Building a Better Beast:

Native vs. RISC-like vs. VLIW Methods of Implementing x86 Microprocessors

Cyrix Corporation

Cyrix

x86 Microprocessors

	Native	RISC-like	VLIW
5th Gen.	5x86 *P5*		
6th Gen.	M1	*P6*	
7th Gen.	M?		*P7*

Cyrix

Native vs. RISC-like: Statistics

	M1	P6
Design Style:	Native	RISC-like
L1 Cache Size:	16KB	16KB
Die Size (mm^2):	225	300
Frequency (MHz):	133	133
Performance* (vs. P5):	1.3x	1.0x (?)
Production Ship:	3Q95	?

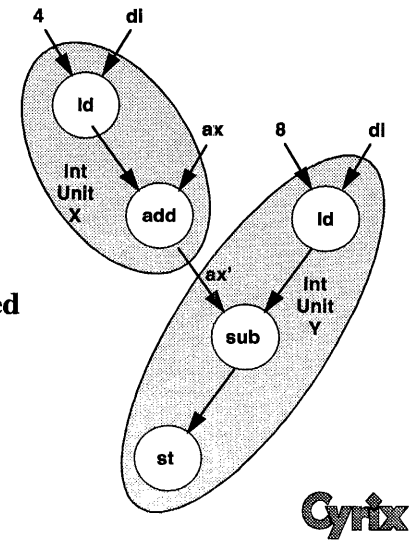
* when running under Win95

Cyrix

Executing Simple x86 Instructions

Native Method of Implementation

ex: 1. *add* ax, 4[di]
2. *sub* 8[di], ax



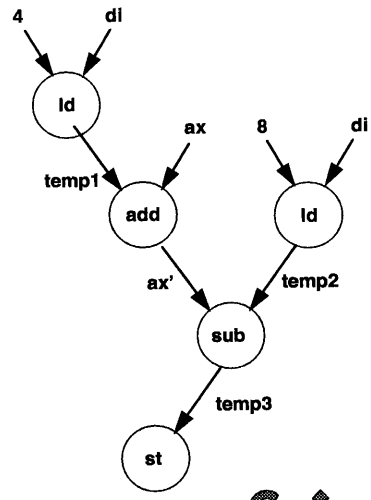
All operations required to complete each instruction are performed in tightly coupled "macro" functional units.

Cyrix

Executing a Simple x86 Instruction

RISC-like Method of Implementation

ex: 1. *add* ax, 4[di]
2. *sub* 8[di], ax



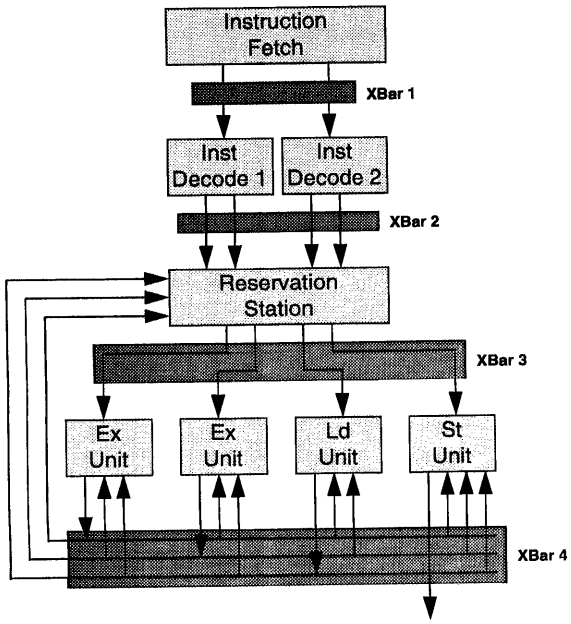
x86 ISA instructions are decomposed into the following sequence of RISC-like micro-op's:

1. load temp1, 4[di]
2. add ax', ax, temp1 ← X86 Instruction Boundary
3. load temp2, 8[di]
4. sub temp3, temp2, ax'
5. store temp3

