Advanced Computer Architecture
16 September 2013

A) A processor has the following cache hierarchy: L1 I-cache, 16 KB, direct mapped, 8-byte block; L1 D-cache 16KB, 2-way associative, 8-byte block; unified L2, 512 KB, 2-way associative, 16-byte block. The latencies (disregarding virtual memory TLB) expressed in clock cycles are: 2 in L1, 3 in L2. The processor executes the IA-32 ISA (but with 32-bit addressing mode).

a1) compute the number of 1x1Kbit SRAM modules necessary to build the caches;
a2) considering a floating point array v[], with 512 8-byte entries, allocated in memory from address 0xAAAAFF00 v[0], compute the number of misses incurred upon in the execution of the following algorithm:

```
w=v[511];
for (i=511; i>0; i--) {
v[i]=v[i-1]};
v[0]=w;
```

a3) Suggest another algorithm that minimises the number of misses.

The D-caches are supposed empty. Disregard I-cache.

B) A processor has a superscalar, 2-way pipeline, that fetches, decodes issues and retires (commits) bundles containing each 2 instructions. The front-end in-order section (fetch and decode) consists of 2 stages. The issue logic takes 1 clock cycle, if the instructions in the bundle are independent, otherwise it takes 2 clock cycles. The architecture supports dynamic speculative execution, and control dependencies from branches are solved only at commit time, even if the outcome of the branch logic is available earlier. The execution model obeys the attached state transition diagram.
There are 2 functional units (FUs) Int1-INT2 for integer arithmetics (arithmetic and local instructions, branches and jumps, no multiplication ), 2 FUS FAdd1-Fadd2 for floating point addition/subtraction, a 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 buffers Load1-2, 2 store buffers Store1-2 (see the attached execution model): an instruction bundle is first placed in the ROB (if 2 entries are available), a maximum of 4 instructions are dispatched to the shared RS (if available) and then executed in the proper FU. Floating point FUs are *pipelined* (not the Fdiv one), integers Fus are *blocking*, and have the latencies quoted in the following table:

| | |
|---|---|
| Int - 2 | Fadd – 3 |
| Fmolt – 4 | Fdiv – 6 |

Further assumption

- a BTB with 256 entries, that applies a prediction logic with a (0,2) predictor.
- caches are described in point A) and are assumed empty and invalidated. A miss costs 4 clock cycles.

b1) show state transitions for the instructions of the first iteration of the following code fragment, highlighting conflicts, if any:

```
PC01 MOVI  R1, 0xAAAAFF00
PC02 XOR   R2,R2,R2         set R2 to zero
PC03 ADDI  R2,R2,256
PC04 LD    F3,0(R1)
PC05 MULTF F4,F3,F2
PC06 LD    F5,2048(R1)
PC07 ADDF  F6,F3,F5
PC08 ADDI  R1,R1,8
PC09 MULTF F5,F5,F2
PC10 ADDF  F5,F6,F5
PC11 SD    -8(R1),F5
PC12 SD    2040(R1),F6
PC13 SUBI  R2,R2,1
PC14 BNEZ  R2,PC04
```

b2) show ROB, RS and buffer status at the issue of PC04 of the SECOND iteration.
b3) show the content of the BTB at the end of the algorithm.
b4) extend the analysis of b1) to 2 iterations and compute the CPI of the algorithm.

Dynamic speculative execution
Decoupled ROB RS execution model

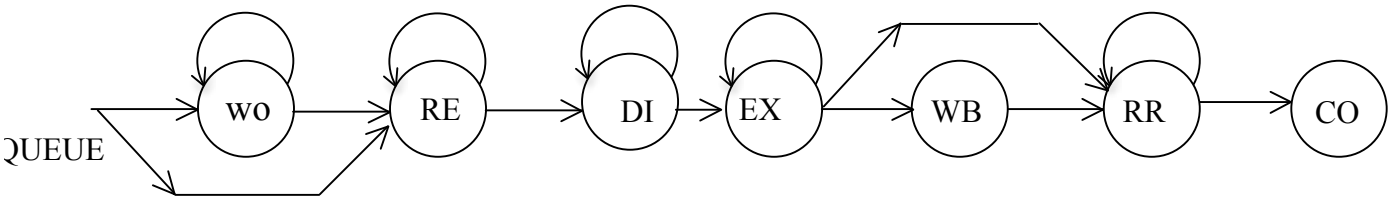
| ISTRUCTION | | | INSTRUCTION STATE | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | n. ite | ROB pos | WO | RE | DI | EX | WB | RR | CO |
| PC01 MOVI R1, 0xAAAAFF00 | - | | | | | | | | |
| PC02 XOR R2,R2,R2 | - | | | | | | | | |
| PC03 ADDI R2,R2,256 | - | | | | | | | | |
| PC04 LD F3,0(R1) | 1 | | | | | | | | |
| PC05 MULTF F4,F3,F2 | 1 | | | | | | | | |
| PC06 LD F5,2048(R1) | 1 | | | | | | | | |
| PC07 ADDF F6,F3,F5 | 1 | | | | | | | | |
| PC08 ADDI R1,R1,8 | 1 | | | | | | | | |
| PC09 MULTF F5,F5,F2 | 1 | | | | | | | | |
| PC10 ADDF F5,F6,F5 | 1 | | | | | | | | |
| PC11 SD -8(R1),F5 | 1 | | | | | | | | |
| PC12 SD 2040(R1),F6 | 1 | | | | | | | | |
| PC13 SUBI R2,R2,1 | 1 | | | | | | | | |
| PC14 BNEZ R2,PC04 | 1 | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |

| | Reservation station and load/store buffers | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | Busy | Op | Vj | Vk | $ROB_j$ | $ROB_k$ | ROB pos | Address |
| Rs1 | | | | | | | | |
| Rs2 | | | | | | | | |
| Rs3 | | | | | | | | |
| Rs4 | | | | | | | | |
| Load1 | | | | | | | | |
| Load2 | | | | | | | | |
| Store1 | | | | | | | | |
| Store2 | | | | | | | | |

$ROB_j$ $ROB_k$: 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 | | | | | | | | | | | | | | |
| state | | | | | | | | | | | | | | |
| Float. | F0 | F1 | F2 | F3 | F4 | F5 | F6 | F7 | F8 | F9 | F10 | F11 | | |
| ROB pos | | | | | | | | | | | | | | |
| state | | | | | | | | | | | | | | |

| Reorder Buffer (ROB) | | | | | |
|---|---|---|---|---|---|
| ROB Entry# | Busy | Op | Status | Destination | Value |
| 1 | | | | | |
| 2 | | | | | |
| 3 | | | | | |
| 4 | | | | | |
| 5 | | | | | |
| 6 | | | | | |
| 7 | | | | | |
| 8 | | | | | |




**Decoupled execution model for bundled 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 RSs are allocated during the ISSUE phase, denoted as RAT (Register Alias Allocation Table) in INTEL microarchitectures, as follows: a bundle (2 instructions) is fetched from the QUEUE of decoded instructions and ISSUED if there is a free couple of consecutive entries in the ROB ( head and tail of the ROB queue do not match); 4 instructions (maximum) are moved into 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: | bundle of RR instructions 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 (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.