

# **A VLIW Processor for Binary Translation**

**Kemal Ebcioglu, Jason Fritts,  
Steve Kosonocky, Michael Gschwind,  
Erik Altman, Krishnan Kailas, Terry Bright**

**ICCD '98**

**October 7, 1998**



# •VLIW Architecture

- Benefits:
  - high issue rate
  - simple, fast hardware
- Problems:
  - binary compatibility
  - branch performance
- Solutions:
  - dynamic binary translation
  - control path combining

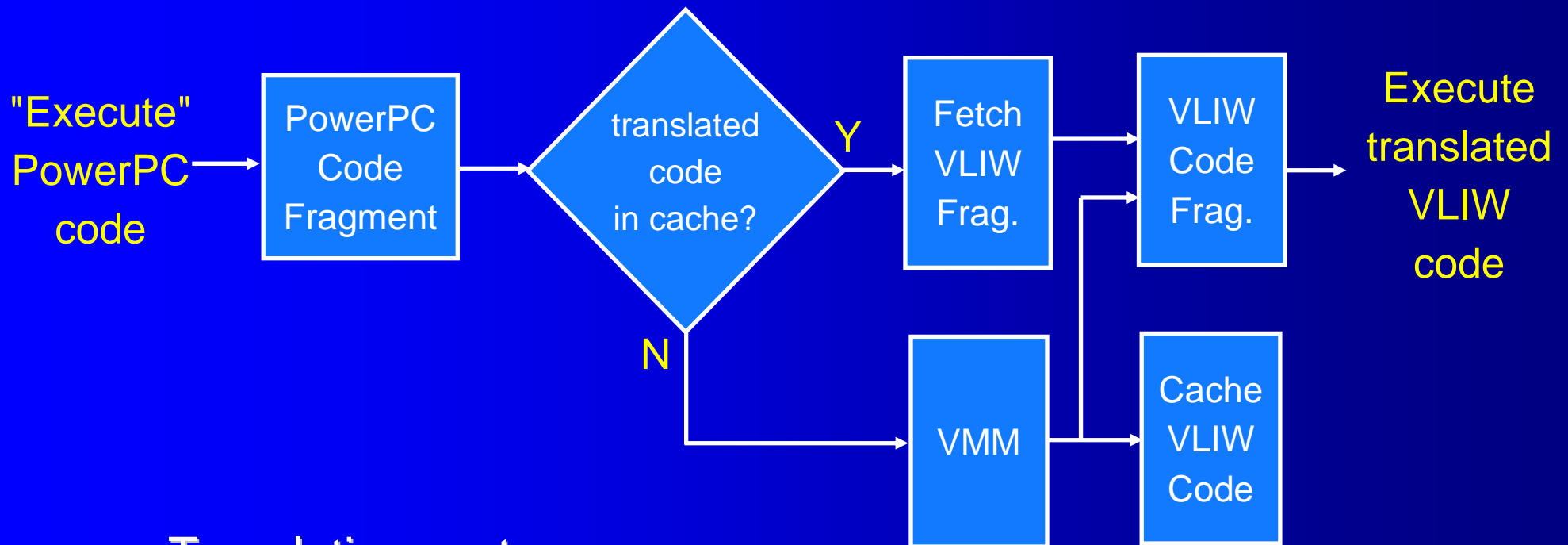
# Binary Translation

- Overcomes compatibility problem
  - PowerPC
  - other IBM architectures
  - Java
- Exploit benefits of recompilation
  - speculation
  - predication
  - trace-based scheduling

# Binary Translation in Software

- Execute PowerPC with 100% compatibility
  - uses DAISY instruction set
    - *Dynamically Architected Instruction Set from Yorktown*
- Virtual Machine Manager (VMM) performs translation:
  - begin executing PowerPC code
  - translate into simple RISC primitives
  - parallelize and schedule into VLIW instructions
  - cache translated VLIW code

# Binary Translation Flowchart



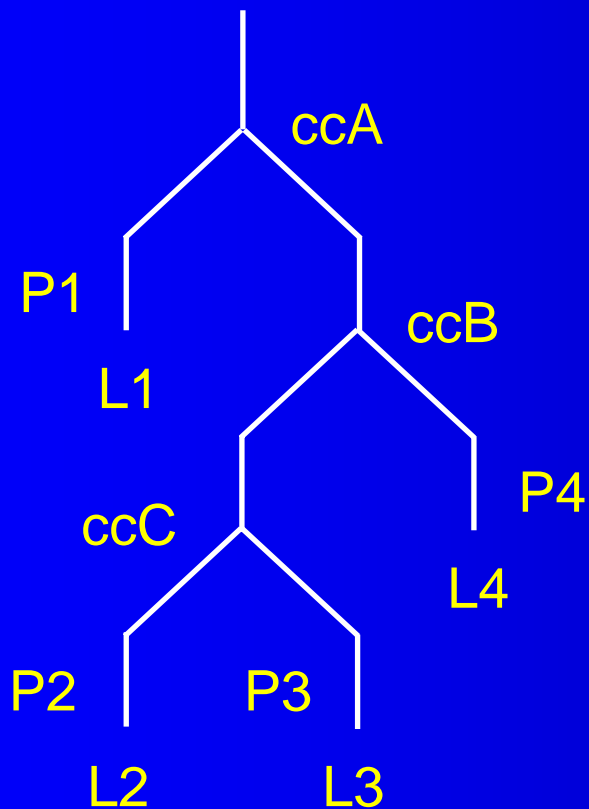
- Translation cost
  - 4000 cycles per PowerPC instruction
  - cache hierarchy flushed

