CS 597d - EPIC

Professor David August
Fall, 1999

Course Information

- Project Meetings are at 3pm on Thursday.

Zhigang Hu	Project				
Jiong Luo	Project				
Jordan Parker	Project	Obs1	Obs2	Obs3	Obs4
Shaojie Wang	Project	Obs1	Obs2	Obs3	Obs4
Subramanian Rajagopalan	Project	Obs1		Obs3	
Grant Wallace	Project	Obs1	Obs2	Obs3	Obs4
Zhining Huang	Project	Obs1	Obs2	Obs3	
Manish Vachharajani	Project	Obs1	Obs2	Obs3	Obs4
Hongli Zhang	Lect6	Obs1	Obs2	Obs3	Obs4
Zhijie Shi	Lect5	Obs1	Obs2	Obs3	Obs4
Xiao Yang	Lect6	Obs1	Obs2	Obs3	Obs4
Dongming Jiang	Lect5				
Cheng Liao	Lect4				
David Brooks	Project				
Matthias Jacob					
Yuan Xie					
Neophytus Michael					
Hongzhang Shan					
Xianfeng Zhou					

Papers

- An Efficient Algorithm for Exploiting Multiple Arithmetic Units - R. M. Tomosulo
- Checkpoint Repair for High-Performance Out-of-Order Execution Machines - Hwu and Patt
- The Cydra 5 Departmental Supercomputer - Rau and others

Assorted 4 Letter Acronyms

- This lecture compares:
  - RISC - Reduced Instruction Set Computing
  - CISC - Complex Instruction Set Computing
  - VLIW - Very Long Instruction Word computing
  - EPIC - Explicitly Parallel Instruction Computing
- An architecture may not fall into any category.
- An architecture may fall into two or more categories.
- This classification system is mostly a product of marketing.

RISC: any computer announced after 1985.

Steven Przybylski

Reduced Instruction Set Computing (RISC)

- Designed to be easy to pipeline.
- Simple instruction set - “atomic” tasks.
- Fixed length instructions. (Usually 32 bits)
- Load/Store architecture.
- Designers discovered that static scheduling improved pipeline performance.
- Claim: 2 to 5 times performance increase over CISC
  Reality: Little, if any, performance increase.

Most companies have had a RISC product.

HP: PA-RISC; Intel: i860; SGI, Digital: MIPS; Sun: SPARC; AMD: 29000; Motorola 88000

Reduced Instruction Set Computing (RISC)

```
for (i=1; i <= 1000; i++)
    if (i & 1)
        odd[i]++;
```

```
      r1 = 1
Loop: r2 = r1 & 1
      branch r2, Else
      load r3 = M[odd + r1]
      r3 = r3 + 1
      store M[odd + r1] = r1
Else: r1 = r1 + 1
      r4 = r1 <= 1000
      branch r4, Loop
```

Complex Instruction Set Computing (CISC)

- Instructions perform any number of “atomic” tasks.
- Variable length instructions.
  - Think of Huffman compression coding.
  - Frequent sequences of atomic tasks can be done in a single compact instruction.
  - Instruction alignment issues - Branching (BTB) issues.
- More compact program representation than RISC.
  (Witness the size of x86 binaries compared to DEC Alpha on Linux.)
- Complex decoder
  - A couple of stages typical on x86 implementations.
  - AMD K7 uses start/stop bits in I-Cache.

Most *successful* companies have had a CISC product.