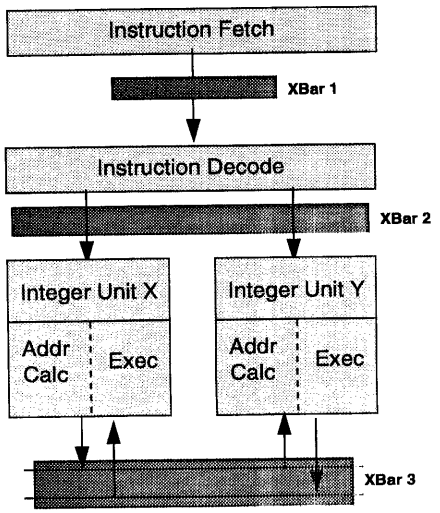
Cyrix

Examples of Today's Implementations

RISC-like



Native



Cyrix

What About the Future?

- As an example, extend the RISC-like and Native methods of implementation to a hypothetical 8 way x86 superscalar.
  - Native implementation would have 8 autonomous processing units, each with its own AC and EX function units for a total of 16 units.

    XBar 1 and XBar3 would be of degree 8x8. XBar2 is optional, as it is only required for load balancing.

    *Total connections[Native] = 2(8 x 8) + 8 = 136**
  - RISC-like implementation could probably get away with only 6 AC's (4 for loads and 2 for stores) and 6 EX's, for a total of only 12 units.

    XBar 1 is also of degree 8x8, but Xbar's 2, 3, and 4 all are more than twice as large, at 12x12.

    *Total connections[RISC-like] = (8 x 8) + 3(12 x 12) = 496**

Cyrix

*note that each of these connections represents ~40 bit bus.

The advantage of the Native over the RISC-like implementations become *even more pronounced* as the degree of parallelism increases.

Cyrix

The Advantages of Native

- Native implementations are optimized for the semantics of complex x86 instructions which are commonly used in existing binary code, and so minimize the amount of communication required on-chip.
- The Native approach to designing x86 microprocessors is symmetric, and thus yields inherently balanced designs which are less prone to performance bottlenecks.
- The symmetry of Native implementations creates opportunities for design reuse, thus reducing design cycle times.

Cyrix

What about VLIW?

- VLIW is a solution in search of a problem.
- Translation to traditional VLIW operations will be inefficient, requiring many op's per x86 instruction.
- Allowing the use of more complex VLIW op's would also require more complicated function units, thus defeating one of the primary goals of VLIW architectures.
- Translation from x86 instructions into VLIW does not address the basic problem microprocessor design:

  Instruction Issue Bandwidth

Cyrix

Performance Model

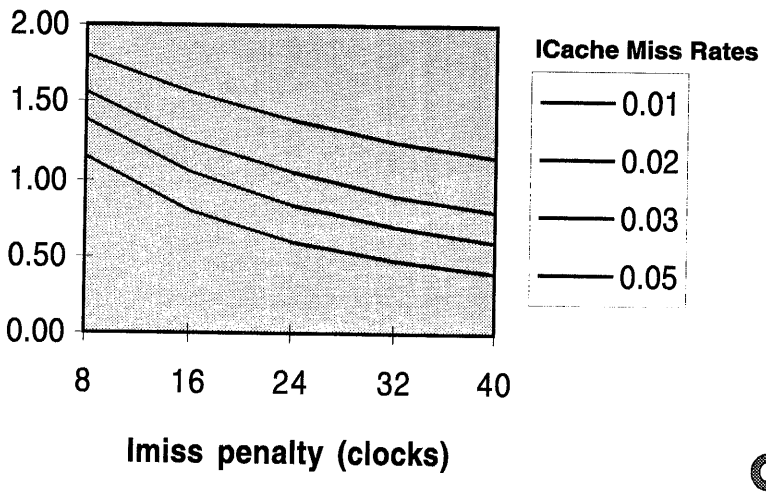
$$IPC = \frac{1}{1/I + BP + IC}$$

- **I = sustained issue rate**
- **BP = average penalty per instruction for branch mispredictions, f(BTB,BPD)**
- **IC = average penalty per instruction for Instruction Cache misses**