# Tree VLIW Instructions

- Multi-way path selection and branching
  - condition code registers select single path
  - each path indicates a branch target
  - only operations on selected path commit
  - branch every cycle
- VLIW tree information contained in header
  - up to 4 branch targets per VLIW



# Tree Execution Example



```
if ( ccA = F )
    execute ops on path P1
    branch to L1
else
    if ( ccB = T )
        execute ops on path P4
        branch to L4
    else
        if ( ccC = F )
            execute ops on path P2
            branch to L2
        else
            execute ops on path P3
            branch to L3
```

# DAISY Instruction Set

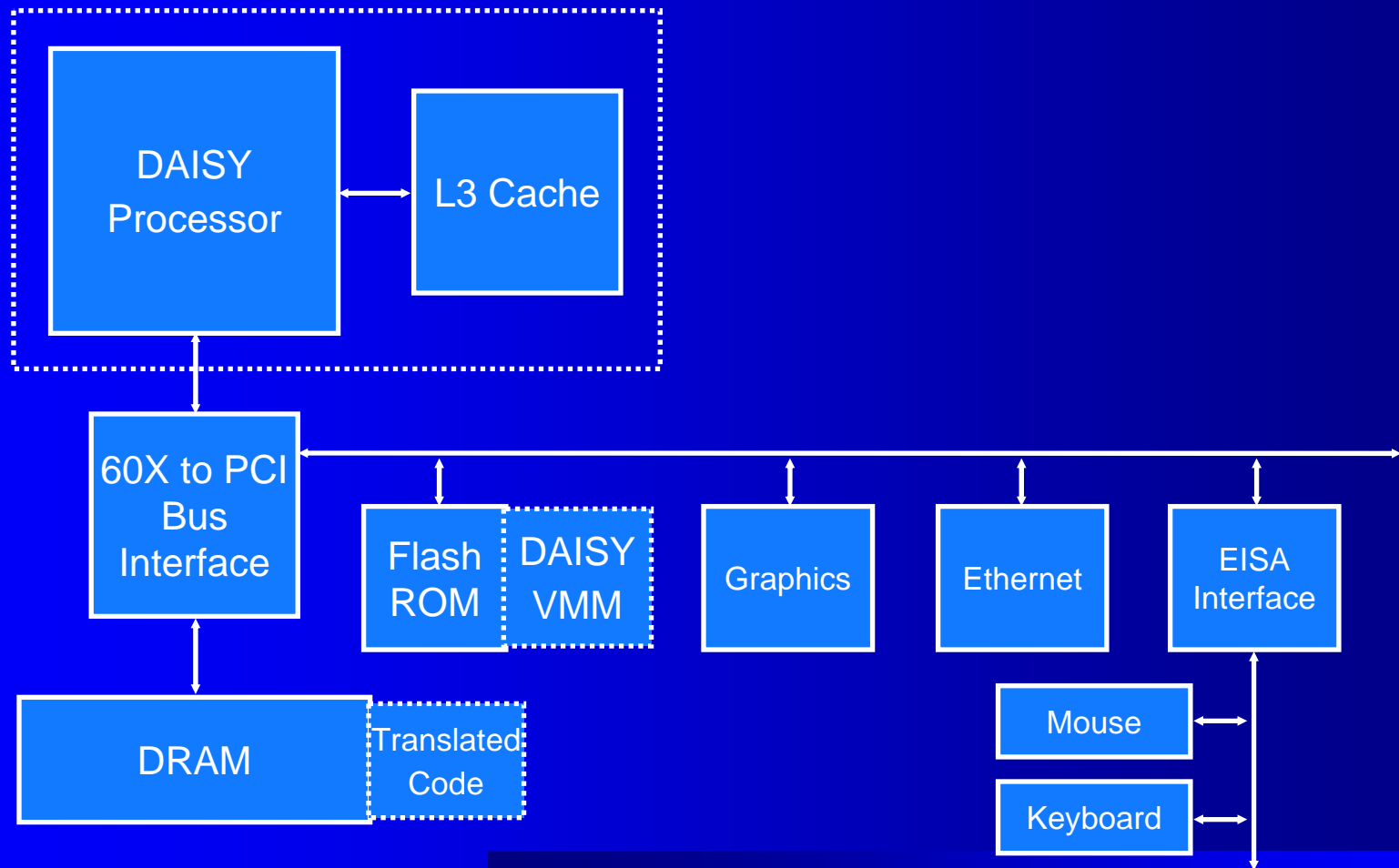
- Operation set conducive to binary translation
  - primarily simple, generic operations
  - some compatibility operations
  - load VLIW instruction address (LVIA) operation
  - no floating point, multiply, or divide
- Speculative execution support
  - speculative registers
  - register extender bits (carry, overflow, exception)
  - commit operation
- Out of order load support
  - load verify operations