Intel: x86

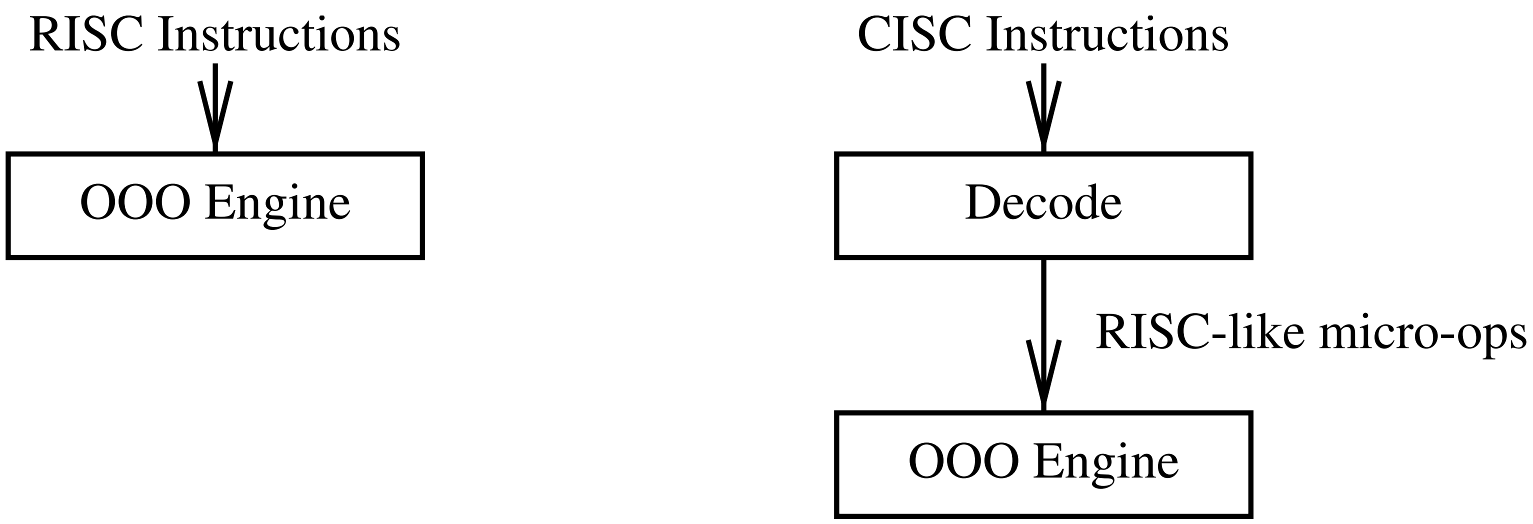
Complex Instruction Set Computing (CISC)

```
      r1 = 1
Loop: r2 = r1 & 1
      branch r2, Else
      load r3 = M[odd + r1]
      r3 = r3 + 1
      store M[odd + r1] = r1
Else: r1 = r1 + 1
      r4 = r1 <= 1000
      branch r4, Loop
```

```
      r1 = 1
Loop: r2 = r1 & 1
      branch r2 <> 0, Else
      incr M[odd + r1]
Else: r1 = r1 + 1
      branch r1 <= 1000, Loop
```

