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

Multiple Issue

11-2

Multiple-Issue Machine

A processor that can sustain 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.

11-3

Types of Multiple Issue Machines

Superscalar Processor

A multiple-issue machine that implements a conventional ISA (such as DLX 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, most code must be re-compiled.

Idea developed in early 1980's, . . .

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

. . . will be used in Intel's next generation processor.

Intel's Merced (code name) will implement the IA-64 VLIW ISA (Intel's) name.

Superscalar Machines

11-4

n-Way Superscalar Machine Construction

Start with a single-issue machine.

Duplicate hardware so each part can handle n instructions per cycle.

Don't forget about control and data hazards.

Superscalar Difficulties

11-5

Register File

Single-issue: 2 reads, 1 write per cycle.

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

Dependency Checking (Static Scheduling) For ALU Instructions

Single issue, about 4 comparisons per cycle.

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

Loads

Single issue, 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.

EE 4720 Lecture Transparency. Formatted 16:07, 1 December 1999 from lsli11.

11-6

Typical Superscalar Characteristics

11-7

Instruction Fetch

Instructions fetched in aligned blocks.

Unneeded instructions ignored.

Instruction Decode

Must issue all instructions fetched before processing next block.

Execution

Not all hardware is duplicated. (Don't expect n divide units.)

Some instruction pairs forbidden.

For example, early processors could simultaneously start one floating-point and one integer instruction.

11-8

Superscalar Execution Example

Two-Way Superscalar Example (Similar to 1998 Final Question)

Machine Characteristics

The program below executes on a 2-way, dynamically scheduled superscalar processor. The processor's features are summarized in the list below, the table gives details of the functional units. (The load/store unit computes the address in its first cycle and does the actual access in its second cycle. Address computation starts as soon as the address operand is ready.)

- ♢ Two-way superscalar instruction issue.
- ♢ Dynamically scheduled (see table), re-order buffer.
- ♢ CDB can accommodate results from any two functional units in any cycle.
- ♢ Reorder buffer for precise exceptions.
- ♢ Reorder buffer can retire as many as two instructions per cycle.
- ♢ Reservation stations, *not* reorder buffer, used for renaming.
- ♢ Branches do not have delay slots.

11-9

Name	Abbreviation	Latency	Initiation Interval	Reservation Station Nums
Load/Store	L	1	1	0-1
Integer	EX	0	1	2-3
F.P. Add	A	1	1	4-5
F.P. Mul.	M	5	2	6-7
Branch	BR	0	1	8-9
F.P. Divide	DIV	22	23	10-11

```
! When loop first entered r2-r1 large (loop iterates many times).
LOOP: ! LOOP = 0x1000
 slt   r3, r1, r2
 lf    f0, 0(r1)     ! Don't overlook true dependencies on f0!
 multf f0, f0, f2
 addf  f0, f0, f3
 sf    4(r1), f0
 addi  r1, r1, #8
 bnez  r3, LOOP
 div   f4, f5, f6
```

Superscalar Example

Numbers before stage names indicate reservation station number. (Reorder buffer entry numbers not shown).