# Outline

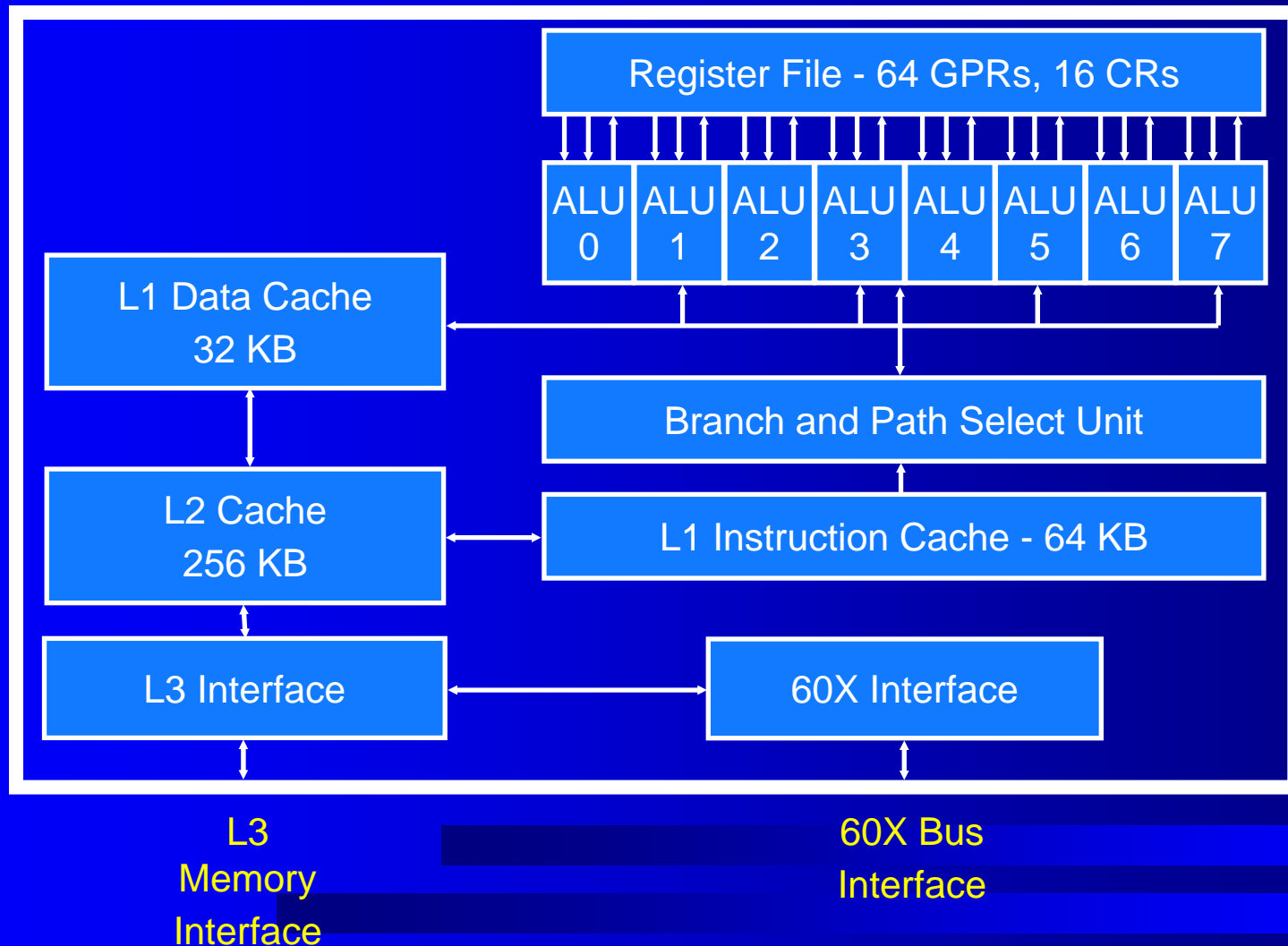
---

- Introduction
  - Binary Translation
  - Tree VLIWs and DAISY
- Basic Architecture
- Implementation
  - ALU
  - Memory Hierarchy
- Performance
- Conclusions

