This Set

11-1

These slides do not give detailed coverage of the material. See class notes and solved problems (last page) for more information.

Text covers multiple-issue machines in Chapter 4, but does not cover most of the topics presented here.

Outline

- Multiple Issue Introduction

- Superscalar Machines

- VLIW Machine

- Sample Problems

11-1

Multiple Issue

11-2

Multiple-Issue Machine:

A processor that can sustain fetch and execution of more than one instruction per cycle.

n-Way Processor:

A multiple issue machine that can sustain execution of n instructions per cycle.

Single-Issue Machine:

A processor that can sustain execution of at most one instruction per cycle. A neologism for the type of processor covered in Chapter 3 and part of Chapter 4.

Sustain Execution of n IPC:

Achieve a CPI of $\frac{1}{n}$ for some code fragment ...

... written by a friendly programmer ...

... to avoid cache misses and otherwise avoid stalls.

11-2

Types of Multiple Issue Machines

Superscalar Processor:

A multiple-issue machine that implements a conventional ISA (such as MIPS and SPARC).

Code need not be recompiled.

General-purpose processors were superscalar starting in early 1990's.

VLIW Processor:

A multiple-issue machine that implements a VLIW ISA ...

... in which simultaneous execution considered. (More later.)

Since VLIW ISAs are novel, code must be re-compiled.

Idea developed in early 1980's, ...

... so far used in special-purpose and stillborn commercial machines, ...

... and is being used in Intel's next generation processor.

Intel's Itanium implements the Itanium (née IA-64) VLIW ISA.

(Name of ISA and implementations are both Itanium.)

n-Way Superscalar Machine Construction

Start with a scalar, a.k.a. single-issue, machine.

Duplicate hardware so that most parts can handle n instructions per cycle.

Don't forget about control and data hazards.

Superscalar Difficulties

Register File

Scalar: 2 reads, 1 write per cycle.

n-way: $2n$ reads, n writes per cycle.

Dependency Checking and Bypass Paths For ALU Instructions

Scalar, about 4 comparisons per cycle.

n-way, about $n(2(2n+n-1) = 6n^2 - 2n$ comparisons.

Loads-Use Stalls

Scalar, only following instruction would have to stall (if dependent).

n-way, up to the next $2n-1$ instructions would have to stall (if dependent).

Superscalar Difficulties

11-6

Instruction Fetch

Memory system may be limited to aligned fetches ...

... for example, if branch target is 0x1114 ...

... instructions starting at 0x1110 may be fetched (and the first ignored) ...

... wasting fetch bandwidth.

11-6

Typical Superscalar Processor Characteristics

11-7

Instruction Fetch

Instructions fetched in *groups*, which must be aligned in some systems.

Unneeded instructions ignored.

Instruction Decode (ID)

Entire group must leave ID before next group (even 1 insn) can enter.

Execution

Not all hardware is duplicated . . .
. . . and therefore some instruction pairs cause stalls.

For example, early processors could simultaneously start one floating-point and one integer instruction . . .
. . . but could not simultaneously start two integer instructions.

VLIW

11-8

Very-Long Instruction Word (VLIW):
An ISA or processor in which instructions are grouped into *bundles* which are designed to be executed as a unit.

Explicitly Parallel Instruction Computing:
Intel's version of VLIW. Here, VLIW includes EPIC.

Key Features

Instructions grouped in bundles.

Bundles carry dependency information.

Can only branch to beginning of a bundle.

11-8

Current Examples

Texas Instruments VelociTI (Implemented in the C6000 Digital Signal Processor).

Intended for signal processors, which are usually embedded in other devices . . .
. . . and do not run general purpose code.

Intel Itanium (née IA-64) ISA (Implemented by Itanium, Itanium 2).

Intended for general purpose use.

VLIW-Related Features

Instructions grouped into 128-bit bundles.

Each bundle includes three 41-bit instructions and five *template bits*.

Template bits specify dependency between instructions and the type of instruction in each slot.

Other Features

128 64-bit General [Purpose Integer] Registers

128 82-bit FP Registers

Many additional special-purpose registers.

Makes extensive use of predication.

Cray Tera MTA implemented by the Tera Computer Company.

(Tera bought by Cray.)

Intended for scientific computing.

VLIW-Related Features

Instructions grouped into 64-bit bundles.

Each bundle holds three instructions.

Restrictions: one load/store, one ALU, and one ALU or branch.

Bundle specifies number of following non-dependent bundles in a *lookahead* field.

Serial bit for specifying intra-bundle dependencies.

Other Features

Radical: Can hold up to 128 threads, does not have data cache.

Ordinary: 32 64-bit registers.

Extra bits on memory words support inter-processor synchronization.

Branches can examine any subset of 4 condition code registers.

VLIW Bundle and Slot Definitions Definitions

Bundle: a.k.a. *packet*

The grouping of instructions and dependency information which is handled as a unit by a VLIW processor.

Slot:

Place (bit positions) within a bundle for an instruction.

A typical VLIW ISA fits three instructions into a 128-bit bundle ...

... such a bundle is said to have three slots.

Example: Itanium (née IA-64)

Bundle Size, 128 bits; holds three instructions.

Slot 2	Slot 1	Slot 0	dep. info
127 87	86 46	45 5	4 0

Instruction Restrictions In Bundles

11-14

ISA may forbid certain instructions in certain slots ...
... *e.g.*, no load/store instruction in Slot 1.

Tera-MTA: Three slots per 64-bit bundle. (Slot 0, Slot 1, Slot 2.)

Slot 0: Load/Store

Slot 1: ALU

Slot 2: ALU or Branch

Itanium (née IA-64): Three slots per 128-bit bundle.

