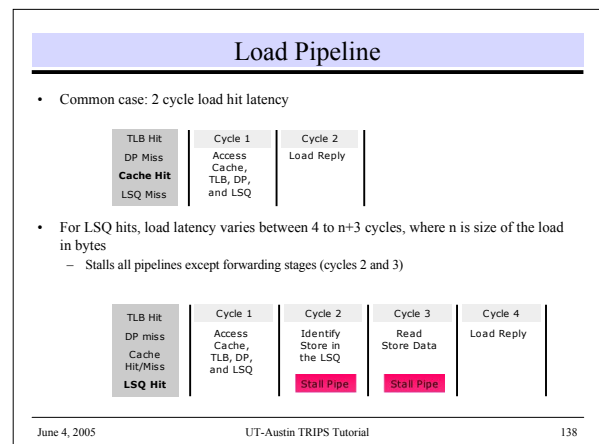
- ### Primary Memory Latencies
- Data cache is interleaved on a cache line basis (64 bytes)
 - Instruction placement affects load latency
 - Best case load-to-use latency: 5 cycles
 - Worst case load-to-use latency: 17 cycles
 - Loads favored in E-tile column adjacent to the D-tiles
 - Deciding row placement is difficult
 - Memory access latency: Three components
 - From E-Tile (load) to D-Tile (⊕)
 - D-Tile access (this talk) (⊙)
 - From D-Tile back to E-Tile (load target) (⊙)
-
- June 4, 2005 UT-Austin TRIPS Tutorial 135



Load Execution Scenarios

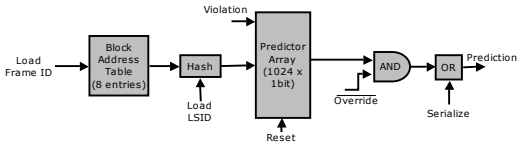
TLB	Dep. Pr	LSQ	Cache	Response
Miss	-	-	-	Report TLB Exception
Hit	Wait (Hit)	-	-	Defer load until all prior stores are received, Non deterministic latency
Hit	Miss	Miss	Hit	Forward data from L1 Cache
Hit	Miss	Miss	Miss	Forward data from L2 Cache (or memory) Issue cache fill request (if cacheable)
Hit	Miss	Hit	Hit	Forward data from LSQ
Hit	Miss	Hit	Miss	Forward data from LSQ Issue cache fill request (if cacheable)

June 4, 2005 UT-Austin TRIPS Tutorial 137



Dependence Prediction

- Prediction at memory side (not at issue)
 - New invention because of distributed execution
- Properties
 - PC-based, updated on violation
 - Only loads access the predictor, loads predicted dependent are *deferred*
 - *Deferred* loads are woken up when *all* previous stores have arrived
 - Reset periodically (flash clear) based on number of blocks committed
 - Dependence speculation is configurable: Override, Serial, Regular

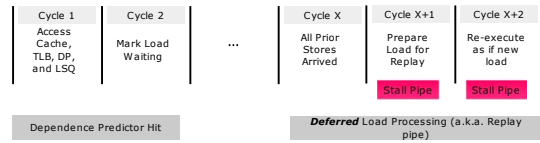


June 4, 2005

UT-Austin TRIPS Tutorial

139

Deferred Load Pipeline



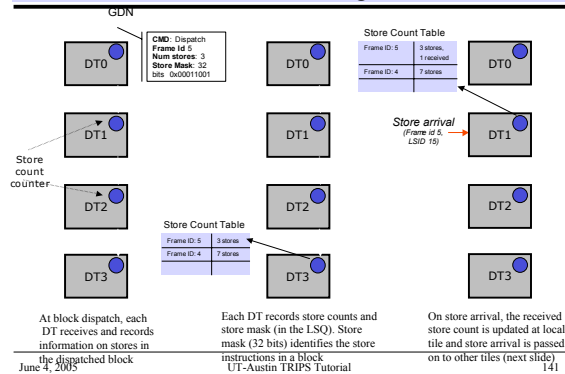
- Deferred loads are woken up when all prior stores have arrived at D-tiles
 - Loads between two consecutive stores can be woken up out-of-order
- Deferred loads are re-injected into the main pipeline, as if they were new load
 - During this phase, loads get the value from dependent stores, if any
- Pipeline stall
 - When deferred load in cycle X+2 cannot be re-injected into the main pipeline
 - Load cannot be prepared in cycle X+1 because of resource conflicts in the LSQ

June 4, 2005

UT-Austin TRIPS Tutorial

140

Store Counting



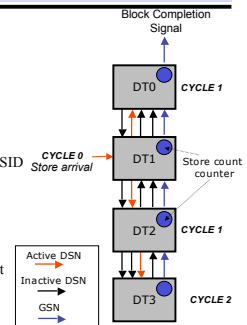
June 4, 2005

UT-Austin TRIPS Tutorial

141

Block Store Completion Detection

- Need to share store arrival information between D-tiles
 - Block completion
 - Waking up deferred loads
- Data Status Network (DSN) is used to communicate store arrival information
 - Multi-hop broadcast network
 - Each link carries store FrameID and store LSID
- DSN Operation
 - Store arrival initiates transfer on DSN
 - All tiles know about the store arrival after 3 cycles
- GSN Operation
 - Completion can be detected at any D-tile but special processing is done in D-tile0 before sending it to G-tile
 - Messages sent on GSN

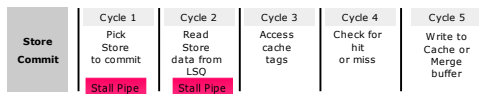


June 4, 2005

UT-Austin TRIPS Tutorial

142

Store Commit Pipeline



- Stores are committed in LSID and block order at each D-tile
 - Written into L1 cache or bypassed on a cache miss
 - One store committed per D-tile per cycle
 - Up to four stores total can be committed every cycle, possibly out-of-order
 - T-morph: Store commits from different threads are not interleaved
- On a cache miss, stores are inserted into the merge buffer
- Commit pipe stalls when cache ports are busy
 - Fills from missed loads take up cache write ports

June 4, 2005

UT-Austin TRIPS Tutorial

143

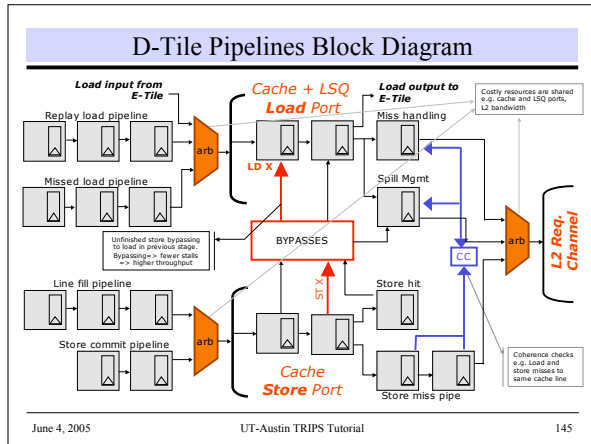
Load and Store Miss Handling

- Load Miss Operation
 - Cycle 1: Miss determination
 - Cycle 2: MSHR allocation and merging
 - Cycle 3: Requests sent out on OCN, if available
- Load Miss Return
 - Cycle 1: Wake up all loads waiting on incoming data
 - Cycle 2: Select one load; data for the load miss starts arriving
 - Cycle 3: Prepare load and re-inject into main pipeline, if unblocked
- Store Merging
 - Attempts to merge consecutive writes to same line
 - Snoops on all incoming load misses for coherency
- Resources are shared across threads

