The IA-64 Architecture

Bruce Jacob

Department of Electrical Engineering
University of Maryland at College Park

OUTLINE:

- **Architecture overview**
- **Background**
- **Architecture Specifics**

THE IA-64
ARCHITECTURE

Bruce Jacob

University of
Maryland

Salient Points

128 Registers (1KB cache)

- **Eliminates renaming**
- **Reduces cache accesses**

Speculative Loads

- **Moves loads past control barriers**

Predicated Execution

- **Eliminates 40% of conditional branches**

First time we’ve seen all these in one place

Goals of Architecture

Overcome performance limiters:

- **Branches**
- **Memory latency**
- **Sequential program model**

Long Architecture Lifetime

- **Large register file**
- **Fully interlocked architecture**
- **No fixed issue-width**

Retain Backward Compatibility with x86

Background

In the Beginning ... CISC

RISC -> larger register files, simpler design

- **(better IC technologies)**

VLIW -> possible to do lots in parallel

- **Influenced superscalar design, dynamic scheduling**

Superscalar Design:

- **Dependency-checking, I-dispatch, register renaming, out-of-order**
- **A lot of hardware required**

Performance Limiters: Branches

20-30% Execution Time on Today's Processors: MISPREDICTS

4-issue, 8-stage pipe, BR resolved stage 7: 24 issue slots squashed per mispredict

Mispredicts:

- **95% prediction accuracy**
- **Branches = 1/6 instrs**

$$\frac{\text{24 stalls}}{\text{120 instr}}$$

ALSO: break program into "basic blocks" Severely limits opportunities for parallelism

THE IA-64 ARCHITECTURE

Bruce Jacob

University of Maryland

Performance Limiters: Memory Latency

Processor Speed/Performance = 60% per yr
Memory Technology = 5% per yr

Not Just Memory => Cache Technology:

- **Today's caches often pipelined**
- **2+ cycle latency (assumes AGI)**

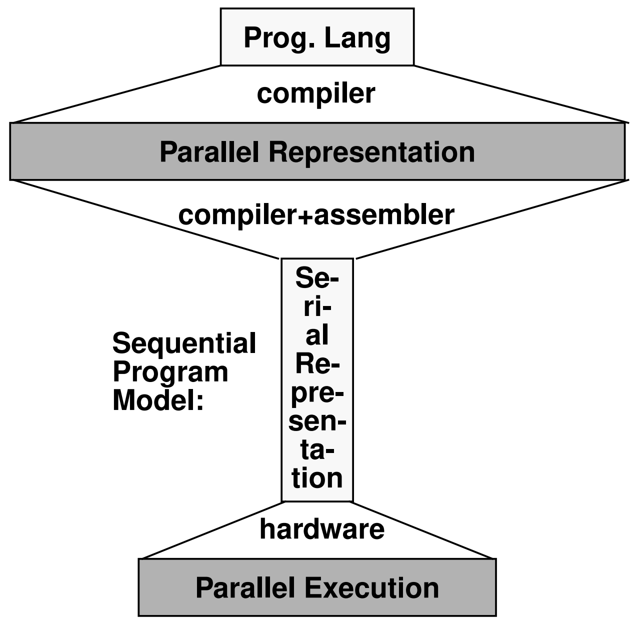
load use	=>	load STALL use

COMPOUNDED BY ISSUE WIDTH:

- **STALL *= issue-width**

Performance Limiters: Sequential Program Model

Today:



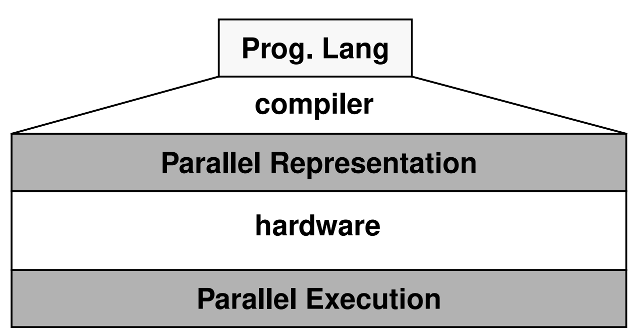
THE IA-64 ARCHITECTURE
Bruce Jacob
University of Maryland

Performance Limiters: Sequential Program Model

Sequential model undoes much of the compiler's work—we spend HUGE amounts of die area to recover it

Allows "artificial sequentiality" to creep in

Goal:



Obvious Answer: VLIW

(Very Long Instruction Word)

Where did VLIW fail that Intel/HP can win?

- **Code expansion**

A	B	NOP	NOP

- **Binary compatibility**

A	B	C	D

vs:

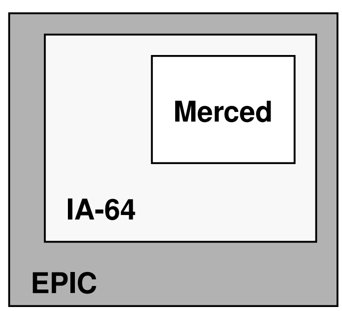
A	B

- **Cannot exploit full parallelism:**
  Control instructions serialize execution
  Cannot move loads above branches

THE IA-64 ARCHITECTURE
Bruce Jacob
University of Maryland

Intel's Solution: EPIC

(Explicitly Parallel Instruction Computing)



PREDICATED EXECUTION
eliminates if-then-else

SPECULATIVE LOADS
allow crossing control

LARGE REGISTER FILE
enables prefetches,
reduces cache misses

VARIABLE INSTRUCTION WIDTH
never need to insert NOP instructions

Variable Instruction: Bundle

128 … 0

Instr2	Instr1	Instr0	Template

Instructions:

- **Opcode**
- **Predicate register (6)**
- **Source1 & Source2 (7 each)**
- **Dest (7)**

Template (8?):

- **Instruction grouping**
  (can do in 3 bits: I0, I1, I2 paired w/ prev)
- **Prefetch hints?**

THE IA-64 ARCHITECTURE
Bruce Jacob
University of Maryland

Predicated Execution

Jerry Huck on instruction-level parallelism:

"Just because you allow it doesn't mean you're going to *get* any of it."

BRANCHES LIMIT ILP:

Sequential, no-predict: normal bank teller

Sequential, predict: fill out slip in advance (predict whether deposit or withdrawal)

Predicated Execution: fill out both slips, throw away whichever is wrong

Predication Example

```
IF      i1
        i2
        BEQ a b label1

THEN    i3
        i4
        JUMP label2

ELSE    label 1:   i5
                   i6

        label2:    i7
                   i8
```

Becomes:

```
                      i1
                      i2
   p1, p2  <-   (a == b)
(p1)                  i3
(p1)                  i4
(p2)                  i5
(p2)                  i6
                      i7
                      i8
```

64 1-bit predicate registers

Speculative Loads

```
              i1
              i2
  BEQ a b label1
----------------
            load
             use

label 1:      i3
              i4
```

```
          load.S
              i1
              i2
  BEQ a b label1

           chk.S
             use

label 1:      i3
              i4
```

Why the explicit check?
Not explained — “a good reason”

Long Architecture Life

Large Register File

- Like jump from 8 -> 32 in 70's -> 80's

Fully Interlocked

- Not tied to a particular implementation

VLIW w/ Variable Instruction Width

- Not tied to particular implementation, Works well for "shovels" of different widths