# System Architecture



# Logical Architecture

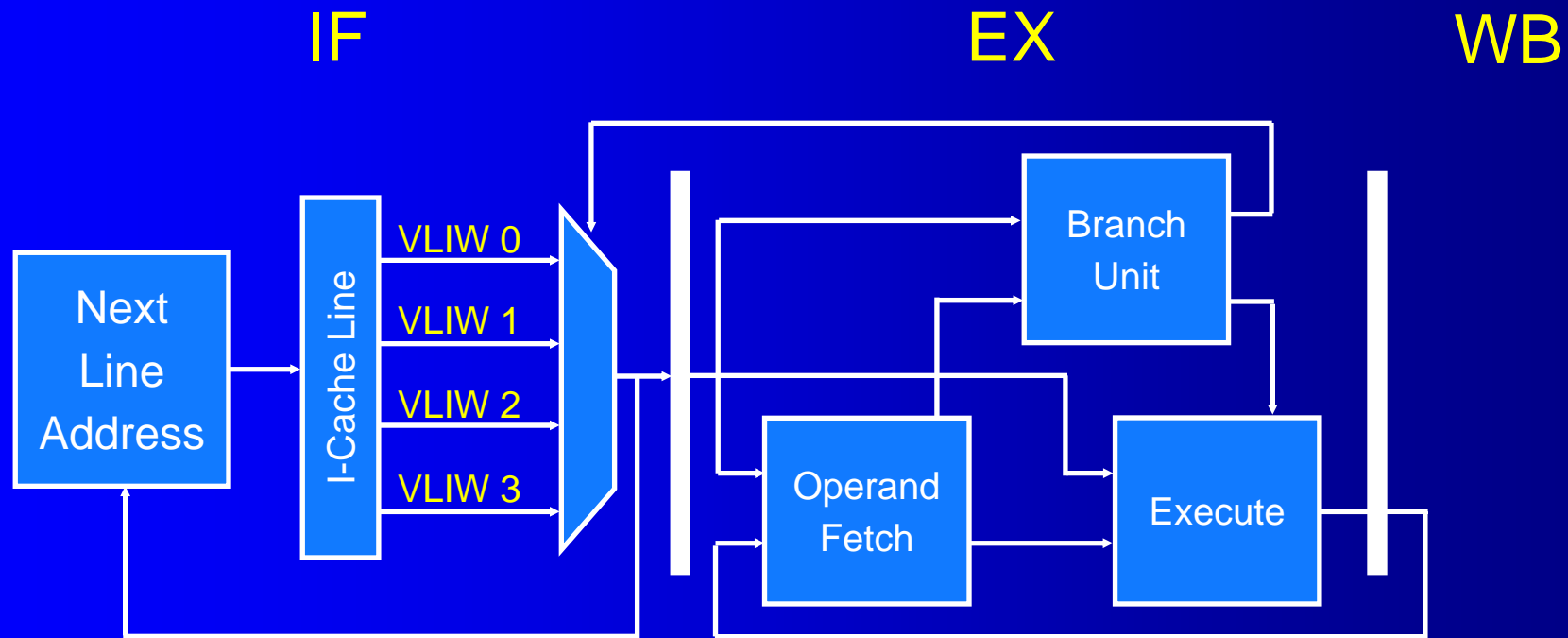




**ALU**

---

# Pipeline



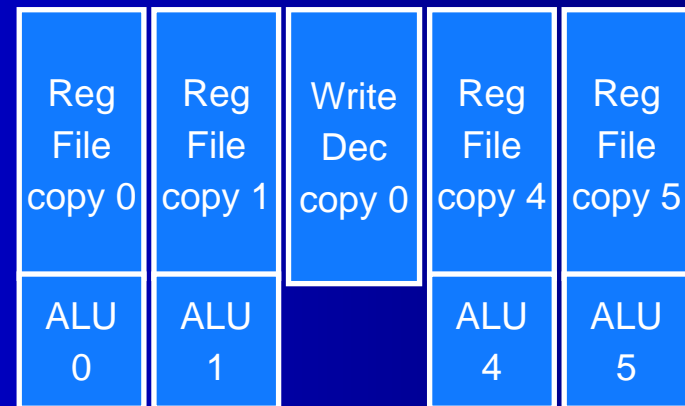
- IF - instruction fetch
- EX - register fetch and execute
- WB - write back

# ALU Operations

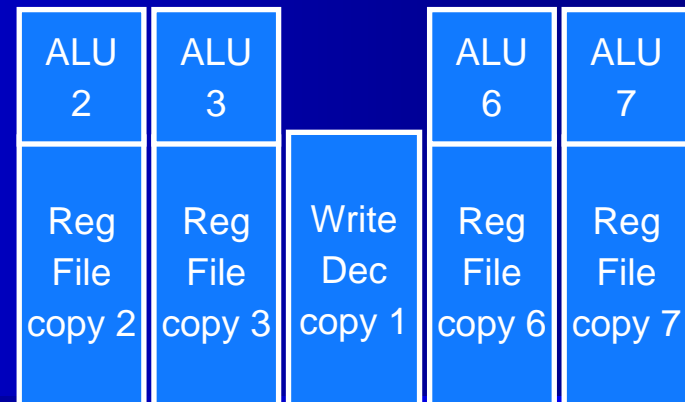
Operations:	Max per VLIW
add/sub	8
load/store	odd 4
addr gen	even 4
shifts	8
logic	8
compare	8
cond ops	8
misc ops	8
load VLIW instr addr	odd 4
commit	8
load verify	odd 4
extenders	odd 4

