Chapter 5b: ILP, cont.

Superscalar

Theory of ILP

IBM RS/6000

CRISP

Superscalar

Two limits on Pipelining

- Latch overheads & signal skew
- Unpipelined instruction issue logic (Flynn limit: CPI ≥ 1)

Superscalar: Parallelism in instruction logic

```
      1   2   3   4   5   6   7   8   9
i    IF  ID  EX  MEM WB
i+1  IF  ID  EX  MEM WB
i+2      IF  ID  EX  MEM WB
i+3      IF  ID  EX  MEM WB
i+4          IF  ID  EX  MEM WB
i+5          IF  ID  EX  MEM WB
i+6              IF  ID  EX  MEM WB
i+7              IF  ID  EX  MEM WB
i+8                  IF  ID  EX  MEM WB
i+9                  IF  ID  EX  MEM WB
```


Superscalar, cont.

IF Parallel access to I-cache
Require alignment?

ID Replicate logic
Fixed-length instructions?
HANDLE INTRA-CYCLE HAZARDS
Dynamic techniques?

EX Parallel/pipelined
(as before)

MEM > 1 per cycle?
If so, hazards & multi-ported D-cache

WB Different register files?
Multi-ported register files?

Progression

- Integer + floating-point
- Any two instructions
- Any four instructions
- Any ***n*** instructions?

Superscalar DLX

Assume two instructions per cycle

- One integer, load/store, or branch
- One floating point

Could require 64-bit alignment and ordering of instruction pair.

```
I  F        I  F        F  I
I  F        F  I        F  I

OK          NOT         NOT
            OK          OK
```

Best case

- CPI = 0.5
- But ....

Superscalar DLX, cont.

Hazards are a big problem

Loads

- Latency is 1 cycle
- Was 1 instruction
- NOW 3 instructions

Branches

- NOW 3 instructions

Floating point loads and stores

- May cause structural hazards
- Additional ports?
- Additional stalls?

Superscalar, cont.

Myths

- Superscalar precludes pipelining
- CPI < 1 $\Rightarrow$ high-performance (e.g., SPECmarks/cycle)
- Instruction-level parallelism required is superscalar degree $\times$ pipeline length

Truths

- All superscalar CPUs are pipelined
- Minimize program execution time = path length $\times$ CPI $\times$ cycle time
- Full speed with superscalar degree $\times$ operation latency

P.S.

- Superpipelining = Pipelining

Vs. VLIW (Very Long Instruction Word)

Multiple independent operations per instruction

- Like horizontal microcode
- Generalization of vector instructions

Versus Superscalar

- \- Not binary compatible
- \- Code size explosion
- \- Complex compiler
- \- Wasteful of hardware resources
- \+ Issue logic simpler
- \+ Exploit more parallelism

Beyond Superscalar?

- Unlevel playing field in 1980s
- RISC or VLIW on a single chip
- When will superscalar run out of gas?

An ILP Taxonomy [Jouppi & Wall]

Basic pipelining

- issues 1 instrn/cycle
  cycle = operation latency

Superscalar: issues ***n*** instrn/cycle

Superpipelined: cycle < operation latency

VLIW: ***n*** parallel operations/instrn

Vector: ***n*** pipelined, identical operations/instrn

Fundamentals of ILP Processing

Express Parallelism

Programming language issue

Extract parallelism in software

Compiler Issue

Extract parallelism in hardware

Hardware issue

Exploit parallelism

Resource issue

Extract Parallelism

Establish a block of operations to be executed

Establish dependencies between operations in block

Register: Easy

Memory: Hard

Schedule operations: move operations that occur ''later'' in the block ''up'' so that they execute ''earlier'' in time, in parallel with other operations.

Scheduling flexibility limited by accuracy of dependency determination

Exploit Parallelism

Multiple functional units

Interconnections between units

Storage elements to buffer values (registers)

Data memory systems

Instruction memory and supply systems

Establishing Dependencies

Register Dependencies: determine if two instructions access the same register

Requires knowledge of which registers accessed in an arbitrary run of program

Trivial

Memory Dependencies: determine if two instructions access the same memory location

Requires knowledge of which memory locations accessed in an arbitrary run of program

Very hard in general (example, pointers)

Doable for regular data structures, e.g., arrays

Establishing Block of Instructions

Where is the ILP?

Basic Blocks

- Very limited ILP

Beyond Basic Blocks

- Other parts of the program control flow graph (CFG)

Beyond Basic Blocks

(Conditional) branch terminates basic block

- Don't know what code to execute next

Control Parallelism: Execute code from *control independent* block

Speculative Execution: execute code from block that is *not guaranteed to execute*.

Control Parallelism

Control independent block guaranteed to execute

Not many control independent blocks in important parts of several programs

Motions between arbitrary control independent blocks limited because of data dependencies

Speculative Execution

Establish a collection of likely to execute blocks of instructions

- Trace
- Window

Branch prediction used to determine likely path

- Static
- Dynamic

Execute instructions from likely path even though *not guaranteed* to execute

Provide recovery mechanisms in case if exceptions

Note: with guarded or predicated instructions, could execute instructions from multiple paths

ILP Architectures

Architecture is contract between programs for an architecture and set of implementations for architecture

- includes info embedded in program pertaining to parallelism

Sequential Architectures

Dependence Architectures

Independence Architectures:

ILP Architectures

Sequential Architectures

- No info expected from program
- Info reconstructed by hardware at run time
- Hardware makes scheduling decisions

