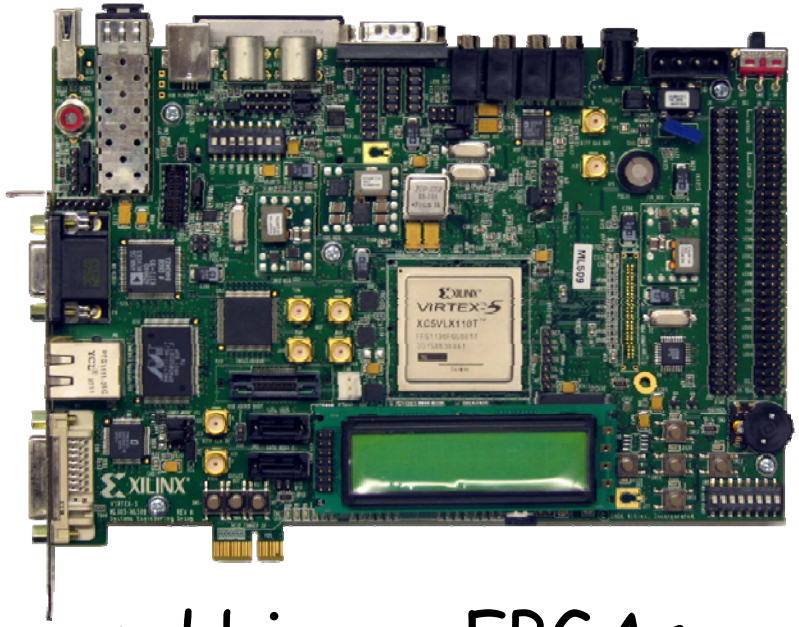
Beehive, a Multi-core Platform for Low-level Systems Research

Andrew Birrell, Tom Rodeheffer,
Chuck Thacker

Microsoft Research, Silicon Valley

Ingredients

- Xilinx field-programmable gate array (FPGA)
- XUPV5 development board ($750 academic)
  - Virtex-5 XC5VLX110T:
    - 17280 x 4 flip-flops
    - 17280 x 4 LUTs
    - 148 x 4KByte BRAMs
  - Gbit Ethernet
  - 2 GB DRAM



- Or BEE3 board with more and bigger FPGAs

Results so far

- Multi-core RISC (entirely home-grown)
- 100 MHz, one cycle per instruction (if cached)
- 13 cores per FPGA (using only half the FPGA)
- C compiler (and most of an MSIL one)
- Completely malleable architecture

