Cryptomaniac
A Cautionary Tale

Don’t Let This Happen to You!
AES Selection Process

• Started in 1997
• 3 years
• 15 proposals (CAST-256, CRYPTON, DEAL, DFC, E2, FROG, HPC, LOKI97, MAGENTA, MARS, RC6, Rijndael, SAFER+, Serpent, and Twofish)

• Criteria
  • Security
  • Performance (HW, SW, limited memory, etc.)
– 5 finalists (MARS, RC6, Rijndael, Serpent, and Twofish).
– Rijndael won.
Current Web Statistics (Just out of curiosity)

- Web objects are now 7.3KB on average (down from 20KB) (Why?)
- ~42-44 objects/page; 312KB/page
  - 184KB of images
  - 65KB of javascript
  - 27KB of style sheets
  - 36KB of “other”
- For SSL sites: 263KB/page
Architecture in practice!

Figure 3. Performance Comparison between Alpha 21264 and Dataflow Machine.
Intel’s AES Instructions

Non-AES performance

![Non-AES performance chart](chart1)

AES performance

![AES performance chart](chart2)

Adjusting for underlying CPU performance, it’s 3.4x improvement.
VLIW
Very Long Instruction Word (VLIW)

- Put two (or more) instructions in one!

Each sub-instruction is just like a normal instruction.

The instructions execute **at the same time**.

The processor can treat them as a single unit.

Typical VLIW widths are 2-4 instructions, but some machines have been much higher.
VLIW Example

- **VLIW-MIPS**
  - Two MIPS instruction/VLIW instruction word
  - Not a real VLIW ISA.

MIPS Code:
```mips
ori $s2, $zero, 6
ori $s3, $zero, 4
add $s2, $s2, $s3
sub $s4, $s2, $s3
```

Results:
$s2 = 10$
$s4 = 6$

Since the add and sub execute *sequentially*, the sub sees the new value for $s2$

VLIW-MIPS Code:
```mips
<ori $s2, $zero, 6; ori $s3, $zero, 4>
<add $s2, $s2, $s3; sub $s4, $s2, $s3>
```

Results:
$s2 = 10$
$s4 = 2$

Since the add and sub execute *at the same time* they both see the original value of $s2$
VLIW Challenges

- VLIW has been around for a long time, but it’s not seen mainstream success.
- The main challenge is finding instructions to fill the VLIW slots.
- This is tortuous by hand, and difficult for the compiler.

VLIW-MIPS Code

```
<ori $s2, $zero, 6;  ori $s3, $zero, 4>
<add $s2, $s2, $s3; nop              >
<sub $s4, $s2, $s3; nop              >
```

Results:
$s2 = 10
$s4 = 6

Now, the add and sub execute sequentially, but we've wasted space and resources executing nops.
VLIW’s History

• VLIW has been around for a long time
  • It’s the simplest way to get CPI < 1.
  • The ISA specifies the parallelism, the hardware can be very simple
  • When hardware was expensive, this seemed like a good idea.

• However, the compiler problem (previous slide) is extremely hard.
  • There end up being lots of noops in the long instruction words.
  • Especially for “branchy” code (word processors, compilers, games, etc.)

• As a result, they have either
  • 1. met with limited success as general purpose machines (many companies) or,
  • 2. Become very complicated in new and interesting ways (for instance, by providing special registers and instructions to eliminate branches), or
  • 3. Both 1 and 2 -- See the Itanium from intel.
VLIW’s Success Stories

• VLIW’s main success is in digital signal processing
  • DSP applications mostly comprise very regular loops
    • Constant loop bounds,
    • Simple data access patterns
    • Non-data-dependent computation
  • Since these kinds of loops make up almost all (i.e., $x$ is almost 1.0) of the applications, Amdahl’s Laws says writing the code by hand is worthwhile.

• These applications are cost and power sensitive
  • VLIW processors are simple
    • Simple means small, cheap, and efficient.

• I would not be surprised if there are several VLIW processors in your cell phone.
“Pareto-optimal” designs are those for which no other design is better by both metrics.