June 4, 2005

UT-Austin TRIPS Tutorial

144

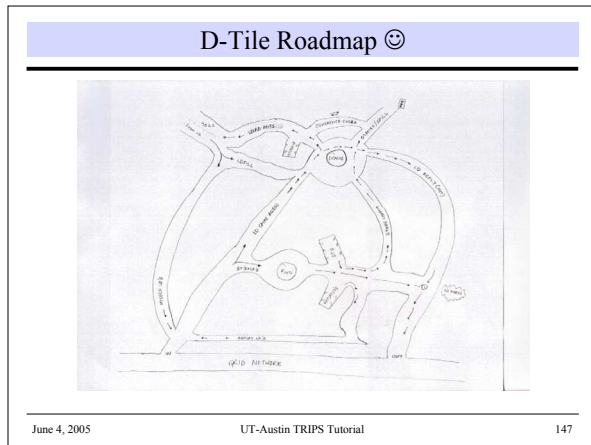


Memory System Design Challenges

- Optimizing and reducing the size of the LSQ
 - Prototype LSQ size exceeds data cache array size

- Further optimization of memory-side dependence prediction
 - Prior state of the art is done at instruction dispatch/issue time
- Scaling and decentralizing the Data Status Network (DSN)

June 4, 2005 UT-Austin TRIPS Tutorial 146



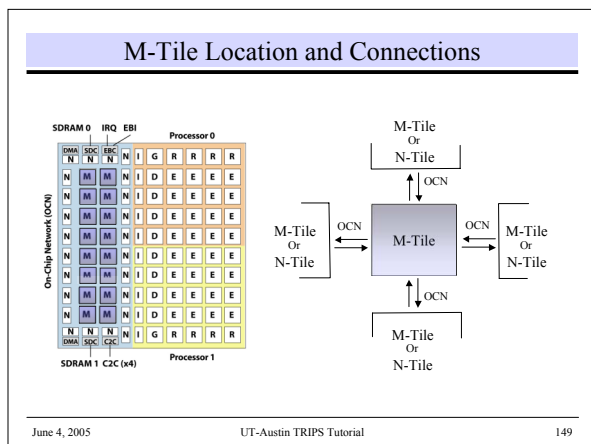
Secondary Memory System

ISCA-32 Tutorial

Changkyu Kim

Department of Computer Sciences
The University of Texas at Austin

June 4, 2005 UT-Austin TRIPS Tutorial 148



M-Tile Network Attributes

- Total 1MB on-chip memory capacity
 - Array of 16 64KB M-Tiles
 - Connected by specialized on-chip network (OCN)
- High bandwidth
 - A peak of 16 cache accesses per cycle
 - 68 GB/sec to the two processor cores
- Low latency for future technologies
 - Non-uniform (NUCA) level-2 cache
- Configuration
 - Scratchpad / Level-2 cache modes on a per-bank basis
 - Configurable address mapping
 - Shared L2 cache / Private L2 cache

June 4, 2005 UT-Austin TRIPS Tutorial 150

Non-Uniform Level-2 Cache (NUCA)

- Non-uniform access latency
 - Closer bank can be accessed faster
- Designed for wire-dominated future technologies
 - No global interconnect
- Connected by switched network
 - Can fit large number of banks with small area overhead
- Allows thousands of in-flight transactions
 - Network capacity = 2100 flits
 - Maximize throughput

June 4, 2005 UT-Austin TRIPS Tutorial 151

Multiple Modes in TRIPS Secondary Memory

■ : M-Tile configured as a level-2 Cache
■ : M-Tile configured as a scratchpad memory

All shared level-2 caches Per-processor level-2 caches All scratchpad memories Half level-2 caches, half scratchpad memories

June 4, 2005 UT-Austin TRIPS Tutorial 152

Major Structures in the M-Tile

- 8-way set associative cache
- 64B Line size
- 1 MSHR entry (Total 16 entries in the whole cache)
- Configurability (tag on/off)
- 1B ~ 64B Read / Write / Swap transactions support
- Error checking and handling

June 4, 2005 UT-Austin TRIPS Tutorial 153

Back-to-Back Read Pipeline

7-cycle latency for transferring all 64B data
4-cycle latency for sending a reply header

	Cycle 0	Cycle 1	Cycle 2	Cycle 3	Cycle 4	Cycle 5	Cycle 6	Cycle 7	Cycle 8	Cycle 9
First Read	Request Arrives at Input FIFO	Tag Access Data Access 1	Data Access 2	Send reply header	Send 1 st 16B of reply data	Send 2 nd 16B of reply data	Send 3 rd 16B of reply data	Send 4 th 16B of reply data		
Second Read		Request Arrives at Input FIFO					Tag Access Data Access 1	Data Access 2	Send reply header	Send 1 st 16B of reply data

Seamless Data Transfer (6.8GB/sec) →

June 4, 2005 UT-Austin TRIPS Tutorial 154

Read Miss Pipeline

4-cycle latency for sending a fill request to SDC

	Cycle 0	Cycle 1	Cycle 2	Cycle 3
Incoming read miss request	Request Arrives at Input FIFO	Tag Access	MSHR Allocation	Send outgoing read miss request

Access the main memory

9-cycle latency for filling 64B data and sending a reply header

	Cycle 0	Cycle 1	Cycle 2	Cycle 3	Cycle 4	Cycle 5	Cycle 6	Cycle 7	Cycle 8
Incoming read reply	Reply Arrives at Input FIFO	Tag Access	Fill 1 st 16B of reply data	Fill 1 st 16B of reply data	Fill 1 st 16B of reply data	Fill 1 st 16B of reply data	MSHR Access	Generate reply header	Send 1 st 16B of reply data

June 4, 2005 UT-Austin TRIPS Tutorial 155

Chip- and System-Level Networks and I/O

ISCA-32 Tutorial

Paul Gratz

Department of Computer Sciences
 The University of Texas at Austin

June 4, 2005 UT-Austin TRIPS Tutorial 156

OCN Location and Connections

June 4, 2005 UT-Austin TRIPS Tutorial 157

Goals and Requirements of the OCN

- Fabric for interconnection of processors, memory and I/O
- Handle multiple types of traffic:
 - Routing of L1 and L2 cache misses
 - Memory requests
 - Configuration and data register reads/writes by external host
 - Inter-chip communication
- OCN Network Attributes
 - Flexible mapping of resources
 - Deadlock free operation
 - Performance (16 Client Links – 6.8GB/sec per link)
 - Guaranteed delivery of request and reply

June 4, 2005 UT-Austin TRIPS Tutorial 158

OCN Protocol

- Flexible mapping of resources
 - System address mapping tables are runtime re-configurable to allow for runtime re-mapping between L2 cache banks and scratchpad memory banks
- Deadlock Avoidance
 - Dimension order routed network (Y-X)
 - Four virtual channels used to avoid deadlock
- Performance
 - Links are 128 bit wide in each direction
 - Credit-based flow control
 - One cycle per hop plus cycle at network insertion
 - 68 GB/sec at processor interfaces

June 4, 2005 UT-Austin TRIPS Tutorial 159

N-Tile Design

- Goals
 - Router for OCN packets
 - System address to network address translation
 - Configurable L2 network addresses
 - One cycle per hop latency
- Attributes
 - Composed of OCN router with translation tables on local input
 - Portions of the system address index into translation tables
 - Translation tables configured through memory mapped registers
 - Error back on invalid addresses