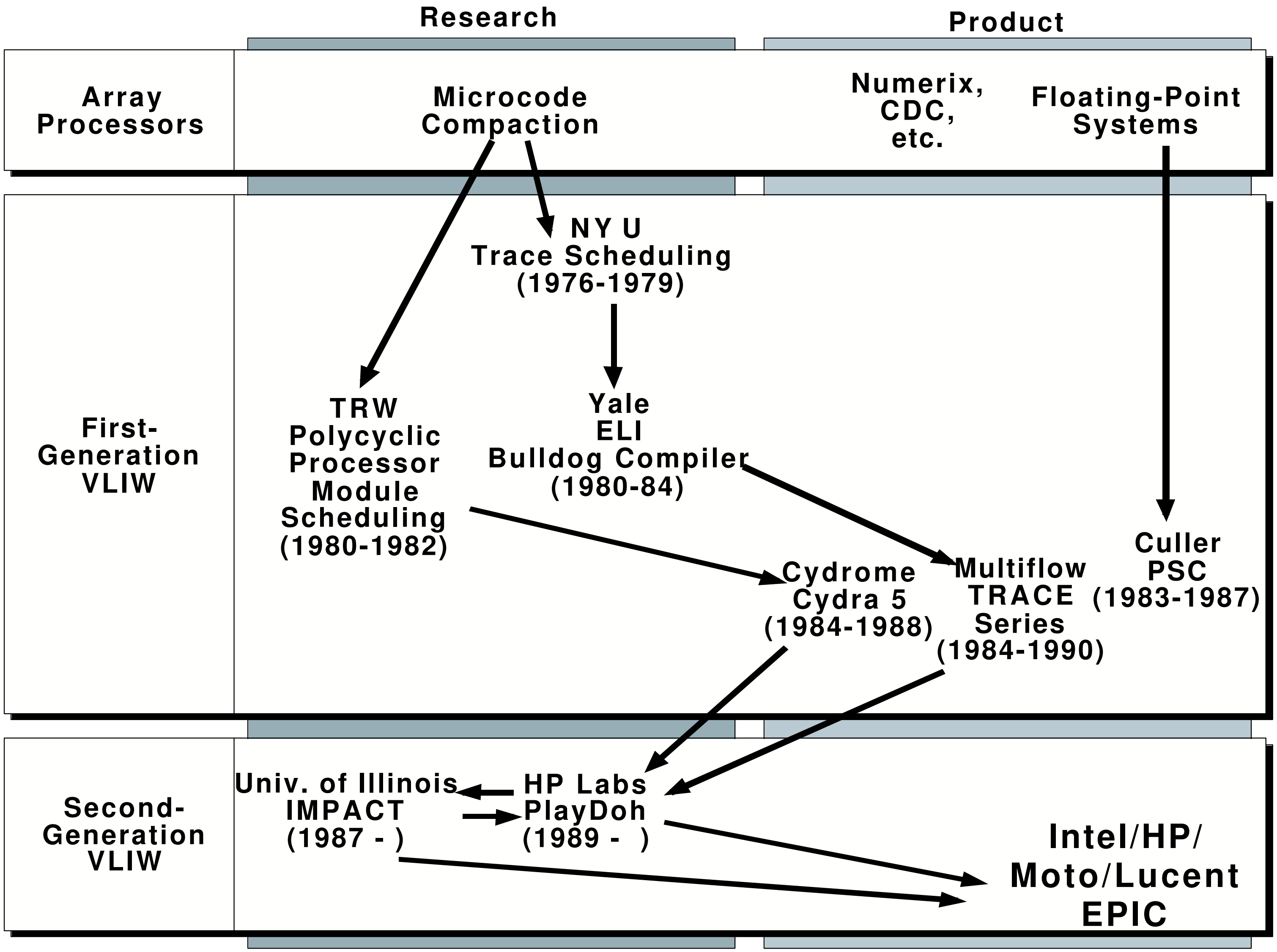
Instruction-Level Parallelism (ILP)

- Pipelining nearly achieved 1 cycle per instruction (CPI).
- To do better, execute more than 1 instruction per cycle (IPC).
- CISC and RISC designers looked to ILP for existing instruction sets.
- OOO Engines - variants of scoreboarding or Tomasulo's Algorithm. ($O(n^2)$)
- CISC and RISC architects did this the same way:



"An Efficient Algorithm for Exploiting Multiple Arithmetic Units" -Tomasulo

Very Long Instruction Word (VLIW)



Very Long Instruction Word (VLIW)

- Finding Instruction-Level Parallelism (ILP) is requires complex analysis and optimization.
- Therefore, finding ILP should be the compiler's responsibility.

VLIW Attributes:

- *Instructions* contain multiple independent *Operations*.

Instruction:

Operation1	Operation2	Operation3	Operation4

- Architecture is exposed:
  - Operation latencies
    add = 1 cycle, sub = 1 cycle, load = 2 cycles, mul = 4 cycles, div = 12 cycles
  - Resources (functional units, read/write register ports)

Instruction:

Integer	Integer	Floating Point	Branch

Very Long Instruction Word (VLIW)

Simple hardware

- No runtime scheduling decisions
- No runtime resource allocation
- No runtime dependence checks
  - Independence check for simultaneous operations
  - Flow dependence check for previously issued operations

Result

- Short pipeline
- Fast cycle time

Unit Assumed Latency (UAL)

- Conventional sequential programs are Unit Assumed Latency (UAL).
- Program can be understood by assuming each instruction is completed before the next one is issued.

$$
\begin{aligned}
r1 &= r2 + r3 \\
r1 &= r4 + r5 \\
r6 &= r1
\end{aligned}
$$

- RISC and CISC processors are UAL.

VLIW can also be NUAL - Non-Unit Assumed Latency...