Cyrix

IPC for a 4-way Machine

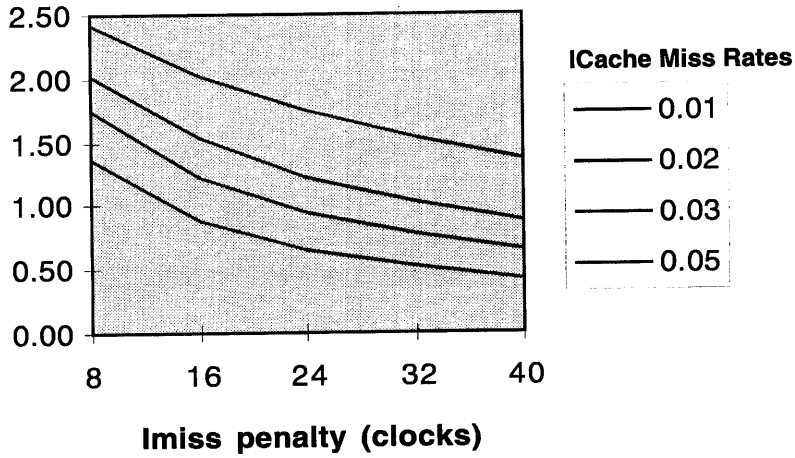
- **I=3.5 BTB=(95%,8) BPD=(93%,12)**



Cyrix

IPC for an 8-way Machine

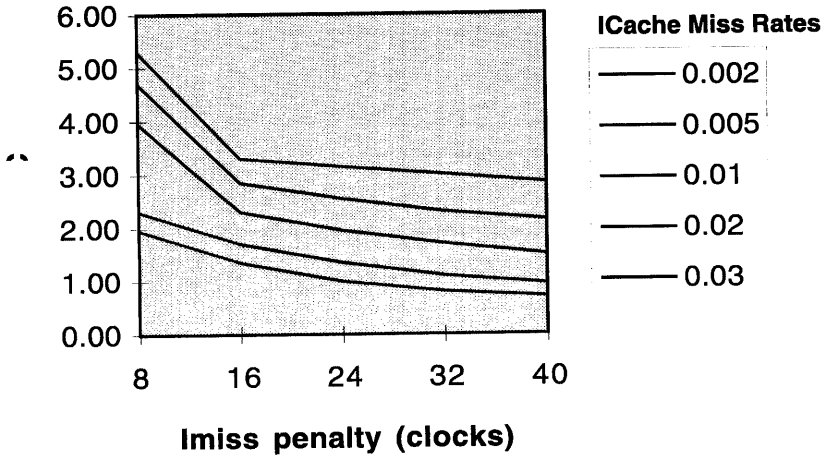
- I=7 BTB=(95%,8) BPD=(93%,12)



Cyrix

IPC for a REALLY Good 8-way Machine

- I=7 BTB=(99%,3) BPD=(97%,6)



Cyrix

2.3-15

Is VLIW the Answer?

- VLIW is not compatible with existing x86 software!
- VLIW machines will be hard to build:
  - They will require very large caches and/or fast main memory
  - They will require extensive compiler support:
    - instruction/data prefetching
    - instruction target registers
    - superblock compilation
    - guarded execution (= predicated execution)
- VLIW does not address the key issues:
  - Improving instruction cache hit rates and/or instruction memory bandwidth.
  - Improving branch direction and target prediction and/or reduce misprediction penalties.

Cyrix

2.3-16

How to Build a Better Beast...

- There is plenty of headroom for improving the performance of x86 compatible microprocessors by improving the valid instruction issue rate.
- The greatest strength of the x86 ISA lies in its installed code base. It is, therefore, fundamentally important to design microprocessors which not only remain compatible with the letter of the x86 architecture, but are optimized for the characteristics of current software.

Cyrix's Native implementations embody these ideas.

Cyrix

Result:

Faster

Simpler

x86 Compatible

Higher Performance

Microprocessors

Cyrix