# ALU

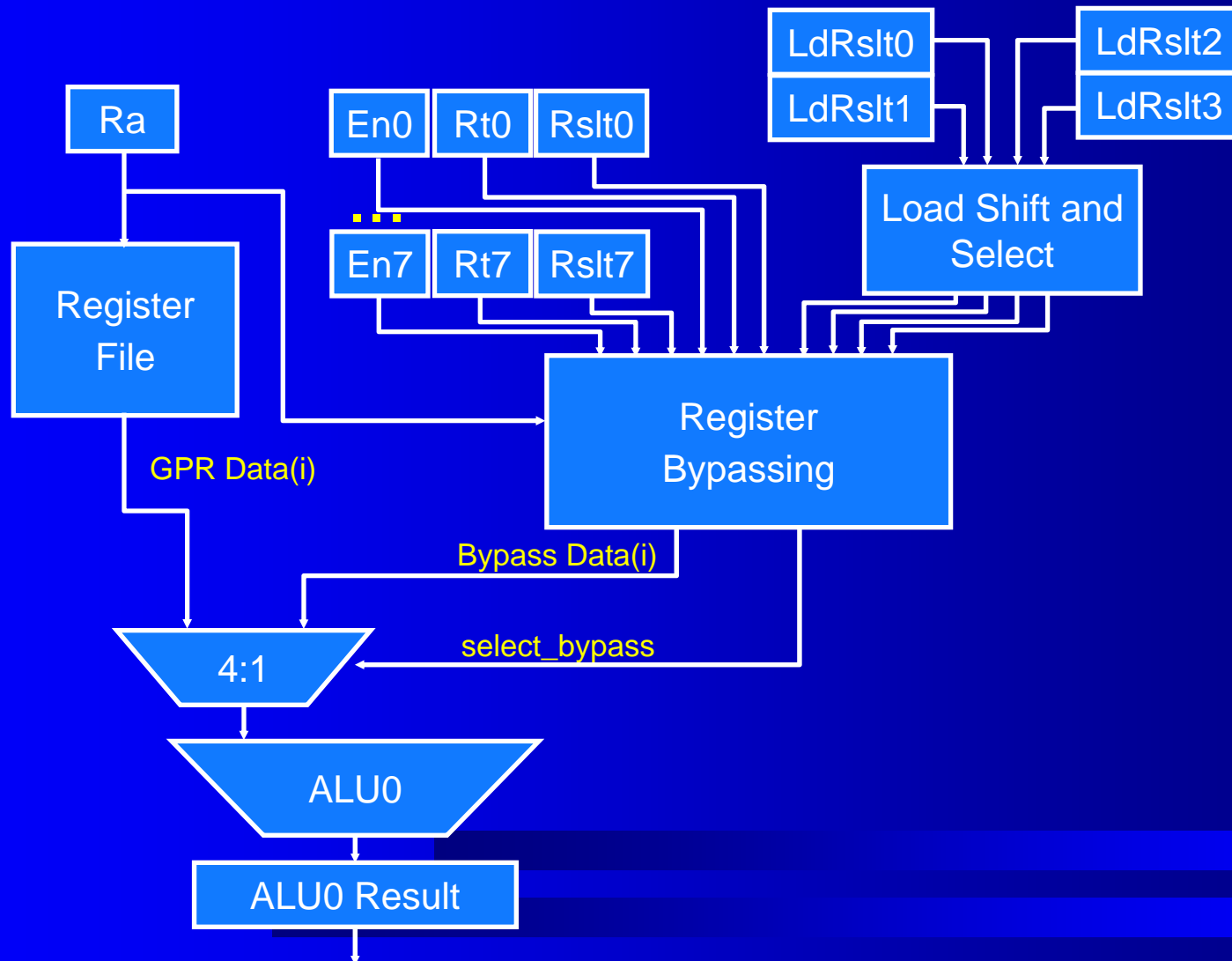
- 8 issue slots
  - single cycle execution
  - integer operations
  - up to 4 memory accesses per VLIW
- 8 copies of register file
  - 8 write / 2 read ports each
  - reduces read time
  - write speed not critical in current ASIC technology



bus signals



# ALU Critical Path



# Control Critical Path

- Stall and Exception Timing

- stalls / exceptions recognized during cycle after occurrence
- invalid results written in previous cycle
- last valid state is restored and re-executed upon returning from stall

Stall and Exception Timing:



