Advanced Computer Architecture
05-09-2019
Simplified single issue case

A) A processor embeds two cores that have private L1 and L2 caches, L3 is shared. The caches obey the MESI protocol and have the following structure: 32KB, 2-way L1 I-cache and D-cache, each 32-byte block; 2048KB, 8-way associative, 32-byte block L2 cache; 16MB, 8-way associative, 32-byte block L3 cache. The latencies (disregarding virtual memory TLB) expressed in clock cycles are: 2 in L1, 4 in L2, 8 in L3. Addresses are 48-bit long. Write operations are managed with a write-back policy. Assuming initially empty and invalidated cache lines throughout the hierarchy, consider the following memory accesses:

core 1) LD F1, 0000FFFFA0A0hex; (8-byte load)

core 2) LD F3, 0000FFFFA0A8hex; (8-byte load)

core 1) ST F1, 0000FFFFA0A8hex; (8-byte store)

core 2) ST F3, 0000FFFFA0B0hex; (8-byte store)

a1) show the blocks involved in each cache, and the associated MESI state after each instruction; describe actions carried out by cache controllers;

a2) show a fragment of code to be run in core 1 that completely fills the set involved in the execution of the operations above.

B) Let us consider the following code fragment of compiler-unscheduled instructions:

 ADDI R1,R0,0000FFFFA0A0hex -- R1 set to base address 0000FFFFA0B0hex

loop: LD F1,0(R1)

 LD F3,8(R1)

ADDF F2,F1,F1

ADDI R1,R1,8

ADDF F4,F3,F3
 LD F3,-16(R1)

 MULTF F5,F4,F2

 ST F4,0(R1)

 ADDF F6,F3,F5

 ST F6,8(R1)

 ADDI R1,R1,24

 BLI R1, 000100000000hx,loop -- compare immediate, branch on less

assuming this code fragment is executed in a statically scheduled pipeline with stages

IF1|ID|INT1| |ME1|ME2|WB

 |A1 |A2 |

 |M1 |M2 |M3|

and proper forwarding units, assuming that branch instructions take a decision in stage ME1, assuming that all LD hit in L1 D-cache during ME1, and that all ST hit during ME2

b1) show a producer/consumer table

b2) show a POE (compiler schedule) that minimizes the clock cycles required to complete execution, using register renaming, if necessary;

b3) show a ROE for the first iteration of the optimized POE and determine the CPI of the execution of this optimized kernel.

C) Determine the number of hit and misses in all memory hierarchy for the loop of point B.

D) Each core of the processor described in A) is organized as superscalar, 1-way pipeline, that fetches, decodes issues and retires (commits) instructions. The front-end in-order section (fetch and decode) consists of 2 stages. The architecture supports dynamic speculative execution, and control dependencies from branches are solved when the branch evaluates the condition, even if it is not at commit. The execution model obeys the attached state transition diagram. There is a functional unit (FUs) Int1 for integer arithmetics (arithmetic and local instructions, branches and jumps, no multiplication), 1 FUs FAdd1 for floating point addition/subtraction, 1 FU FMolt1 for floating point multiplication, and a FU for division, FDiv1.

There are 12 integer (R0-R11) and 12 floating point (F0-F11) registers. Speculation is handled through a 8-entry ROB, a pool of 4 Reservation Stations (RS) Rs1-4 shared among all FUs, 2 load buffer Load1-Load2, 2 store buffers Store1 – Store2 (see the attached execution model): an instruction bundle is first placed in the ROB (if two entries are available), then up to 2 instructions are dispatched to the shared RS (if available) when they are ready for execution and then executed in the proper FU. FUs are *pipelined* (except for the float division unit, which is blocking) and have the latencies quoted in the following table:

|  |  |
| --- | --- |
| Int - 2 | Fadd – 2 |
| Fmolt – 4 | Fdiv – 8 |

Further assumption

* The code is assumed to be already in the I-cache; data caches are described in point A) and are assumed empty and invalidated; the cost of a miss is 40.

d1) assuming a write-back protocol for cache management, show state transitions for all instructions in the first iteration and instructions up to PC04 included in the second iteration :

 PC01 ADDI R1,R0,0000FFFFA0A0hex -- R1 set to base address 0000FFFFA0B0hex

 PC02 LD F1,0(R1)

 PC03 LD F3,8(R1)

 PC04 ADDF F2,F1,F1

 PC05 ADDI R1,R1,8

 PC06 ADDF F4,F3,F3

 PC07 LD F3,-16(R1)

 PC08 MULTF F5,F4,F2

 PC09 ST F4,0(R1)

 PC10 ADDF F6,F3,F5

 PC11 ST F6,8(R1)

 PC12 ADDI R1,R1,24

 PC13 BLI R1, 000100000000hx,PC02 -- compare immediate, branch on less

d2) show ROB, RS and buffer status at the issue of PC003 in the second iteration;
d3) estimate the speed-up of this code loop resulting from a reduction of the L1 cache latency to 1 clock cycle.