June 4, 2005 UT-Austin TRIPS Tutorial 160

I/O Tiles

- EBC Tile
 - Provides interface between board-level PowerPC 440 GP chip and the TRIPS chip
 - Forwards interrupts to the 440 GP's service routines
- SDC Tile
 - Contains IBM SDRAM Memory Controller
 - Error detection, swap support and address remapping added
- DMA Tile
 - Supports multi-block and scatter/gather transfers
 - Buffered to optimize traffic
- C2C Tile

June 4, 2005 UT-Austin TRIPS Tutorial 161

C2C Tile Design Considerations

- C2C Goals
 - Extend OCN fabric across multiple chips
 - Provide flexible extensibility
- C2C Attributes
 - C2C network protocol similar to OCN except:
 - 64 bit wide
 - 2 virtual channels
 - Operates at up to 266MHz
 - Clock speed configurable at boot
 - Source synchronous off-chip links
 - Multiple clock domains with resynchronization

June 4, 2005 UT-Austin TRIPS Tutorial 162

C2C Network

- Latencies:
 - L2 Hit Latency: 9 – 27 cycles
 - L2 Miss Latency: 26 – 44 cycles (+ SDRAM Access time)
 - L2 Hit Latency (adjacent chip): 33 – 63 cycles
 - L2 Miss Latency (adjacent chip): 50 – 78 cycles (+SDRAM)
 - Each additional C2C hop adds 8 cycles in each direction
- Bandwidth:
 - OCN Link: 6.8 GB/sec
 - One processor to OCN: 34 GB/sec
 - Both processors to OCN: 68 GB/sec
 - Both SDRAMs: 2.2 GB/sec
 - C2C Link: 1.7 GB/sec

June 4, 2005 UT-Austin TRIPS Tutorial 163

Compiler Overview

ISCA-32 Tutorial

Kathryn McKinley

Department of Computer Sciences
 The University of Texas at Austin

June 4, 2005 UT-Austin TRIPS Tutorial 164

Compiler Challenges

“Don’t put off to runtime what you can do at compile time.” Norm Jouppi, May 2005.

- Block constraints & compiler scheduling
 - Makes the hardware simpler
 - Shifts work to compile time

June 4, 2005 UT-Austin TRIPS Tutorial 165

Generating Correct, High Performance Code

- TRIPS specific compiler challenges
 - EDGE execution model
 - What’s a TRIPS block?
- The compiler’s job
 - Correct code
 - Satisfying TRIPS block constraints
 - Optimized code
 - Filling up TRIPS blocks with useful instructions

June 4, 2005 UT-Austin TRIPS Tutorial 166

Block Execution

Address*targets sent to memory, data returned to target

32 read instructions

32 stores

32 loads

32 write instructions

terminating branch

PC read

Reg. banks

Memory

1 - 128 instruction DFG

PC

June 4, 2005 UT-Austin TRIPS Tutorial 167

TRIPS Block Constraints

- **Fixed size:** 128 instructions
 - Padded with no-ops if needed
 - Simpler hardware: instruction size, I-cache design, global control
- **Registers:** 8 reads and writes to each of 4 banks (in addition to 128 instructions)
 - Reduced buffering
 - Read/write instruction overhead, register file size
- **Block Termination:** all stores and writes execute, one branch
 - Simplifies hardware logic for detecting block completion
- **Fixed output:** 32 load or store queue IDs, one branch
 - LSQ size, instruction size

June 4, 2005 UT-Austin TRIPS Tutorial 168

Outline

- Overview of TRIPS compiler components
- How hard is it to meet TRIPS block constraints?
 - Compiler algorithms to enforce them
 - Evaluation
 - **Question:** How often does the compiler under-fill a block to meet a constraint?
 - **Answer:** Almost never
- Optimization
 - High quality TRIPS blocks
 - Scheduling for TRIPS
 - Cache bank optimization
- Preliminary evaluation

June 4, 2005 UT-Austin TRIPS Tutorial 169

Compiler Phases

June 4, 2005 UT-Austin TRIPS Tutorial 170

Compiler Phases (2)

June 4, 2005 UT-Austin TRIPS Tutorial 171

Forming TRIPS Blocks

ISCA-32 Tutorial

Aaron Smith

Department of Computer Sciences
 The University of Texas at Austin

June 4, 2005 UT-Austin TRIPS Tutorial 172

High Quality TRIPS Blocks

- Full
- High ratio of committed to total instructions
 - Dependence height is not an issue
- Minimize branch misprediction
- Meet correctness constraints

June 4, 2005 UT-Austin TRIPS Tutorial 173

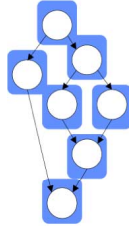
TRIPS Block Constraints

- **Fixed size:** at most, 128 instructions
 - Easy to satisfy by the compiler, but...
 - Good performance requires blocks full of useful instructions
- Registers: 8 reads and writes to each of 4 banks (plus 128 instructions)
- Block Termination
- Fixed output: at most, 32 load or store queue IDs

June 4, 2005 UT-Austin TRIPS Tutorial 174

03: Basic Blocks as TRIPS Blocks

- O3: every basic block is a TRIPS block
- Simple, but not high performance

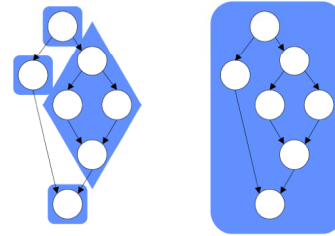


June 4, 2005

UT-Austin TRIPS Tutorial

175

O4: Hammock Hyperblocks as TRIPS Blocks



Two possible hammock hyperblocks for this graph

June 4, 2005

UT-Austin TRIPS Tutorial

176

Static TRIPS Block Instruction Counts

	Static Averages					
	O3			O4		
	min	max	mean	min	max	mean
SPEC2000	7	42	16	8	42	18
EEMBC	7	31	16	9	67	22

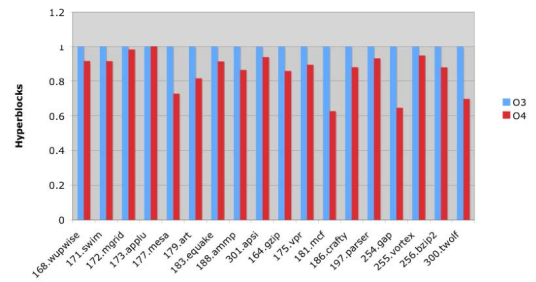
June 4, 2005

UT-Austin TRIPS Tutorial

177

SPEC Total TRIPS Blocks

- 15% average reduction in number of TRIPS blocks between O3 and O4
- Results missing sixtrack, perl, and gcc

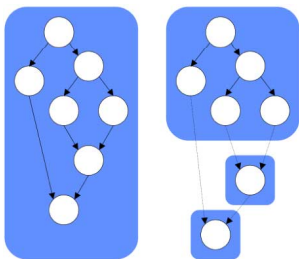


June 4, 2005

UT-Austin TRIPS Tutorial

178

Enforcing Block Size



Initial Hammock Hyperblock After Reverse If-Conversion

Too Big?