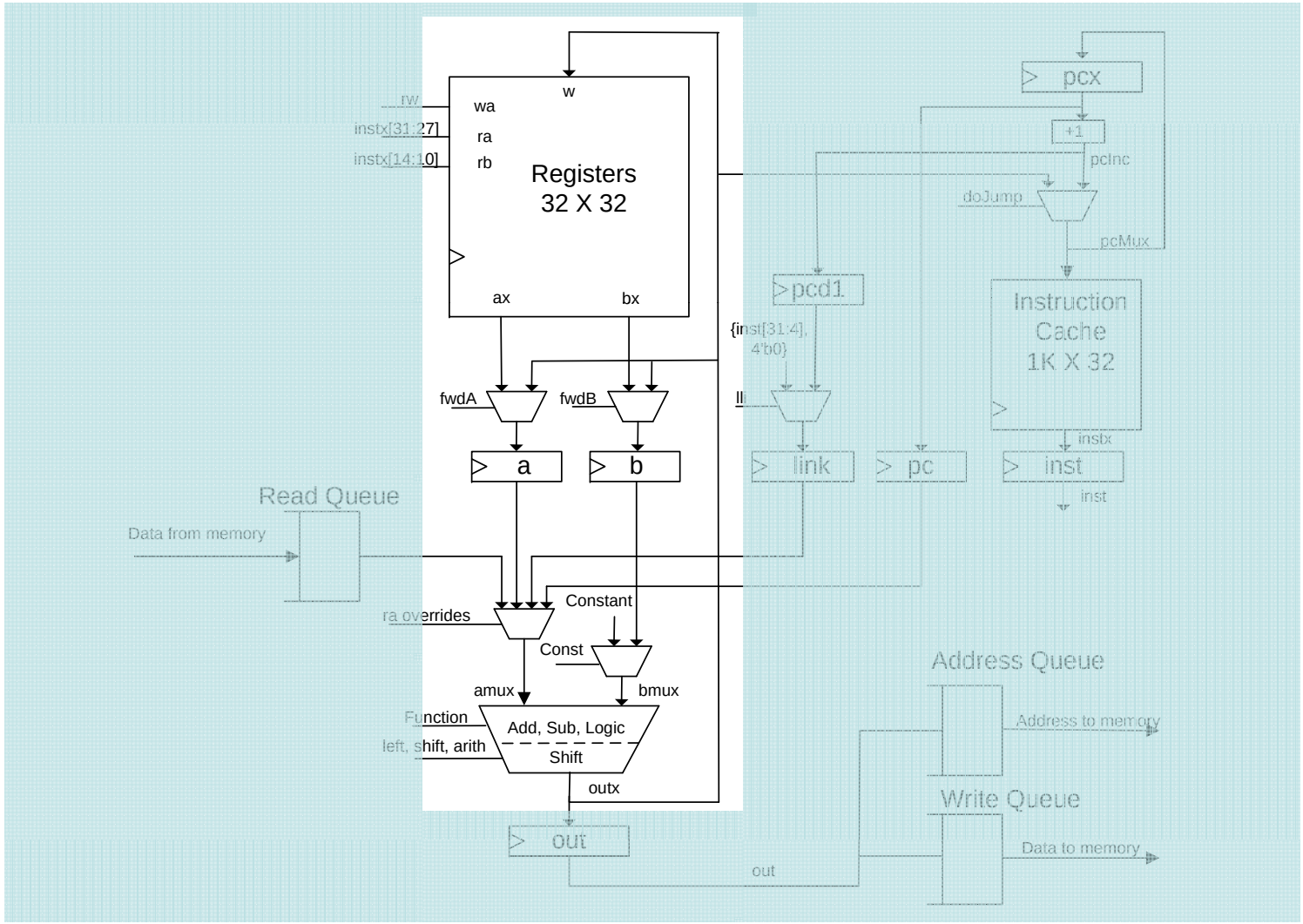
Goals

- Have a mutable computer architecture
  - Make changes in hours or days, not months
  - More realistically than any simulation
- Design blank-sheet system software
  - Without an existing OS
  - In high-level languages (like C#)
- Do both: systems research ↔ architecture
- Usable by universities

Research Examples (Hypothetical)

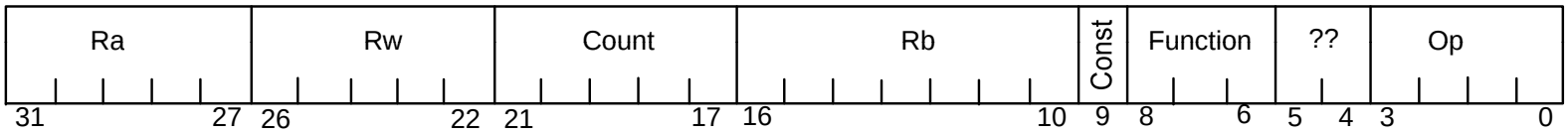
- Forget shared memory, use message passing
- Transactional memory …
  - Apples-to-apples comparison with monitors/CVs
- Do we really need …
  - Coherent shared memory?
  - Interrupts?
  - Virtual memory?
  - Kernel mode?
  - An operating system?