Non-Unit Assumed Latency (NUAL)

A program is NUAL if:

- At least one operation has non-unit assumed latency L.
- L is greater than 1.
- Program is only understood if the next L-1 instructions are considered to have been issued before this operation completes.

Non-Unit Assumed Latency (NUAL)

Instruction	Operation
1	r1 = load(r2)
2	r1 = load(r3)
3	
4	
5	
6	
7	
8	
9	
10	
11	r4 = fmul(r1, r5)
12	r4 = fadd(r1, r6)
13	
14	r7 = fmul(r4, r9)
15	r7 = fadd(r4, r8)

Non-Unit Assumed Latency (NUAL)

Instruction	Phase1 Operation	Phase2 Operation
1	v1 = load(r2)	
2	v2 = load(r3)	
3		
4		
5		
6		
7		
8		
9		
10		r1 = v1
11	v3 = fmul(r1, r5)	r1 = v2
12	v4 = fadd(r1, r6)	
13		r4 = v4
14	v5 = fmul(r4, r9)	r4 = v3
15	v6 = fadd(r4, r8)	
16		r7 = v6
17		r7= v5

NUAL Scheduling Models

Equals (EQ) Model

- Each operation takes *exactly* its specified latency.
- Destination register will be written L cycles after execution.
- Uses fewer registers.
- Produces tighter schedules.

Less-Than-Or-Equals (LEQ) Model

- Each operation takes *up to* its specified latency.
- Destination register will be written 1 to L cycles after execution.
- Simplifies implementation of precise interrupts.
- Provides binary compatibility when latencies are reduced.

Very Long Instruction Word (VLIW)

Cons

- Poor tolerance of latencies which are not statically determinable - schedule is only tuned for one dynamic outcome.
- Poor object code compatibility
  - Changes in functional unit mix
  - Changes in operation latencies
- Exposing parallelism to VLIW requires code growth.
  - I-Cache performance
  - Memory and Media costs
- Compilers are unable to expose this level of parallelism.

Code Size

```
Loop: r3 = M[odd + r1]
      r3 = r3 + 1
      M[odd + r1] = r3
      r1 = r1 + 1
      r4 = r1 <= 1000
      branch r4, Loop
```

r3 = M[odd + r1]	r4 = r1 < 1000		
r3 = r3 + 1			
M[odd + r1] = r3	r1 = r1 + 1	branch r4, Loop	

r3 = M[odd + r1]	r11 = r1 + 1		r4 = r1 < 1000
r3 = r3 + 1	r13 = M[odd + r11]		
M[odd + r1] = r1	r13 = r13 + 1		
	M[odd + r11] = r13	r1 = r1 + 2	branch r4, Loop

Problem Summary

RISC

- Requires HW to do all parallelization.
- HW scheduling, bypass, interlocking are all $O(n^2)$.
- With n roughly 32, sustained IPC < 2 on 4 issue machine.
- Code size large relative to CISC.

CISC

- Variable length instructions complicate decode and limit issue rates.
- Core has become RISC-ish (with same problems).

VLIW

- Code size associated with NOPs and ILP Optimizations.
- Exposed latency creates portability problem.

The EPIC Solution...

Explicitly Parallel Instruction Computing (EPIC)

Intel IA-64 product called Itanium - “production in mid-2000”

EPIC addresses problems associated with CISC/RISC/VLIW with:

- Bundling/LEQ NUAL
- Rotating Registers
- Predication
- Control Speculation
- Data Speculation

EPIC - Bundling

- Hybrid of VLIW/CISC/RISC
- 128-bit sized and aligned containers called **bundles**.
- Three 41-bit operations and a 5-bit template field.
- Template conveys independence information to reduce code size due to NOPs. (VLIW)
- Compiler determines template to express ILP (CISC/RISC).
- Template conveys operation type information to simplify decode and issue logic. (CISC)
- Template can be: mii, mii], mi]i, mi]i], mlx, mlx], mmi, mmi], m]mi, m]mi], mfi, mfi] mmf, mmf], mib, mib], mbb, mbb], bbb, bbb], mmb, mmb], mfb, or mfb]
- LEQ NUAL for binary compatibility.
- Dynamic scheduling of bundles?