Color (or shading) indicates iteration: IF First, IF Second, IF Third, IF Fourth.

```
Cycle:           0   1    2    3    4    5    6    7    8    9    10   11   12   13   14   15   16   17   18   19   20
slt  r3, r1, r2  IF  ID 3:EX 3:WC        IF   ID 2:EX 2:WB                          :C
                                                                   IF   ID 2:EX 2:WB                     :C
                                                                                                   IF   ID 3:RS
lf   f0, 0(r1)   IF  ID 0:L1 0:L2 0:WC   IF   ID 0:L1 0:L2 0:WB                          :C
                                                                   IF   ID ---> 1:L1 1:L2 1:WB                :C
                                                                                                   IF   ID 0:L1
mulf f0, f0, f2      IF   ID 6:RS 6:M1 6:M2 6:M3 6:M4 6:M5 6:M6 6:WC
                                          IF   ID 7:RS 7:M1 7:M2 7:M3 7:M4 7:M5 7:M6 7:WC
                                                                   IF --->   ID 6:RS 6:RS 6:M1 6:M2 6:M3 6:M4
                                                                                                       IF   ID
addf f0, f0, f3      IF   ID 4:RS 4:RS 4:RS 4:RS 4:RS 4:RS 4:RS 4:A1 4:A2 4:WC
                                          IF   ID 5:RS 5:RS 5:RS 5:RS 5:RS 5:RS 5:RS 5:A1 5:A2 5:WC
                                                                   IF --->   ID 4:RS 4:RS 4:RS 4:RS 4:RS 4:RS
                                                                                                       IF   ID
sf   4(r1), f0            IF   ID 1:L1 1:RS 1:RS 1:RS 1:RS 1:RS 1:RS 1:L2 1:WC
                                                IF   ID ---> 0:L1 0:RS 0:RS 0:RS 0:RS 0:RS 0:L2 0:WC
                                                                                IF   ID ---> 1:L1 1:RS 1:RS 1:RS
                                                                                                             IF
addi r1, r1, #8           IF   ID 2:EX 2:WB                                   :C
                                                IF   ID 2:EX 2:WB                                   :C
                                                                                IF   ID 2:EX 2:WB             IF
bnez r3, LOOP                  IF   ID 8:BR 8:WB                                :C
                                                     IF --->   ID 8:BR 8:WB                              :C
                                                                                     IF --->   ID 8:BR 8:WB
div fr,f5,f6                   IF   ID   x              IF --->   ID   x                  IF --->   ID   x
Cycle:           0   1    2    3    4    5    6    7    8    9    10   11   12   13   14   15   16   17   18   19   20
```

11-11

Comments on previous example.

Because machine is 2-way, at each cycle two instructions can be fetched, decoded, write back results, and committed (retired).

Because CDB can handle two results, there can be two writebacks per cycles, as in cycle 10.

At cycles 9, 13, and 16 execution stalls because there are no free load/store RS's.

As with all multiple-issue processors considered here, all instructions must leave ID before another group can enter. This can be seen in the stall at cycles 9, 13, and 16 caused by one instruction in ID but affecting both instructions in IF.

The address of `CYCLE` must be a multiple of 8. (If it weren't, `lf` would not be fetched in the same cycle as `slt`.).

VLIW and EPIC

11-12

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.

VLIW Features.

Instructions grouped into *bundles*, a.k.a. *packets*.

Bundles specify dependencies.

Dynamic scheduling easier because dependencies explicitly specified.

11-13

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.

11-14

Intel IA-64 ISA (Implemented by Merced [code name]).

Intended for general purpose use.

128 64-bit General-Purpose Integer Registers

128 FP Registers

Many additional special-purpose registers.

Instructions grouped into 128-bit bundles.

Each bundle includes three instructions and six *template bits*.

Template bits specify dependencies between and *within* bundles.

Makes extensive use of predication.

Implementation code-named *Merced*, expected to be available in 2000.

11-15

Tera MTA implemented by the Tera Computer Company

Intended for scientific computing.

In addition to VLIW has other unique features (not discussed).

32 64-bit registers.

Branches can examine any subset of 4 condition code registers.

Instructions grouped into 64-bit bundles.

Each bundle holds three instructions, with restrictions: one load/store, one ALU, and one ALU or branch.

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

VLIW DLX

11-16

VLIW Version of DLX: DLXV

128-bit bundles holding three instructions:

Each instruction consists of a DLX instruction ...
... and a 6-bit predicate field.

Predicate field specifies an integer register and test ($= 0$ or $\neq 0$).

Bundle also includes a 3-bit lookahead field and a 1-bit serial bit.

Lookahead field specifies number of following non-data-dependent bundles.

If serial bit is 1 dependencies within bundle are honored ...
... if bit is 0 then source operands refer to values produced in preceding bundles.

Layout: three DLX instructions (96 bits) followed by predicate fields, lookahead field, and serial bit.

11-17

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.
}
```

11-18

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.

EE 4720 Lecture Transparency. Formatted 16:07, 1 December 1999 from lsli11.
11-18

11-19

Lookahead Menomonic

Integer indicating lookahead value.

VLIW Sample Problem

11-20

Based on Spring 1999 HW 4, Problem 7:

Rewrite the code below for the VLIW DLX ISA presented in class. Instructions can be rearranged 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
```

11-21

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
}
```


EE 4720 Lecture Transparency. Formatted 16:07, 1 December 1999 from lsli11.
11-21

11-22

Sample Problems

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.