Per-core CPU


w
wa
ra
rb
rw
instx[31:27]
instx[14:10]
Registers
32 X 32
ax
bx
fwdA
fwdB
a
b
pcx
+1
pcInc
doJump
pcMux
pcd1
{inst[31:4], 4'b0}
Instruction Cache 1K X 32
instx
link
pc
inst
inst
Read Queue
Data from memory
ra overrides
Constant
Const
amux
bmux
Function
left, shift, arith
Add, Sub, Logic
Shift
outx
out
Address Queue
Address to memory
Write Queue
Data to memory
out

Instructions

- 32-bit word (I and D)
- 32 x 32-bit registers

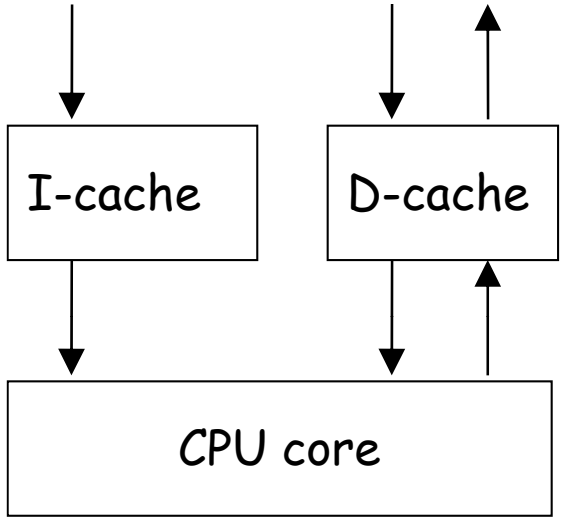


- Rw = (Ra Function Rb) Op Count
  - Function = add, subtract, and, or, xor, etc
  - Op = shift left, shift right, cyclic shift, etc.

- Variations for jumps and memory access

Memory

- 32-bit word, word addressed
  - No byte addressing (sort-of)
- Per-core 1K-word D-cache
- Per-core 1K-word I-cache
  - 8-word lines, direct mapped



- Half of address space is I/O space

Memory Instructions

- Separate addressing and data access
  - Three per-core queues: AQ, RQ, WQ

- Read:

```
AQR r1 r2 0     // Writes r1 but also appends to AQR
...             // Then some time later ...
LD r4 RQ 0      // Moves memory data from RQ to r4
```

- Write:

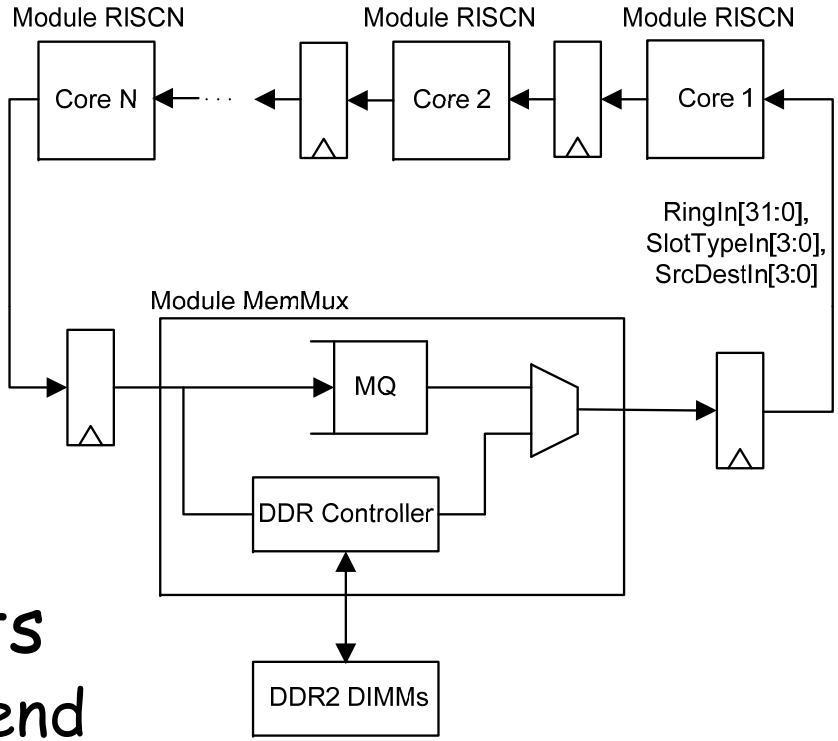
```
AQW r1 r2 0     // Writes r1 but also appends to AQW
LD WQ r4 0      // Appends data from r4 to WQ
```