\* I Stall in n+1  
D Stall in n  
Exception in n

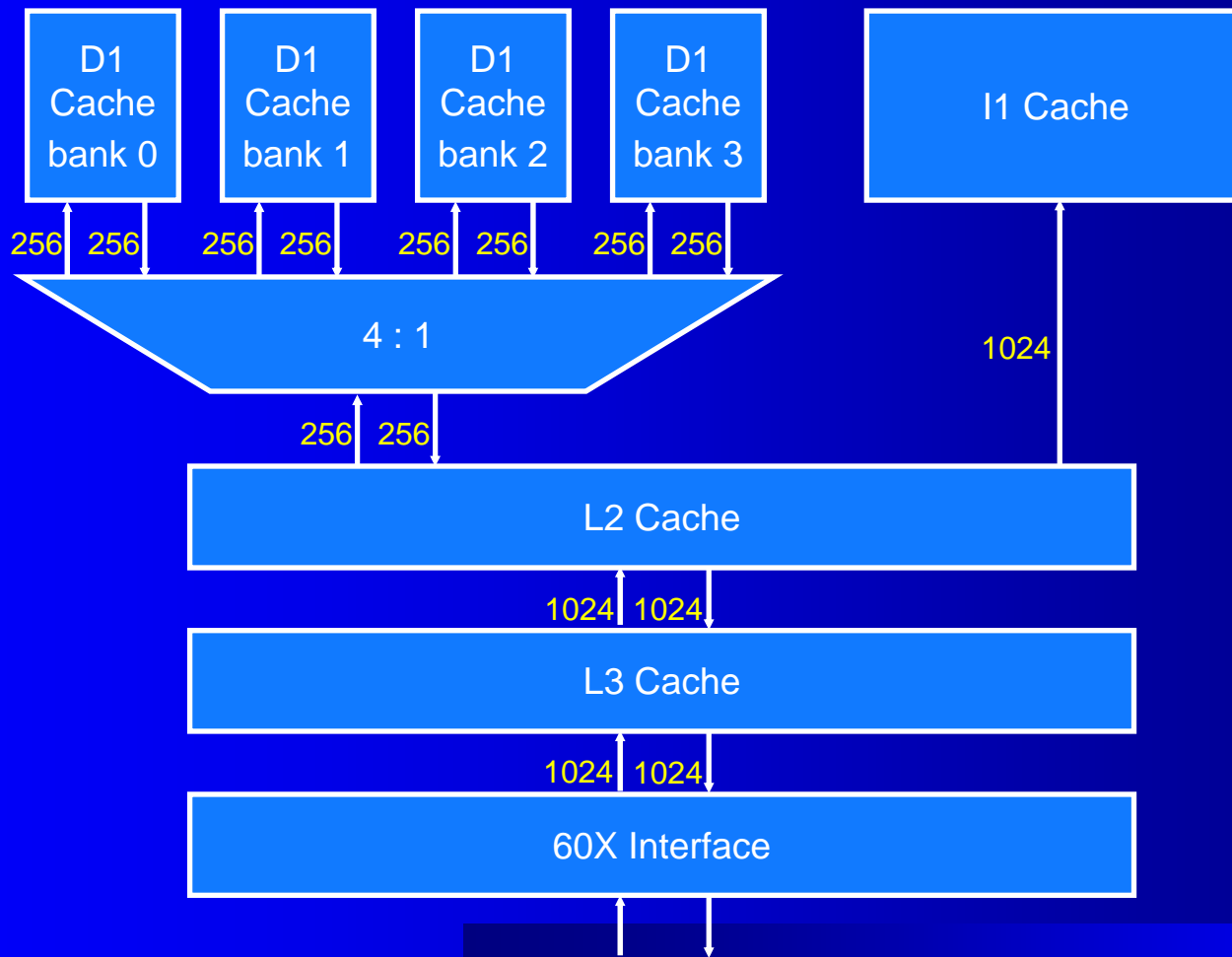




# Memory Hierarchy

---

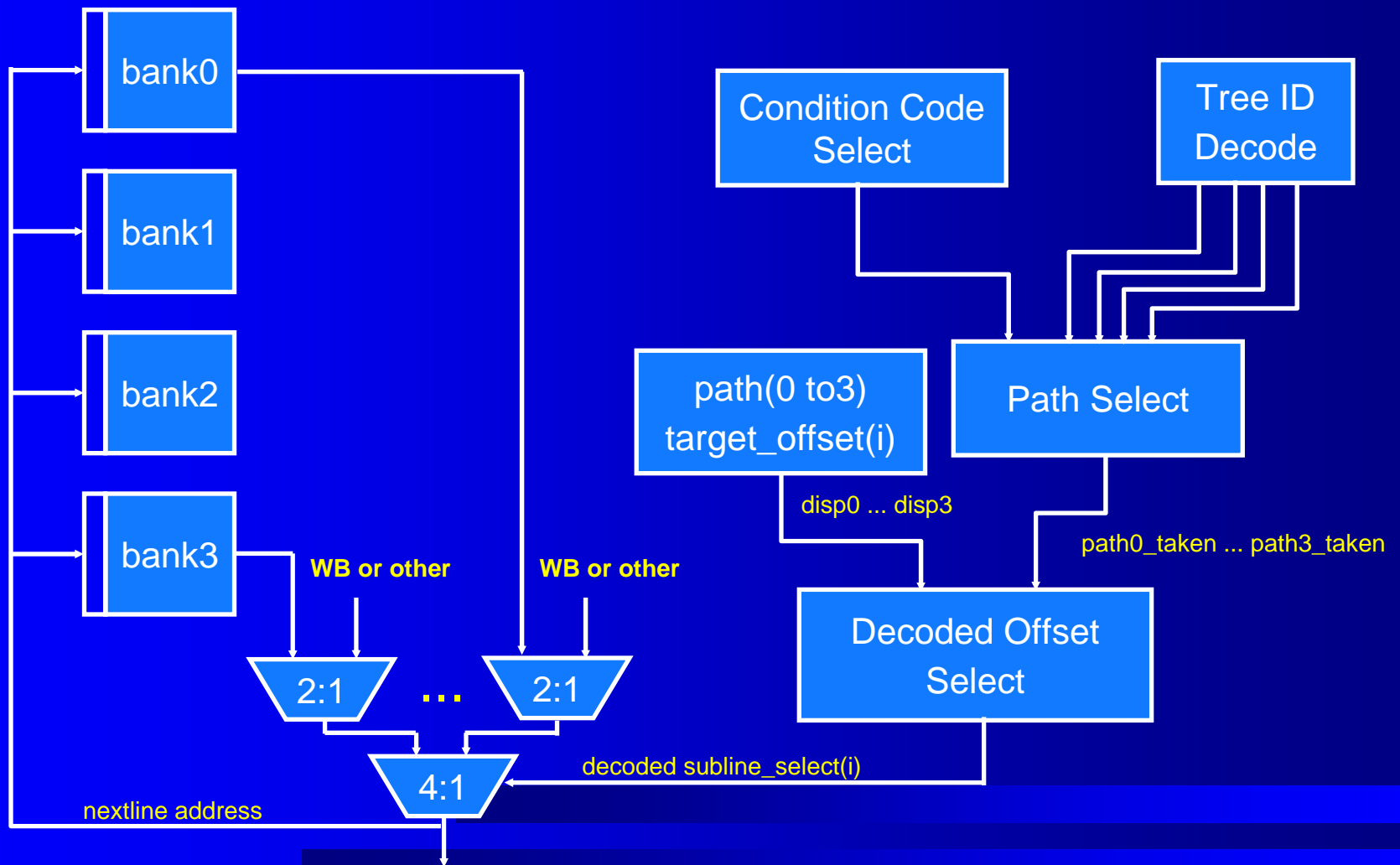
# Memory Hierarchy



# L1 Instruction Cache

- 64 KBytes
- Direct-mapped, 128 byte lines
- Four partitions
  - each partition services two issue slots
  - enables fast multi-way branching
    - each partition has an identical branch unit
- Organization of each partition
  - 4 banks
    - each bank contains one of four possible branch targets
  - contains two operations and control header for each target VLIW

# I1 Cache Critical Path



# L1 Data Cache

- 32 KBytes
- Direct-mapped, 32 byte lines
- 4 read ports, 4 write ports
- Two identical copies
  - halves number of read ports per copy
  - enables single-cycle load operations
- Organization of each copy
  - 2 read ports, 4 write ports per copy
  - 8 banks of single-port memory
  - 1 write buffer per port, per bank (16 total)