1. Unpredicated
 - cut
2. Predicated
 - Reverse if-conversion

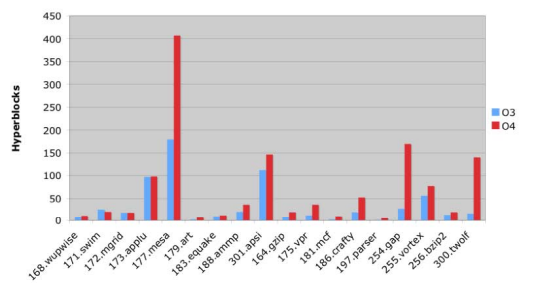
June 4, 2005

UT-Austin TRIPS Tutorial

179

SPEC > 128 Instructions

- At O4 an average of 0.9% TRIPS blocks > 128 insts. (max 6.6%)



June 4, 2005

UT-Austin TRIPS Tutorial

180

In Progress: Tail Duplication

After Reverse If-Conversion

After Tail Duplication

- Tail duplicate A
- Increases block size
- Hot long paths

June 4, 2005 UT-Austin TRIPS Tutorial 181

In Progress: More Aggressive Hyperblocks

```

while (*pp != '\0')
{
  if (*pp == '.') {
    break;
  }
  pp++;
}

```

- Add support for complex control flow (i.e. breaks/continues)
- An example: the number of TRIPS blocks in ammp_1 microbenchmark
 - 38 / 27 / 12 with O3 / O4 (current) / new

June 4, 2005 UT-Austin TRIPS Tutorial 182

While Loop Unrolling

```

while (*pp != '\0')
{
  if (*pp == '.') {
    break;
  }
  pp++;
}

```

- Supported on architectures with predication
- Implemented, but needs improved hyperblock generation

June 4, 2005 UT-Austin TRIPS Tutorial 183

TRIPS Block Constraints

- Fixed size: at most, 128 instructions
- **Registers:** 8 reads and writes to each of 4 banks (plus 128 instructions)
 - Simple linear scan priority order allocator
- Block Termination
- Fixed output: at most, 32 load or store queue IDs

June 4, 2005 UT-Austin TRIPS Tutorial 184

Linear Scan Register Allocation

- 128 registers 32 per 4 banks
- Hyperblocks as large instructions
- Computes liveness over hyperblocks
- Memory savings vs. instructions
 - Fewer hyperblocks than instructions
 - Allocator ignores local variables
- Spills only 5 SPEC2000 vs 18 Alpha
 - gcc: 4290 Alpha - 211 TRIPS
 - sixtrack: 3510 Alpha - 165 TRIPS
 - mesa: 2048 Alpha - 207 TRIPS
 - apsi: 920 Alpha - 7 TRIPS
 - applu: 14 Alpha - 19 TRIPS
- Average spill: 1 store, 2-7 loads
- Spills are less than 0.5% of all dynamic loads/stores

June 4, 2005 UT-Austin TRIPS Tutorial 185

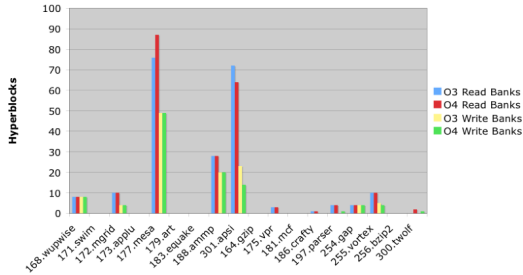
SPEC > 32 (8x4 banks) Registers

- At O4 an average of 0.3% TRIPS blocks have > 32 register reads
- At O4 an average of 0.2% TRIPS blocks have > 32 register writes

June 4, 2005 UT-Austin TRIPS Tutorial 186

SPEC > 8 Registers per Bank

- At O4 an average of 0.12% TRIPS blocks have > 8 read banks
- At O4 an average of 0.07% TRIPS blocks have > 8 write banks



June 4, 2005

UT-Austin TRIPS Tutorial

187

TRIPS Block Constraints

- Fixed size: at most, 128 instructions
- Control flow: 8 branches (soft)
- Registers: 8 reads and writes to each of 4 banks (plus 128 instructions)
- **Block Termination**
- Fixed output: at most, 32 load or store queue IDs

June 4, 2005

UT-Austin TRIPS Tutorial

188

Block Termination

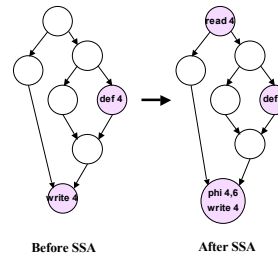
- One branch fires
- All writes complete
 - Write nullification
- All stores complete
 - Store nullification

June 4, 2005

UT-Austin TRIPS Tutorial

189

Register Writes



- How do we guarantee all paths produce a value?
 - Insert corresponding read instructions for all write instructions
 - Transform into SSA form
 - Out of SSA to add moves
- Could nullify writes
 - Increases block size

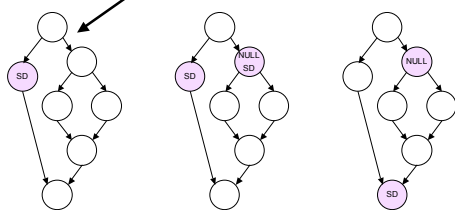
June 4, 2005

UT-Austin TRIPS Tutorial

190

Store Nullification

How do you insert nulls for this? Two possibilities. Which is better?



- Dummy stores
 - Wastes space
 - Easier to schedule?
- Code motion
 - Better resource utilization

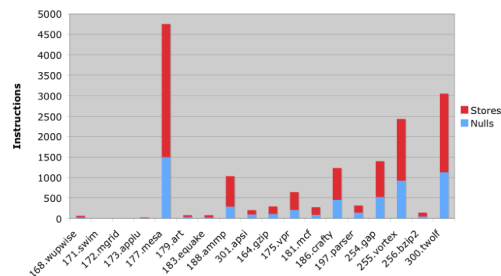
June 4, 2005

UT-Austin TRIPS Tutorial

191

SPEC Store Nullification

- Nullification is 1.2% of instructions for 177.mesa (0.8% dummy stores)
- Is it significant?



June 4, 2005

UT-Austin TRIPS Tutorial

192

TRIPS Block Constraints

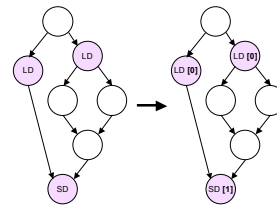
- Fixed size: at most, 128 instructions
- Control flow: 8 branches (soft)
- Registers: 8 reads and writes to each of 4 banks (plus 128 instructions)
- Block Termination: each store issues once
- **Fixed output:** at most, 32 load or store queue IDs

June 4, 2005

UT-Austin TRIPS Tutorial

193

Load/Store ID Assignment



- Ensures Memory Consistency
- Simplifies Hardware
- Overlapping LSQ IDs
 - Increases number of load/store instructions in a TRIPS block
 - But a load and store cannot share the same ID

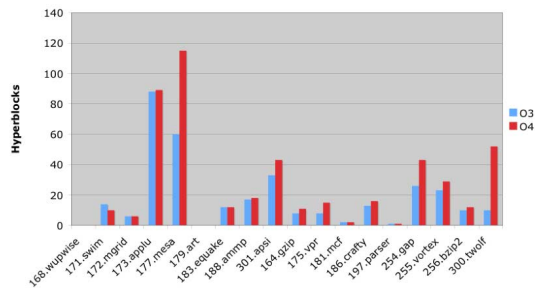
June 4, 2005

UT-Austin TRIPS Tutorial

194

SPEC > 32 LSQ IDs

- At O4 an average of 0.3% TRIPS blocks have > 32 LSQ IDs



June 4, 2005

UT-Austin TRIPS Tutorial

195

Correct Compilation of TRIPS Blocks

- Some new compiler challenges
- New, but straightforward algorithms
- Does not limit compiler effectiveness (so far!)