Architectural Curiosities

- No memory coherence
- No interrupts
- No memory protection
- No virtual memory
- No kernel mode
- No memory bus or I/O bus …

Interconnect: the Ring

- Minimize wires
- Passes through:
  - Each core
  - Ethernet controller
  - DRAM controller
- "train": token + contents
  - Node can modify or append
  - Buffered in DRAM controller node
- Used for real memory and inter-core messages



Inter-core Messages

- Sent by core #x, destination core #y
- Up to 60 words
- Doesn't use the memory system
- No wake-ups: each core polls its message queue
- Accessed by placing an I/O address on AQ

Inter-core Binary Semaphores

- Abstractly, each is a value in [0..1] s.t.
  - "P" = atomic(if > 0 then decrement else fail)
  - "V" = atomic(if == 0 then increment else nothing)
- Implemented by 64-bit register per core
  - Semaphore's "value" is sum of its value in registers
- Uses ring messages, not the memory system
- Polling, not wake-ups
  - E.g. "acquire mutex" spins until "P" succeeds.

Using the Ring: Ethernet

- Ethernet is just a specialized core, #14
  - With an Ethernet attached, and no real memory
- Uses DMA (via ring) for packet contents
- Controlled by messages:
  - Core #n -> core #14: use buffers from address X
  - Core #n -> core #14: send packet at address Y
  - Core #14 -> core #n: packet arrived, at address Z
- One MAC address per core
  - Broadcast/multicast not fully supported

Software Tools

- GCC
  - C to assembler
- Basm
  - Custom assembly language to relocatable binary
- Bld
  - Link binaries into final bootfile image
- Hard-wired bootstrap code
  - Use TFTP to load bootfile into DRAM

C Support Libraries

- Parts of libc: malloc, free, printf
  - "malloc" uses separate cache lines for each item
  - "int i CACHELINE" for global in separate cache line
- Single-core libraries:
  - thread/mutex/CV
  - IP, ARP, UDP, DHCP, DNS, TCP
- Basic multi-core support in "libmc":
  - Library implements "main"; application provides:
    - "mcinit" called first, "mcmain" called per-core
  - Library provides inter-core putchar, malloc, free

Programming in C# and its friends

- Strategy: just compile MSIL (bytecodes)
  - Parse via reflection
  - Covers numerous languages with no extra effort
- Tactic: MSIL to C
- Status
  - First 90% works
  - Working on the rest (particularly exceptions)
- By-product: portable MSIL-to-C compiler

Debugging (work-in-progress)

- Remote debug with GDB via TCP over Ethernet
- Dedicate core #1 for teledebug nub
  - Dedicated code, with no bugs ☺
  - High-level debugging down to the bare metal
- "debug unit" at each core #n:
  - Breakpoint stops core #n, sends message to core #1
  - Or, "kiss of death" message from #1 stops core #n
  - Message from #1 to core #n resumes execution
  - Details: cache flushing, RQ, WQ, condition codes
  - Debug unit is working today (used by libmc)

Creeping Kernelism

- Special hardware state after breakpoints
- Time-slice breakpoints triggered by core #1?
- Breakpoints look a lot like slow interrupts?
- Base-limit registers for multi-image programs?
- Etc.
- But still: adding features only if we need them

What's Next

- Barrelfish port (MSR Cambridge and ETH)
- Try it out on some students (MIT)
- Implement transactional memory
  - Not "software transactional memory" ☺
- Use monitors instead of lock to aid coherence?
- Give it away to more universities