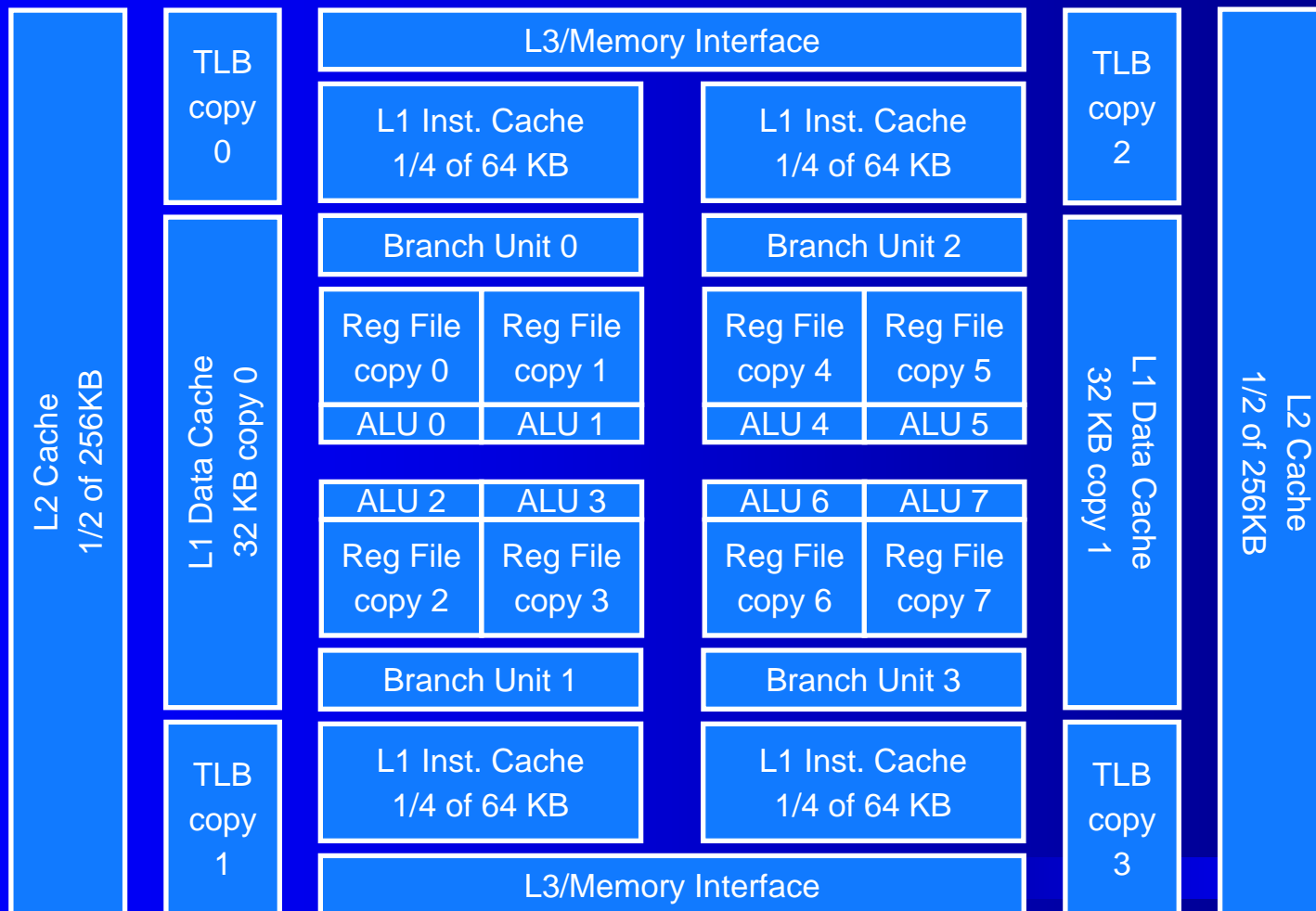
# L2 Cache

- 256 KBytes
- Direct mapped, 128 byte lines
- 2 read ports, 1 write port
  - I1 only reads from L2 cache
  - D1 reads or writes to L2 cache
- Supports different data size for I1 and D1 caches
  - 128 byte line size for I1 cache
  - 32 byte line size for D1 cache
- Performs sequencing between the three ports

# L3 Cache

- 16 MB off-chip
- Direct mapped, 128 byte lines
- On-chip L3 controller
- Physical data store
  - organized as 4 banks
  - 32 128K x 36-bit synchronous SRAM chips
  - 8 ns initial delay, 5 ns pipelined cycle time
- Typical L3 access
  - 1 directory access
  - 4 read/write accesses per line

# Physical Implementation



# Simulated Performance in ASIC Technology

- ASIC Technology
  - SA-12 .25 $\mu$ m CMOS
  - effective gate length .18 $\mu$ m
  - 2.5 V supply
  - 5 wiring levels
- Preliminary gate counts and area
  - 486K complex logic gates
  - 478 KBytes SRAM memory
  - 300 mm<sup>2</sup>
- Critical path simulations
  - 350 MHz nominal performance

# Scheduling Performance

Benchmark	Infinite CPI	I-Cache CPI	D-Cache CPI	TLB CPI	Translation CPI	Total CPI
compress	0.36	0.00	0.31	0.01	0.00	0.68
gcc	0.45	0.38	0.07	0.00	0.02	0.92
go	0.56	0.56	0.20	0.00	0.00	1.32
jpeg	0.34	0.00	0.05	0.01	0.00	0.40
li	0.40	0.02	0.05	0.00	0.00	0.47
m88ksim	0.34	0.02	0.06	0.00	0.00	0.42
perl	0.48	0.06	0.00	0.00	0.00	0.54
vortex	0.37	0.13	0.30	0.07	0.00	0.87
tpcc	0.46	0.89	0.48	0.16	0.00	1.99
SPECint95 geom. mean	0.41	-	-	-	-	0.65

- Encouraging ILP performance compared to superscalar processor implementations

# Conclusions

- Good speed for ASIC 6S2 standard cell technology
  - early simulations indicate 350 MHz typical cycle time
  - anticipate much higher frequencies with full custom
  - optimized stall and exception architecture to achieve high frequency
    - using rollback of processor pipeline registers
- Area
  - area reasonable for servers
  - reduce cache size for network computers
  - full custom will reduce area
- Performance dependent upon scheduling and code reuse
  - good ILP performance over superscalar processor implementations
  - need large memory store for maximum code reuse
    - current cache CPI numbers are relatively high