VLIW & ELI-512
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?

- \([\text{ADD } r1 + r3 \rightarrow r3][\text{LOAD } @(r5) \rightarrow r6][\text{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?

- \([\text{STORE } r1 \rightarrow @(r2)][\text{LOAD } @(r3) \rightarrow r4] \;;
  r2 = r3\]
  - result is value of \(r1\) goes into \(r4\)
  - the previous value in memory goes into \(r4\)
  - undefined
- \([\text{STORE } r1 \rightarrow @(r2)][\text{STORE } r3 \rightarrow @(r2)]\]
  - undefined
  - precedence
- \([\text{STORE } r1 \rightarrow @(r2)][\text{STORE } c3 \rightarrow @(r4)] \;;
  c3 \neq r4\]
  - only allow 1 store
  - precedence

Pros/Cons of binary translation

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