Dependence Architectures

- Dependence relationships conveyed explicitly
- Hardware makes scheduling decisions

Independence Architectures:

- Hardware does nothing
- Compiler conveys everything to hardware

Dataflow Architectures (Dependence)

Use a non-conventional programming paradigm to express maximum amount of parallelism

Each instruction specifies list of dependent instructions

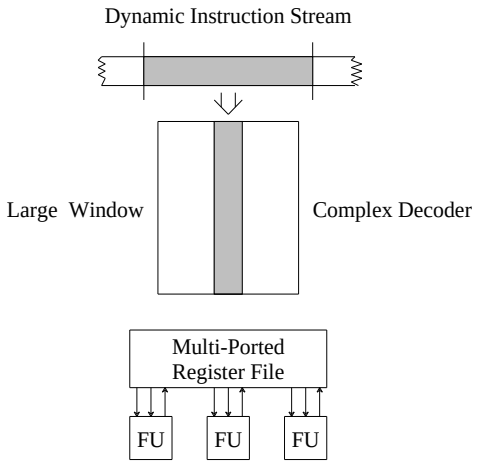
Maximum parallelism extracted

Extraction of parallelism by hardware only

- Considers entire dataflow graph to be a single window Incurs big performance penalty due to making all decisions at run time
  - associative matching hardware
  - "stretching" of communication arcs

Traditionally exploit control parallelism

Superscalar Architectures (Sequential)



Superscalar: Run Time Extraction of ILP

Dynamic creation of window

- Use dynamic branch prediction to go beyond basic blocks
- Windows might be inaccurate since no access to CFG

Dynamic determination of dependencies

- Hardware has to reconstruct information available at compile time

Dynamic instruction motion in time

- Search for ***independent*** instructions in window every cycle
- Affects cycle time

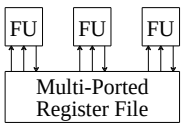
Superscalar Hardware

Hardware checks to see instructions operands interfere with another instruction:

- Already in execution
- Has been issued but is waiting for execution
- Is being issued concurrently

Hardware decides exactly when and on which unit to execute instruction

VLIW Architectures (Independence)



Static creation of window

- As big a window as the entire program

Static determination of dependencies

- Relieves hardware of this burden
- Makes conservative assumptions about run-time information

Static scheduling

VLIW Compilers

Trace is a sequence of likely-to-execute basic blocks

- Static branch prediction used to predict likely blocks

Scheduling: compiler decides which operations from block (or window or trace) to pack into VLIW instruction

Speculative execution done by introducing extra code in case of incorrect prediction

Extra code leaves around enough state to recover in case of incorrect prediction

- Requires extra registers

Local Scheduling: within a block

Use heuristics to pick unscheduled op

- predecessors must have been scheduled

Calculate completion time; select op with largest time to completion

Determine earliest time that appropriate functional unit available; schedule operation

Repeat until done

Global Scheduling: beyond basic blocks

Blocks that are a part of a cycle in CFG

- Part of a loop structure
- Use software pipelining

Blocks from acyclic portions

- Not part of a loop
- Use trace scheduling

Superscalar in the IBM RS/6000

Not just superscalar ...

Reduce path length

- Update addressing mode
- Multiply-add instruction

Reduce CPI

- Conditionals without branches
- SUPERSCALAR
- Register renaming on floating-point loads

Reduce cycle time

- Great technology
- To be improved on single chip

Superscalar in the IBM RS/6000, cont.

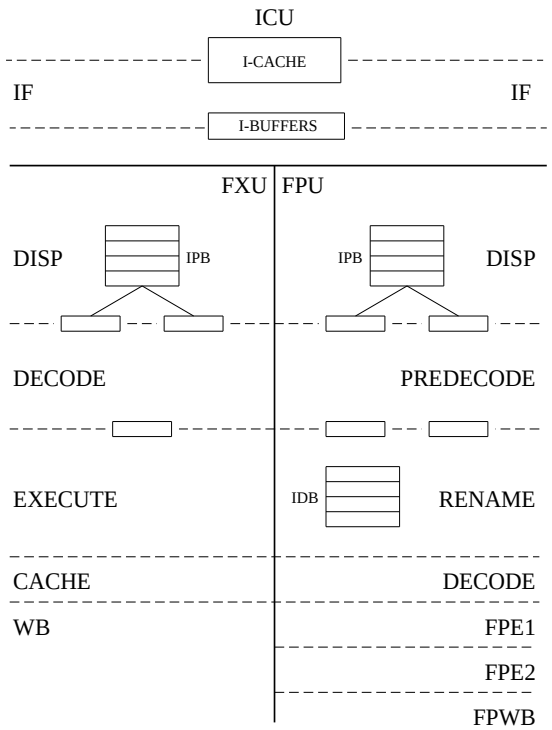
Issues up to four instructions/cycle

- 1 branch
- 1 condition code
- 2 instructions (integer + floating-point)
  - 1 + 1, 0 + 2 or 2 + 0

Basic Organization

- ICU - Instruction Cache Unit
  - (includes branch unit)
- FXU - Fixed-Point Unit (Integer)
  - (all load/stores address calculations)
- FPU - Floating-Point Unit

Superscalar in the IBM RS/6000, cont.



Decoded Instruction Cache (CRISP/Hobbit)

Instrn Unit decodes instructions

- Variable length
- Memory-to-memory call frame relative

Decoded Instruction Cache Entries

- PC is the tag
- Decoded control
- Left & right operand addresses
- Next PC & alternative next PC

Branch Folding

- Unconditional
- Conditional