June 4, 2005

UT-Austin TRIPS Tutorial

196

Code Optimization

ISCA-32 Tutorial

Kathryn McKinley



Department of Computer Sciences
The University of Texas at Austin

June 4, 2005

UT-Austin TRIPS Tutorial

197

Outline

- Overview of TRIPS Compiler components
- Are TRIPS block constraints onerous?
 - Compiler algorithms that enforce them
 - Evaluation
 - Question: How often does the compiler under-fill a block to meet a constraint?
 - Answer: almost never
- Towards optimized code
 - TRIPS specific optimization policies
 - Scheduling for TRIPS
 - Cache bank optimization
- Preliminary evaluation

June 4, 2005

UT-Austin TRIPS Tutorial

198

Towards Optimized Code

- Filling TRIPS blocks with useful instructions
 - Better hyperblock formation
 - Loop unrolling for TRIPS blocks constraints
 - Aggressive inlining
 - Driving block formation with path and/or edge profiles
- Shortening the critical path in a block
 - Tree-height reduction
 - Redundant computation
- 64 bit machine optimizations
- etc.

June 4, 2005 UT-Austin TRIPS Tutorial 199

TRIPS Scheduling Problem

- Place instructions on 4x4x8 grid
- Encode placement in target form

June 4, 2005 UT-Austin TRIPS Tutorial 200

Contrast with Conventional Approaches

- VLIW
 - Relies completely on compiler to schedule code
 - + Eliminates need for dynamic dependence check hardware
 - + Good match for partitioning
 - + Can minimize communication latencies on critical paths
 - Poor tolerance to unpredictable dynamic latencies
 - These latencies continue to grow
- Superscalar approach
 - Hardware dynamically schedules code
 - + Can tolerate dynamic latencies
 - Quadratic complexity of dependence check hardware
 - Not a good match for partitioning
 - Difficult to make good placement decisions
 - ISA does not allow software to help with instruction placement

June 4, 2005 UT-Austin TRIPS Tutorial 201

Dissecting the Problem

- Scheduling is a two-part problem
 - Placement: *Where an instruction executes*
 - Issue: *When an instruction executes*
- VLIW represents one extreme
 - Static Placement and Static Issue (SPSI)
 - + Static Placement works well for partitioned architectures
 - Static Issue causes problems with unknown latencies
- Superscalars represent another extreme
 - Dynamic Placement and Dynamic Issue (DPDI)
 - + Dynamic Issue tolerates unknown latencies
 - Dynamic Placement is difficult in the face of partitioning

June 4, 2005 UT-Austin TRIPS Tutorial 202

EDGE Architecture Solution

- EDGE: Explicit Dataflow Graph Execution
 - Static Placement and Dynamic Issue (SPDI)
 - Renegotiates the compiler/hardware binary interface
- An EDGE ISA explicitly encodes the dataflow graph specifying *targets*

<p style="text-align: center; font-weight: bold;">RISC</p> <pre>i1: movi r1, #10 i2: movi r2, #20 i3: add r3, r2, r1</pre>	<p style="text-align: center; font-weight: bold;">EDGE</p> <pre>ALU-1: movi #10, ALU-3 ALU-2: movi #20, ALU-3 ALU-3: add ALU-4</pre>	
--	--	--

- Static Placement
 - Explicit DFG simplifies hardware → *no HW dependency analysis!*
 - Results are forwarded directly through point-to-point network → *no associative issue queues!*
 - *no global bypass network!*
- Dynamic Instruction Issue
 - Instructions execute in original *dataflow-order*

June 4, 2005 UT-Austin TRIPS Tutorial 203

Scheduling Algorithms (1 & 2)

- CRBLO: list scheduler [PACT 2004]
 - greedy top down
 - C - prioritize critical path
 - R - reprioritize
 - B - load balance for issue contention
 - L - data cache locality
 - O - register bank locality
- BSP: Block Specific Policy
 - Characterizes blocks' ILP and load/stores
 - if $I/s > 50\%$, pre-place them and schedule bottom up
 - elseif $flt\ pt > 50\%$ or $int + flt\ pt > 90\%$ use CRBLO
 - else preplace I/s and schedule top down greedy

June 4, 2005 UT-Austin TRIPS Tutorial 204

Scheduling Algorithms (3)

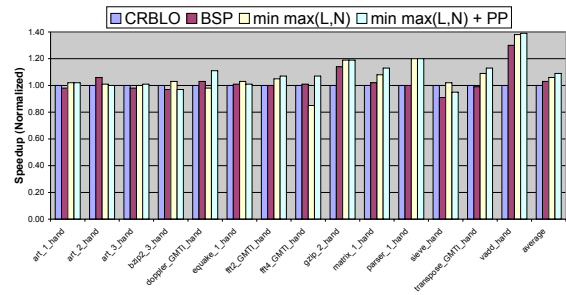
- Physical path scheduling
 - Top down, critical path focused
 - Models network contention (N), in addition to issue contention based on instruction latencies (L)
 - Cost function selects an instruction placement that minimizes maximum contention and latency
 - min of $\max(N, L)$
 - PP - load/store pre-placement
 - tried average, difference, etc.

June 4, 2005

UT-Austin TRIPS Tutorial

205

Improvements on Hand Coded Microbenchmarks



June 4, 2005

UT-Austin TRIPS Tutorial

206

Spatial L1 Bank D-Cache Load/Store Alignment

- Problem
 - Load/store mapping to data L1 cache bank unknown to compiler
- Example

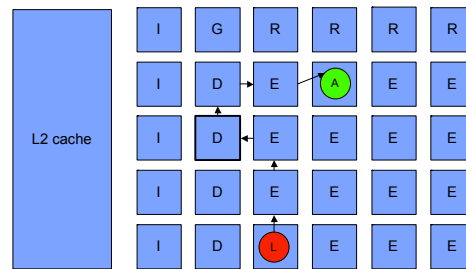

```
for (i=0; i<1024; i++) {
  // A[i], L[i], B[i] to all banks
  A[i] += L[i] + B[i];
}
```

June 4, 2005

UT-Austin TRIPS Tutorial

207

Scheduling for L1 Bank D-cache Communication

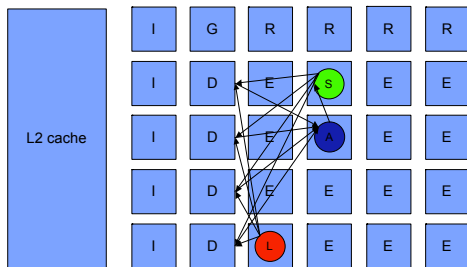


June 4, 2005

UT-Austin TRIPS Tutorial

208

Spatial L1 D-cache Communication



June 4, 2005

UT-Austin TRIPS Tutorial

209

Spatial Load/Store Alignment

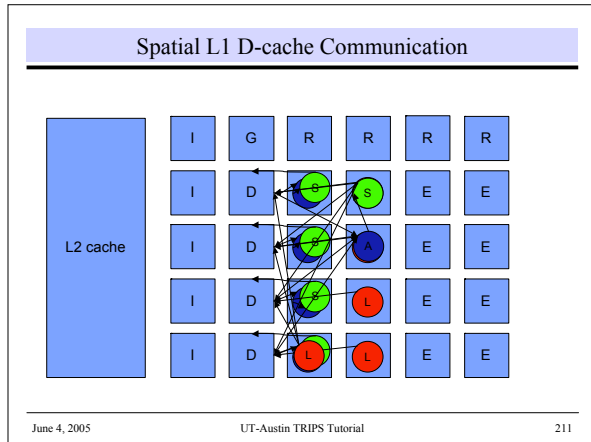
- Solution:
 - Align arrays on bank alignment boundary
 - function of cache line size and number of banks
 - alignment declaration &/or specialized malloc
 - Jump-and-fill new loop transformation
 - similar to unroll-and-jam

```
for (j=0; j<1024; j+=32) {
  for (i=j; i<j+8; i++) {
    C[i] += (A[i] + B[i]); // Bank 0
    C[i+8] += (A[i+8] + B[i+8]); // Bank 1
    C[i+16] += (A[i+16] + B[i+16]); // Bank 2
    C[i+24] += (A[i+24] + B[i+24]); // Bank 3
  }
}
```