EPIC - Rotating Registers

- Modulo scheduling is analogous to HW pipelining, next iteration starts before current completes.
- Unrolling and renaming unnecessary since rotating registers provide each iteration with own set of registers.
- Registers rotate after each iteration.
- Code size reduction (VLIW). Expression of ILP (CISC/RISC).

```
Loop: r3 = M[odd + r1]
      r3 = r3 + 1
      M[odd + r1] = r3
      r1 = r1 + 1
      branch r1 < 10, Loop
```

r3 = M[odd + r1]			
r3 = r3 + 1			
M[odd + r1] = r3	r1 = r1 + 1	branch r1 < 9, Loop	

r3,1 = M[odd + r1,1]	r3,2 = r3,2 + 1	M[odd + r1] = r3,3	r1 = r1 + 1	branch r1 < 9, Loop

EPIC - Predication

- Removal of conditional branches
- Creates large straight line code regions for exposing ILP (VLIW)
- Reduces branch misprediction cost. (CISC/RISC)
- Trade-off between issue resources and branch penalty (mispred, resource, dependence).

```
if (r1 >= r2)
   r3 = r3 + 1
if (r4 >= r5)
   r6 = r6 + 1
```

```
        p1 = r1 < r2
        branch p1, NEXT1
        r3 = r3 + 1
NEXT1:  p2 = r4 < r5
        branch p2, NEXT2
        r6 = r6 + 1
NEXT2:
```

	p1 = r1 < r2	p2 = r4 < r5
	branch p1, NEXT1	
	r3 = r3 + 1	
NEXT1:	branch p2, NEXT2	
	r6 = r6 + 1	
NEXT2:		

p1 = r1 < r2	p2 = r4 < r5
r3 = r3 + 1 if p1	r6 = r6 + 1 if p2

EPIC - Control and Data Speculation

- Allows compiler transformation based on statically unresolvable dependences.
- Control and Data Speculation is not done as well in dynamically scheduled machines. (CISC/RISC)
  - Dynamic speculation has limited scope. ($O(n^2)$)
  - Dynamic speculation cannot perform optimizations exposed by speculation.
- Allows for more consistent level of ILP. (VLIW)
- Like HW speculation, cost for incorrect assumption may be high.
- In EPIC speculation cleanup is the responsibility of the compiler instead of complex HW logic.

EPIC - Control and Data Speculation

```
if (r1 > 0)
   r2 = M[r1]
if (r3 > 0)
   r4 = M[r3]
```

```
        p1 = r1 < 1
        branch p1, NEXT1
        r2 = M[r1]
NEXT1:  p2 = r3 < 1
        branch p2, NEXT2
        r4 = M[r3]
NEXT2:
```

	p1 = r1 < 1	p2 = r3 < 1
	branch p1, NEXT1	
	r2 = M[r1]	
NEXT1:	branch p1, NEXT2	
	r4 = M[r3]	
NEXT2:		

	r2 = M[r1]	r4 = M[r3]
	p1 = r1 < 1	p2 = r3 < 1
	branch p1, NEXT1	
NEXT1:	branch p2, NEXT2	
NEXT2:		

Remainder of Course

Lect.	Date	Topic
6	10/11	EPIC Overview
7	10/13	Instruction Scheduling - dynamic scheduling
8	10/18	Instruction Scheduling - acyclic scheduling, heuristics
9	10/20	Control Speculation - sentinel scheduling
11	10/25	Control Speculation - trace scheduling, compensation code
12	10/27	Control Speculation - superblock, treegions
		FALL BREAK (10/30 to 11/7)
13	11/8	TBA (Project/Observation Review)
14	11/10	TBA
15	11/15	TBA
16	11/17	TBA
17	11/22	Predication - compilation, if-conversion, hyperblocks
18	11/24	Predication - predicate analysis, program decision logic optimization
19	11/29	Predication - predicate-aware dataflow analysis
		THANKSGIVING BREAK (11/25 to 11/28)
20	12/1	Predication - partial reverse if-conversion framework
21	12/6	Memory Dependence Analysis - pointers, sync-arcs, sync-vars
22	12/8	Value Speculation - memory dependence speculation
23	12/13	Value Speculation - value profiling and speculation (guest lecture)
24	12/15	Course Summary - research opportunities