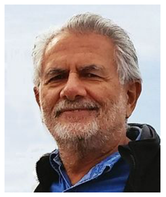
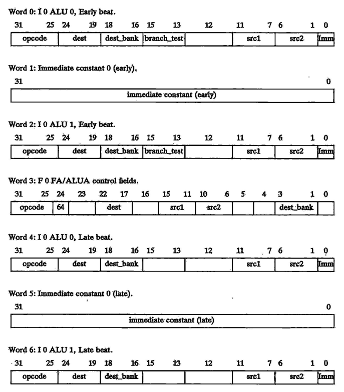
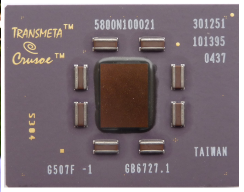
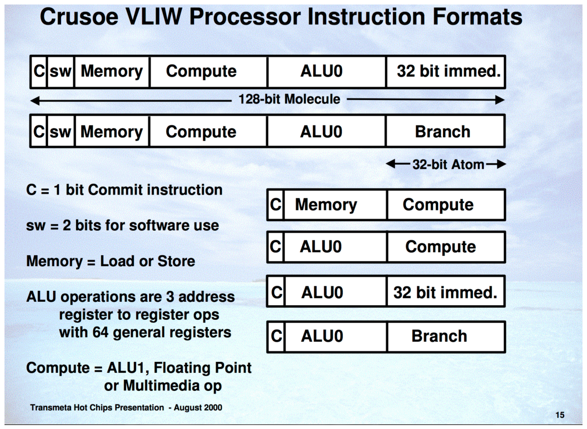
VLIW & ELI-512


Josh Fisher

Trace 7/200







Crusoe VLIW Processor Instruction Formats

C
sw
Memory
Compute
ALU0
32 bit immed.
128-bit Molecule
C
sw
Memory
Compute
ALU0
Branch
32-bit Atom
C
Memory
Compute
C
ALU0
Compute
C
ALU0
32 bit immed.
C
ALU0
Branch

C = 1 bit Commit instruction

sw = 2 bits for software use

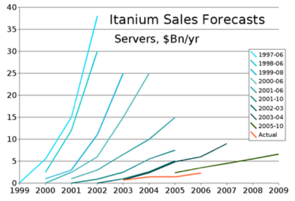
Memory = Load or Store

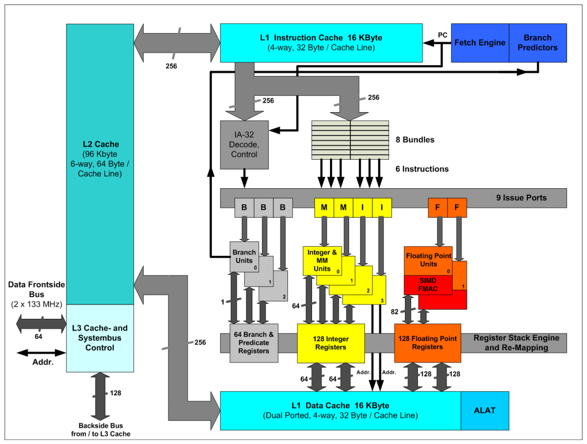
ALU operations are 3 address register to register ops with 64 general registers

Compute = ALU1, Floating Point or Multimedia op

Transmeta Hot Chips Presentation - August 2000

15




L1 Instruction Cache 16 KByte (4-way, 32 Byte / Cache Line)
PC
Fetch Engine
Branch Predictors
256
L2 Cache (96 Kbyte 6-way, 64 Byte / Cache Line)
IA-32 Decode, Control
8 Bundles
6 Instructions
9 Issue Ports
B B B
M M I I
F F
Branch Units
Integer & MM Units
Floating Point Units
SIMD FMAC
Data Frontside Bus (2 x 133 MHz)
L3 Cache- and Systembus Control
64 Branch & Predicate Registers
128 Integer Registers
128 Floating Point Registers
Register Stack Engine and Re-Mapping
Addr.
128
Backside Bus from / to L3 Cache
L1 Data Cache 16 KByte (Dual Ported, 4-way, 32 Byte / Cache Line)
ALAT

[Processor table](#)

...it's Intel Itanium processors

Why VLIW?

- To utilize ILP
  - in a simple HW design
- Good for scientific computing & signal processing, crypto

Why not VLIW?

- *if you can build* a wide issue Tomasulo's algorithm (aka "Super Scalar") processor, then *it will be faster* than the same width VLIW processor.
- there just isn't enough provably statically available ILP
- Code compatibility

How to handle branches?

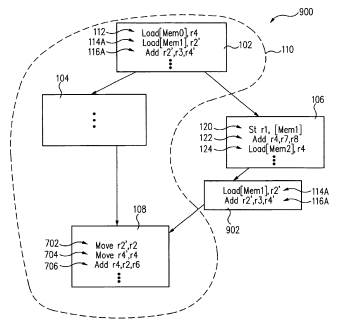
- [ALU][ALU][ALU][BLEZ][BGTZ][BEQZ]
- Option 1: don't do that: [ALU][ALU][ALU][BLEZ]
- Option 2: assign precedence

Exceptions?

- [ADD r1 + r3 -> r3][LOAD @(r5) -> r6][DIV]
  - Allow instructions that don't fault to complete
    - OS has to fix the code
    - mask off instructions that have completed on restart
  - Throw out all results on completion
    - Potential for live-lock

What about memory ordering?

- [STORE r1 -> @(r2)][LOAD @(r3) -> r4] ;;; r2 = r3
  - result is value of r1 goes into r4
  - the previous value in memory goes into r4
  - undefined
- [STORE r1 -> @(r2)][STORE r3 -> @(r2)]
  - undefined
  - precedence
- [STORE r1 -> @(r2)][STORE r3 -> @(r4)] ;;; r2 != r4
  - only allow 1 store
  - precedence



Pros/Cons of binary translation

- Perhaps not as fast as having the source
  - but debatable