June 4, 2005

UT-Austin TRIPS Tutorial

210



Unrolling for Larger Hyperblocks

- Vector Add Unrolled Continuously (tcc 03)**

```

for (i=0; i<1024; j+=32) {
  C[i ] += (A[i ] + B[i ]); // Bank 0
  C[i+1] += (A[i+1] + B[i+1]); // Bank 0
  C[i+2] += (A[i+2] + B[i+2]); // Bank 0
  . . .
  C[i+31] += (A[i+31] + B[i+31]); // Bank 3
}

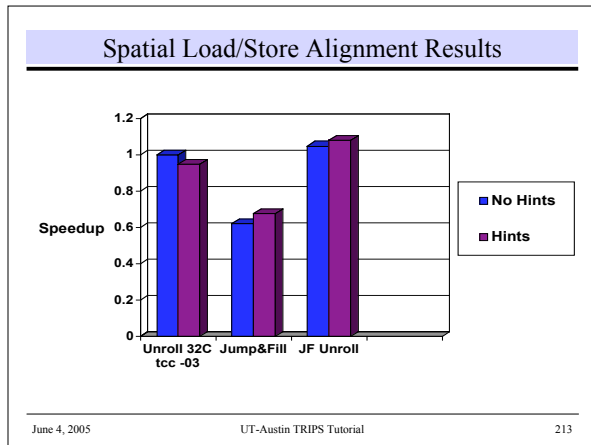
```
- Vector Add Unrolled with Jump-and-Fill (JF Unroll)**

```

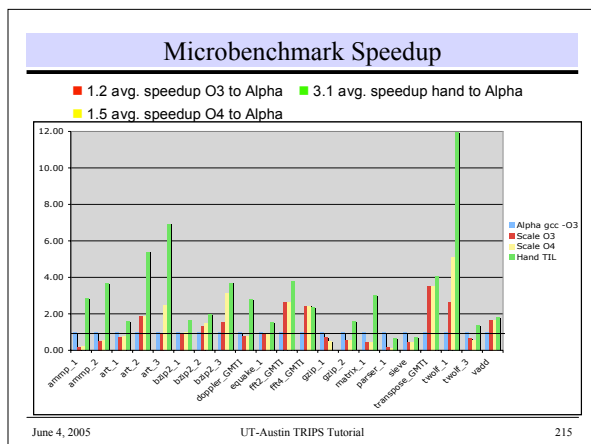
for (i=0; i<1024; j+=32) {
  C[i ] += (A[i ] + B[i ]); // Bank 0
  C[i+8] += (A[i+8] + B[i+8]); // Bank 1
  C[i+16] += (A[i+16] + B[i+16]); // Bank 2
  C[i+24] += (A[i+24] + B[i+24]); // Bank 3
  C[i+1] += (A[i+1] + B[i+1]); // Bank 0
  . . .
  C[i+31] += (A[i+31] + B[i+31]); // Bank 3
}

```

June 4, 2005 UT-Austin TRIPS Tutorial 212



- ### Performance Goals
- Technology comparison points
 - Alpha
 - Fairest comparison we can do
 - High ILP, high frequency, low cycle count machine
 - gcc
 - Alpha compiler
 - Scale on Alpha
 - Scale on TRIPS
 - Performance goal: 4x speed up over Alpha
 - Benchmarks
 - Hand coded microbenchmarks
 - Metric for measuring compiler success
 - Industry standards: EEMBC, SPEC
- June 4, 2005 UT-Austin TRIPS Tutorial 214



- ### Progress Towards Correct & Optimized Code
- SPEC 2000 C/Fortran-77 -- 03: basic blocks
 - March 2004: 0/21
 - November 2004: 6/21
 - March 2005: 15/21
 - May 2005: 19/21
 - SPEC 2000 C/Fortran-77 -- 04: hammock hyperblocks
 - May 2005: 8/21
 - EEMBC -- 04
 - May 2005: 30/30
 - GCC torture tests and NIST -- 04
 - May 2005: 30/30
 - Plenty of work left to do!**
- June 4, 2005 UT-Austin TRIPS Tutorial 216

Wrap-up and Discussion

ISCA-32 Tutorial

TRIPS Team and Guests



Department of Computer Sciences
The University of Texas at Austin

June 4, 2005

UT-Austin TRIPS Tutorial

217

Project Plans

- Tools release
 - Hope to build academic/industrial collaborators
 - Plan to release compilers, toolchain, and simulators sometime in the (hopefully early) fall
- Prototype system
 - Plan to have it working December 2005/January 2006
 - Software development effort increasing after tape-out
- Commercialization
 - Actively looking for interested commercial partners
 - Discussing best business models for transfer to industry
 - Happy to listen to ideas
- Will EDGE architectures become widespread?
 - Many questions remain unanswered (power, performance)
 - Promising initial results, hope to have more concrete answers by next spring
 - A key question is: how successful will universal parallel programming become?
 - Perhaps room for parallel programming & EDGE architectures

June 4, 2005

UT-Austin TRIPS Tutorial

218

And finally ...

THANK YOU for your interest, questions and taking all of this time!

Subset of complete manuals and specifications available at:
<http://www.cs.utexas.edu/users/cart/trips>

June 4, 2005

UT-Austin TRIPS Tutorial

219

Appendix A - Supplemental Materials

- ISA Specification
- Bandwidth summary
- Additional Tile Details

June 4, 2005

UT-Austin TRIPS Tutorial

220

ISA: Header Chunk Format

Bit Offsets			
	31	27	0
0	H0	Read 0.0	Write 0.0
4	H1	Read 1.0	Write 1.0
8	H2	Read 2.0	Write 2.0
12	H3	Read 3.0	Write 3.0
16	H4	Read 0.1	Write 0.1
20	H5	Read 1.1	Write 1.1
24	H6	Read 2.1	Write 2.1
28	H7	Read 3.1	Write 3.1
...
96	H24	Read 0.6	Write 0.6
100	H25	Read 1.6	Write 1.6
104	H26	Read 2.6	Write 2.6
108	H27	Read 3.6	Write 3.6
112	H28	Read 0.7	Write 0.7
116	H29	Read 1.7	Write 1.7
120	H30	Read 2.7	Write 2.7
124	H31	Read 3.7	Write 3.7

Byte Offsets

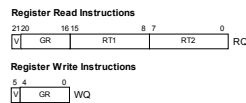
- Includes 32 General Register “read” instructions
- Includes 32 General Register “write” instructions
- Up to 128 bits of header information is encoded in the upper nibbles
 - Block Type
 - Block Size
 - Block Flags
 - Store Mask