Slot 0: Integer, memory or branch.

Slot 1: Any instruction

Slot 2: Any instruction that doesn't access memory.

There are further restrictions.

Dependency Information in Bundles

11-15

Common feature: Specify boundary between dependent instructions.

```
add r1, r2, r3
sub r4, r5, r6
! Boundary: because of r1 instruction below might wait.
xor r7, r1, r8
```

Because dependency information is in bundle less hardware is needed to detect dependencies.

How Dependency Information Can Be Specified (Varies by ISA):

- *Lookahead:*
  Number of bundles before the next true dependency.

- *Stop:*
  Next instruction depends on earlier instruction.

- *Serial Bit:*
  If 0, no dependencies within bundle(can safely execute in any order).

Specifying Dependencies Using Lookahead

11-16

Used in: Tera MTA.

Lookahead:
The number of consecutive following bundles not dependent on current bundle.

If lookahead 0, may be dependencies between current and next bundle.

If lookahead 1, no dependencies between current and next bundle, but may be dependencies between current and 2nd following bundle.

Setting the lookahead value:

Compiler analyzes dependencies in code, taking branches into account.

Sets lookahead based on nearest possible dependency.

Lookahead Example: (Two-instruction bundles.)

11-17

```
Bundle1: add r1, r2, r3
         add r4, r5, r6
         Lookahead = 1   ! Bundle 2 not dependent.

Bundle2: add r7, r7, r9
         add r10, r11, r12
         Lookahead = 2   ! Bundle 3 and Bundle 1 not dependent.

Bundle3: add r2, r1, r14
         bneq r20, Bundle1
         Lookahead = 0   ! Bundle 1 is dependent.

Bundle4: add r18, r8, r19
         bneq r21, Bundle1
         Lookahead = 11  ! Assuming twelfth bundle below uses r18.

Bundle5: nop
         nop

! (Next 10 bundles contain only nops)
```

Specifying Dependencies Using Stops

11-18

Used by: Itanium (née IA-64)

Stop:

Boundary between instructions with true dependencies and output dependencies.

Stop (and taken branches) divide instructions into *groups*.

Groups can span multiple bundles.

Within a group true and output register dependencies are not allowed, with minor exceptions.

Memory dependencies are allowed.

Assembler Notation (Itanium): Two consecutive semicolons: ;;.

Example:

```
L1: add r1, r2, r3
L2: add r4, r5, r6 ;;
L3: add r7, r1, r0 ;;
L4: add r8, r7, r0
L5: add r9, r4, r0
! Three groups: Group 1: L1, L2;   Group 2: L3;   Group 3: L4, L5
```


11-18

DLXV Assembly Notation

Example:

```
{ P 3                       ! Exec. in parallel or ignore dep. Lookahead = 3
  (r10)  add r1, r2, r3     ! Execute if r10 not zero.
  (~r11) sub r4, r1, r5     ! Execute if rll is zero.
         and r6, r7, r8     ! Always execute.
}
{ S 0                       ! Execute serially or honor dep. Lookahead = 0
  (r10)  add r20, r22, r23 ! Execute if r10 not zero.
  (~r11) sub r24, r20, r25 ! Execute if rll is zero.
  (~r0)  and r26, r27, r28 ! Always execute.
}
```

DLXV Assembly Notation

DLX instruction mnemonic ...

... with instruction preceded by predicate ...

... and group of three surrounded by braces ({}) ...

... starting with serial bit and lookahead.

Predicate format:

Register number possibly preceded by tilde.

Without tilde, instruction executes if register value non-zero.

With tilde, instruction executes if register value zero.

Serial Bit Mnemonic

If `S` then honor dependencies in bundle (serial bit = 1) ...

... if `P` then serial bit = 0.

Lookahead Menomonic

Integer indicating lookahead value.

VLIW Sample Problem

11-22

Based on Spring 1999 HW 4, Problem 7:

Rewrite the code below for the VLIW DLX ISA presented in class. Instructions can be re-arranged and register numbers changed. In order of priority, try to minimize the number of bundles, minimize the use of the serial bit, and maximize the value of the lookahead field. When determining the lookahead assume that any register can be used following the last bundle in your code.

```
LOOP:
 lf    f0, 0(r1)
 multf f1, f0, f0
 multf f2, f0, f1
 addf  f3, f3, f0
 lf    f4, 8(r1)
 sf    4(r1), f1
 multf f1, f4, f4
 multf f2, f4, f1
 addi  r1, r1, #16
 sub   r3, r4, r5
 xor   r6, r7, r8
 or    r9, r10, r11
```

Solution:

```
LOOP:
{ P 0
 lf    f0, 0(r1)
 lf    f4, 8(r1)
 sub   r3, r4, r5
}
{ P 0
 multf f1, f0, f0
 multf f11, f4, f4
 addf  f3, f3, f0
}
{ P 1
 sf    4(r1), f1
 multf f12, f0, f1
 multf f2, f4, f11
}
{ P 0
 addi  r1, r1, #16
 xor   r6, r7, r8
 or    r9, r10, r11
}
```

Sample Problems

11-24

Superscalar

1998 Final Exam, Problem 2. (Includes later material on branches.)

1998 Homework 5, Problems 1, 2. (Static scheduled superscalar.)

1998 Homework 5, Problem 3. (Includes later material on branches.)

1997 Final Exam problem 2.

VLIW

1998 Homework 5, Problem 4.

11-24

VLIW and Superscalar Comparison

11-25

What is Being Compared

An n-way superscalar implementation of conventional ISA.

An n-way implementation of a VLIW ISA.

Common Benefit

Can potentially execute n instructions per cycle.