Dynamic speculative execution

Decoupled ROB RS execution model – single issue

|  |  |  |  |
| --- | --- | --- | --- |
| **ISTRUCTION** |  |  | **INSTRUCTION STATE**  |
|  | **n.ite** | **ROBpos** | **WO** | **RE** | **DI** | **EX** | **WB** | **RR** | **CO** |
| PC01 ADDI R1,R0,0000FFFFA0A0hex  | **1** | **0** |  | **1** | **2** | **3-4** | **5** | **6** | **7** |
| PC02 LD F1,0(R1) | **1** | **1** | **2-4** | **5** | **6** | **7-60** | **61** | **62** | **63** |
| PC03 LD F3,8(R1) | **1** | **2** | **3-4** | **5** | **6** | **7-61** | **62** | **63** | **64** |
| PC04 ADDF F2,F1,F1 | **1** | **3** | **4-60** | **61** | **62** | **63-64** | **65** | **66** | **67** |
| PC05 ADDI R1,R1,8 | **1** | **4** |  | **5-6** | **7** | **8-9** | **10** | **11-67** | **68** |
| PC06 ADDF F4,F3,F3 | **1** | **5** | **6-61** | **62** | **63** | **64-65** | **66** | **67-68** | **69** |
| PC07 LD F3,-16(R1) | **1** | **6** | **7-9** | **10-61** | **62** | **63-116** | **117** | **118** | **119** |
| PC08 MULTF F5,F4,F2  | **1** | **7** | **8-65** | **66** | **67** | **68-71** | **72** | **73-119** | **120** |
| PC09 ST F4,0(R1) | **1** | **0** | **9-65** | **66** | **67** | **68-69** | **-** | **70-120** | **121** |
| PC10 ADDF F6,F3,F5 | **1** | **1** | **64-116** | **117** | **118** | **119-110** | **121** | **122** | **123** |
| PC11 ST F6,8(R1) | **1** | **2** | **65-120** | **121** | **122** | **123-124** | **-** | **125** | **126** |
| PC12 ADDI R1,R1,24 | **1** | **3** |  | **68** | **69** | **70-72** | **73** | **74-126** | **127** |
| PC13 BLI R1, 000100000000hx,PC02  | **1** | **4** | **69-72** | **73** | **74** | **75-76** | **-** | **77-127** | **128** |
| PC14 INSTR A |  | **5** | **Cancelled at 76** |
| PC15 INSTR B |  |  | **Cancelled at 76** |
| PC02 LD F1,0(R1) | **2** | **5** |  | **81** | **82** | **83-136** | **137** | **138** | **139** |
| PC03 LD F3,8(R1) | **2** | **6** |  | **82** | **83-117** | **118-137** | **138** | **139** | **140** |
|  |  |  |  |  |  |  |  |  |  |

PC13 sends the new PC to NextPC logic at 77, I-cache fetch in 78 and 79, decode in 80, so issue of PC12 at 81.

|  |  |
| --- | --- |
|  | Reservation station and load/store buffers |
| Busy | Op | Vj | Vk | ROBj | ROBk | ROB pos | Address |
| Rs1 |  |  |  |  |  |  |  |  |
| Rs2 |  |  |  |  |  |  |  |  |
| Rs3 |  |  |  |  |  |  |  |  |
| Rs4 |  |  |  |  |  |  |  |  |
| Load1 |  |  |  |  |  |  |  |  |
| Load2 |  | PC02 LD F1,0(R1) |  |  | 3 |  | 5 | 0+[ROB3] |
| Store2 |  |  |  |  |  |  |  |  |
| Store1 |  |  |  |  |  |  |  |  |