June 4, 2005

UT-Austin TRIPS Tutorial

221

ISA: Read and Write Formats



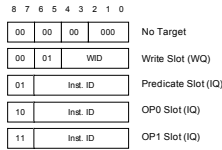
- General Register access is controlled by special read and write instructions
- Read instructions will be executed from the Read Queue
- Write instructions will be executed from the Write Queue
- Read instructions may target the 1st or 2nd operand slot of any instruction
- General registers are divided into 4 architectural banks
 - 32 registers per bank
 - Bank i holds GRs $i, i+4, i+8, \dots$
 - Up to 8 reads and 8 writes may be performed at each bank
 - Bank position is implicit based on read and write instructions in block header
 - Implicit two bits added to 5-bit general register fields for 128 registers total

June 4, 2005

UT-Austin TRIPS Tutorial

222

ISA: Microarchitecture-Specific Target Formats



- A 9-bit general target specifier is used by most instructions
- Read target specifiers are truncated to 8 bits, with the implicit 9th bit always set to 1 (they cannot target a write slot or a predicate slot)
- The all-zero encoding is used to represent no target
- Write Queue entry 0.0 is implicitly targeted by branch instructions and cannot be explicitly targeted
- Most instructions are allowed to target register outputs, predicates, 1st operands, and 2nd operands

June 4, 2005

UT-Austin TRIPS Tutorial

223

ISA: More Details on Predication

PR	Description
00	Not Predicated
01	(Reserved)
10	Predicated Upon False
11	Predicated Upon True

- Any instruction using the G, I, L, S, or B format may be predicated
- A 2-bit predicate field specified whether an instruction is predicated and, if so, upon what condition
- Predicated instructions must receive all of their normal operands plus a matching predicate before they are allowed to execute
- Instructions that meet these criteria are said to be *enabled*
- Instructions that don't meet these criteria are said to be *inhibited*
- Inhibited instructions may cause other dependent instructions to be indirectly inhibited
- A block completes executing when all of its register and memory outputs are produced (it is not necessary for all of its instructions to execute)

June 4, 2005

UT-Austin TRIPS Tutorial

224

ISA: Loads and Stores

Mnemonic	Name	Format	Sources	Targets
LB	Load Byte	L	OP0, IMM	T0
LBS	Load Byte Signed	L	OP0, IMM	T0
LH	Load Halfword	L	OP0, IMM	T0
LHS	Load Halfword Signed	L	OP0, IMM	T0
LW	Load Word	L	OP0, IMM	T0
LWS	Load Word Signed	L	OP0, IMM	T0
LD	Load Doubleword	L	OP0, IMM	T0
SB	Store Byte	S	OP0, OP1, IMM	None
SH	Store Halfword	S	OP0, OP1, IMM	None
SW	Store Word	S	OP0, OP1, IMM	None
SD	Store Doubleword	S	OP0, OP1, IMM	None

- LB: TO = ZEXT(MEM_B[OP0 + SEXT(IMM)])
- LBS: TO = SEXT(MEM_B[OP0 + SEXT(IMM)])
- LH: TO = ZEXT(MEM_H[OP0 + SEXT(IMM)])
- LHS: TO = SEXT(MEM_H[OP0 + SEXT(IMM)])
- LW: TO = ZEXT(MEM_W[OP0 + SEXT(IMM)])
- LWS: TO = SEXT(MEM_W[OP0 + SEXT(IMM)])
- LD: TO = ZEXT(MEM_D[OP0 + SEXT(IMM)])
- SB: MEM_R[OP0 + SEXT(IMM)] = OP1[15:0]
- SH: MEM_W[OP0 + SEXT(IMM)] = OP1[31:0]
- SW: MEM_D[OP0 + SEXT(IMM)] = OP1[63:0]
- SD: MEM_D[OP0 + SEXT(IMM)] = OP1[63:0]

June 4, 2005

UT-Austin TRIPS Tutorial

225

ISA: Integer Arithmetic

Mnemonic	Name	Format	Sources	Targets
ADD	Add	G	OP0, OP1	T0, T1
ADDI	Add Immediate	I	OP0, IMM	T0
SUB	Subtract	G	OP0, OP1	T0, T1
SUBI	Subtract Immediate	I	OP0, IMM	T0
MUL	Multiply	G	OP0, OP1	T0, T1
MULI	Multiply Immediate	I	OP0, IMM	T0
DIVS	Divide Signed	G	OP0, OP1	T0, T1
DIVSI	Divide Signed Immediate	I	OP0, IMM	T0
DIVU	Divide Unsigned	G	OP0, OP1	T0, T1
DIVUI	Divide Unsigned Immediate	I	OP0, IMM	T0

- There is no support for overflow detection or extended arithmetic
- The multiply instruction may be split into signed and unsigned versions

June 4, 2005

UT-Austin TRIPS Tutorial

226

ISA: Integer Logical

Mnemonic	Name	Format	Sources	Targets
AND	Bitwise AND	G	OP0, OP1	T0, T1
ANDI	Bitwise AND Immediate	I	OP0, IMM	T0
OR	Bitwise OR	G	OP0, OP1	T0, T1
ORI	Bitwise OR Immediate	I	OP0, IMM	T0
XOR	Bitwise XOR	G	OP0, OP1	T0, T1
XORI	Bitwise XOR Immediate	I	OP0, IMM	T0

- XORI may be used to perform a bitwise complement – XORI(A, -1)

June 4, 2005

UT-Austin TRIPS Tutorial

227

ISA: Integer Shift

Mnemonic	Name	Format	Sources	Targets
SL	Shift Left Logical	G	OP0, OP1	T0, T1
SLLI	Shift Left Logical Immediate	I	OP0, IMM	T0
SRL	Shift Right Logical	G	OP0, OP1	T0, T1
SRLI	Shift Right Logical Immediate	I	OP0, IMM	T0
SRA	Shift Right Arithmetic	G	OP0, OP1	T0, T1
SRAI	Shift Right Arithmetic Immediate	I	OP0, IMM	T0

- Operand 1 provides the data to be shifted
- Operand 2 or IMM provides the shift count
- The lower 6 bits of the shift count are interpreted as an unsigned quantity
- The higher bits of the shift count are ignored

June 4, 2005

UT-Austin TRIPS Tutorial

228

ISA: Integer Extend

Mnemonic	Name	Format	Sources	Targets
EXTSB	Extend Signed Byte	G	OP0	T0, T1
EXTSH	Extend Signed Halfword	G	OP0	T0, T1
EXTSW	Extend Signed Word	G	OP0	T0, T1
EXTUB	Extend Unsigned Byte	G	OP0	T0, T1
EXTUH	Extend Unsigned Halfword	G	OP0	T0, T1
EXTUW	Extend Unsigned Word	G	OP0	T0, T1

- These are better than doing a left shift followed by a right shift

June 4, 2005

UT-Austin TRIPS Tutorial

229

ISA: Integer Test

Mnemonic	Name	Format	Sources	Targets
TEQ	Test EQ	G	OP0, OP1	T0, T1
TNE	Test NE	G	OP0, OP1	T0, T1
TLE	Test LE	G	OP0, OP1	T0, T1
TLEU	Test LE Unsigned	G	OP0, OP1	T0, T1
TLT	Test LT	G	OP0, OP1	T0, T1
TLTU	Test LT Unsigned	G	OP0, OP1	T0, T1
TGE	Test GE	G	OP0, OP1	T0, T1
TGEU	Test GE Unsigned	G	OP0, OP1	T0, T1
TGT	Test GT	G	OP0, OP1	T0, T1
TGTU	Test GT Unsigned	G	OP0, OP1	T0, T1

- Test instructions produce true (1) or false (0)