ROBj ROBk: sources not yet available

ROB pos: ROB entry number where instruction is located

|  |  |
| --- | --- |
|  | Result Register status |
| Integer | R0 | R1 | R2 | R3 | R4 | R5 | R6 | R7 | R8 | R9 | R10 | R11 |  |  |
| ROB pos |  | 3 |  |  |  |  |  |  |  |  |  |  |  |  |
| state |  | B |  |  |  |  |  |  |  |  |  |  |  |  |
| Float. | F0 | F1 | F2 | F3 | F4 | F5 | F6 | F7 | F8 | F9 | F10 | F11 |  |  |
| ROB pos |  | 5 |  | 6 |  | 7 | 1 |  |  |  |  |  |  |  |
| state |  | B |  | B |  | B | B |  |  |  |  |  |  |  |

|  |
| --- |
| Reorder Buffer (ROB) |
|  ROB Entry#  | Busy | Op | Status | Destination | Value |
| 0 | yes | PC09 ST F4,0(R1) | RR | 0+[ROB4] | [ROB5] |
| 1 | yes | PC10 ADDF F6,F3,F5 | WO | F6 |  |
| 2 | yes | PC11 ST F6,8(R1) | WO | 8+[ROB4] | [ROB1] |
| 3 | yes | PC12 ADDI R1,R1,24 | RR | R1 |  |
| 4 | yes | PC13 BLI R1, 000100000000hx,PC02  | RR |  | PC02 |
| 5 | yes | PC02 LD F1,0(R1) | DI | F1 |  |
| 6 | yes | PC03 LD F3,8(R1) | WO | F3 |  |
| 7 | yes | PC08 MULTF F5,F4,F2  | RR | F5 |  |

CLOCK cycle 82

DI

EX

WB

RR

RE

CO

wo

QUEUE

**Decoupled execution model for bundled (paired) instructions**

The state diagram depicts the model for a dynamically scheduled, speculative execution microarchitecture equipped with a Reorder Buffer (ROB) and a set of Reservation Stations (RS). The ROB and RSs are allocated during the ISSUE phase, denoted as RAT (Register Alias Allocation Table) in INTEL microarchitectures, as follows: one instructions if fetched from the QUEUE of decoded instructions and ISSUED if there is a free entry in the ROB ( head and tail of the ROB queue do not match); a maximum of two instructions are moved into the RS (if available) when all of their operands are available. Access memory instructions are allocated in the ROB and then moved to a load/store buffer (if available) when operands (address and data, if proper) are available .

**States** are labelled as follows:

WO: Waiting for Operands (at least one of the operands is not available)

RE: Ready for Execution (all operands are available)

DI: Dispatched (posted to a free RS or load/store buffer)

EX: Execution (moved to a load/store buffer or to a matching and free UF)

WB: Write Back (result is ready and is returned to the Rob by using in exclusive mode the Common Data Bus CDB)

RR: Ready to Retire (result available or STORE has completed)

CO: Commit (result is copied to the final ISA register)

**State transitions** happen at the following events:

 *from* QUEUE *to* WO: ROB entry available, operand missing

*from* QUEUE *to* RE: ROB entry available, all operands available

*loop at* WO: waiting for operand(s)

*from* WO *to* RE: all operands available

*loop at* RE: waiting for a free RS or load/store buffer

*from* RE *to* DI: RS or load/store buffer available

*loop on* DI: waiting for a free UF

*from* DI *to* EX: UF available

*loop at* EX: multi-cycle execution in a UF, or waiting for CDB

*from* EX *to* WB: result written to the ROB with exclusive use of CDB

*from* EX *to* RR: STORE completed, branch evaluted

*loop at* RR: instruction completed, not at the head of the ROB, or bundled with a not RR instruction

*from* RR *to* CO: RR instruction at the head of the ROB, no exception raised

**Resources***Register-to-Register* instructions hold resources as follows:

ROB: from state WO (or RE) up to CO, inclusive;

RS: state DI

UF: EX and WB

*Load/Store* instructions hold resources as follows:

ROB: from state WO (or RE) up to CO, inclusive;

Load buffer: from state WO (or RE) up to WB

Store buffer: from state WO (or RE) up to EX (do not use WB)

**Forwarding**: a write on the CDB (WB) makes the operand available to the consumer in the same clock cycle. If the consumer is doing a state transition from QUEUE to WO or RE, that operand is made available; if the consumer is in WO, it goes to RE in the same clock cycle of WB for the producer.

**Branches**: they compute Next-PC and the branch condition in EX and optionally forward Next-PC to the “in-order” section of the pipeline (Fetch states) in the next clock cycle. They do not enter WB and go to RR instead.