June 4, 2005

UT-Austin TRIPS Tutorial

230

ISA: Integer Test #2

Mnemonic	Name	Format	Sources	Targets
TEQI	Test EQ Immediate	I	OP0, IMM	T1
TNEI	Test NE Immediate	I	OP0, IMM	T1
TLEI	Test LE Immediate	I	OP0, IMM	T1
TLEUI	Test LE Unsigned Immediate	I	OP0, IMM	T1
TLTI	Test LT Immediate	I	OP0, IMM	T1
TLTUI	Test LT Unsigned Immediate	I	OP0, IMM	T1
TGEI	Test GE Immediate	I	OP0, IMM	T1
TGEUI	Test GE Unsigned Immediate	I	OP0, IMM	T1
TGTI	Test GT Immediate	I	OP0, IMM	T1
TGTUI	Test GT Unsigned Immediate	I	OP0, IMM	T1

- Test instructions produce true (1) or false (0)

June 4, 2005

UT-Austin TRIPS Tutorial

231

ISA: Floating Point Arithmetic

Mnemonic	Name	Format	Sources	Targets
FADD	FP Add	G	OP0, OP1	T0, T1
FSUB	FP Subtract	G	OP0, OP1	T0, T1
FMUL	FP Multiply	G	OP0, OP1	T0, T1
FDIV	FP Divide (simulator only!)	G	OP0, OP1	T0, T1

- These operations are performed assuming double-precision values
- All results are rounded as necessary using the default IEEE rounding mode (round-to-nearest)
- No exceptions are reported
- There is no support for gradual underflow or denormal representations

June 4, 2005

UT-Austin TRIPS Tutorial

232

ISA: Floating-Point Test

Mnemonic	Name	Format	Sources	Targets
FEQ	FP Test EQ	G	OP0, OP1	T0, T1
FNE	FP Test NE	G	OP0, OP1	T0, T1
FLE	FP Test LE	G	OP0, OP1	T0, T1
FLT	FP Test LT	G	OP0, OP1	T0, T1
FGE	FP Test GE	G	OP0, OP1	T0, T1
FGT	FP Test GT	G	OP0, OP1	T0, T1

- These operations are performed assuming double-precision values
- Test instructions produce true (1) or false (0)

June 4, 2005

UT-Austin TRIPS Tutorial

233

ISA: Floating Point Conversion

Mnemonic	Name	Format	Sources	Targets
FITOD	Convert Integer to Double FP	G	OP0	T0, T1
FDTOI	Convert Double FP to Integer	G	OP0	T0, T1
FSTOD	Convert Single FP to Double FP	G	OP0	T0, T1
FDTOS	Convert Double FP to Single FP	G	OP0	T0, T1

- FITOD: Converts a 64-bit signed integer to a double-precision float
- FDTOI: Converts a double-precision float to a 64-bit signed integer
- FSTOD: Converts a single-precision float to a double-precision float
- FDTOS: Converts a double-precision float to a single-precision float
- FSTOD is needed to load of a single-precision float value
- FDTOS is needed to store of a single-precision float value

June 4, 2005

UT-Austin TRIPS Tutorial

234

ISA: Control Flow

Mnemonic	Name	Format	Sources	Targets
BR	Branch	B	OP0	PC
BRO	Branch with Offset	B	OFFSET	PC +=
CALL	Call	B	OP0	PC
CALLO	Call with Offset	B	OFFSET	PC +=
RET	Return	B	OP0	PC
SCALL	System Call	B	None	PC

- Predication should be used for conditional branches
- Special branch and call instructions may produce an address offset (in chunks) rather than a full address
- Call and return behave like normal branches (but may be treated in a special way be a hardware branch predictor)
- The System Call instruction will trigger a System Call Exception

June 4, 2005

UT-Austin TRIPS Tutorial

235

ISA: Miscellaneous

Mnemonic	Name	Format	Sources	Targets
NULL	Nullify Output	G	None	T0, T1
MOV	Move	G	OP0	T0, T1
MOV1	Move Immediate	I	IMM	T0
MOV3	Move to three targets	M3	OP0	T0, T1, T2
MOV4	Move to four targets	M4	OP0	T0, T1, T2, T3
MFC	Move from PC	I	None	T0
GENS	Generate Signed Constant	C	CONST	T0
GENU	Generate Unsigned Constant	C	CONST	T0
APP	Append Constant	C	OP0, CONST	T0
NOP	No Operations	C	None	None
Lock	Load and Lock	L	OP0	T0

- NULL is a special instruction for cancelling one or more block outputs
- MOV is the same as RPT (but with a conventional name)
- The generate and append instructions may be used to form large constants
- Three instructions are required to form a 48-bit constant
- Four instructions are required to form a 64-bit constant

June 4, 2005

UT-Austin TRIPS Tutorial

236

Bandwidth Summary

Interface	Width	Freq	Multiplier	Bytes/Sec
Processor Data Cache (Fill + Spill)	128 bits	533 MHz	4 * 2	68 GB/s
Processor Data Cache (Load + Store)	64 bits	533 MHz	4 * 2	34 GB/s
Processor Inst Cache (Fetch or Fill)	128 bits	533 MHz	4	34 GB/s
Processor to OCN Interface	128 bits	533 MHz	5 * 2 * 0.8	68 GB/s
On-Chip Network Link (64B Read)	128 bits	533 MHz	0.8	6.8 GB/s
On-Chip Memory Bank (64B Read)	128 bits	533 MHz	0.8	6.8 GB/s
Chip-to-Chip Network Link (64B Read)	64 bits	(266 MHz)	0.8	(1.7 GB/s)
DDR 266 SDRAM (64B Read)	64 bits	266 MHz	0.5	1.1 GB/s
External Bus Interface	32 bits	66 MHz	0.33	88 MB/s

(Maximum bandwidth calculations)

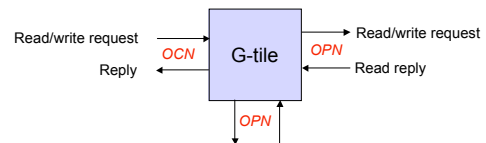
June 4, 2005

UT-Austin TRIPS Tutorial

237

G-tile: Access to Architected Registers

Access to architected registers allowed when processor is halted



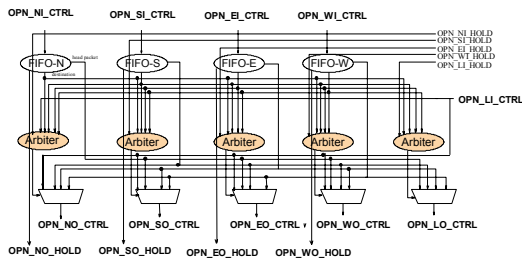
Only one read/write request handled at a time

June 4, 2005

UT-Austin TRIPS Tutorial

238

OPN: Detailed Router Schematic



June 4, 2005

UT-Austin TRIPS Tutorial

239

E-Tile: List of Pipelines

- Instruction Dispatch Pipeline (GDN)
 - Update Instruction Register File and Status Buffer
- Operand Delivery Pipeline (OPN)
 - Update Operand Register File and Status Buffer
- Instruction Wakeup and Select Pipeline
 - Two stage instruction selection among three potential candidates
 - Definite ready instruction
 - Instruction Late Selection: instruction with remote/local bypassed operand
 - Guarantee absence of retire conflict
- Instruction Execute and Writeback Pipeline
 - Pipelined execution
 - Retire local targets through local bypass
 - Retire remote targets through OPN
- Instruction Flush / Commit Pipeline (GCN)
 - Clear block state (pipeline registers, status buffer etc.)

June 4, 2005

UT-Austin TRIPS Tutorial

240

