# 18640 – Fall 2014
# Homework 2
# (Due Date – 10/14/2014 4:00 pm PST)

**1. Multiple Choice Questions (2 points each)**

1. In general, increasing cache block size(no. of bytes per cache line) causes ___
   a) The number of cold misses to increase
   **b) The number of capacity misses to increase**
   **c) The number of conflict misses to increase**
   d) The number of all types of misses to decrease

2. Path history in a dynamic branch predictor ___
   a) Is equivalent to local branch outcome history
   b) Is equivalent to global branch outcome history
   **c) Can be better than either local or global branch history outcomes**
   d) Is always better than either local or global branch history outcomes

3. The return address stack(RAS): ___
   a) Pushes the return address whenever a call instruction in encountered
   b) Pops a return address for every return instruction
   c) Raises an exception when the stack underflows or overflows
   **d) Both (a) and (b)**

4. To support precise exceptions, an out-of-order processor can: ___
   a) Use a future file to buffer committed register values
   b) Use a history file to buffer uncommitted register values
   **c) Use a physical register file to hold both committed and uncommitted register values**
   d) None of the above

5. Register names are reused as real programs execute because: ___
   a) Programmers are stingy with register usage
   b) Register reuse reduces the size of the program binary, reducing instruction cache misses
   c) There just aren't enough registers, so they have to be reused
   **d) Programs have loops and multiple loop iterations are executed concurrently**

**2. Register Renaming (20 points)**

The following set of instructions is to be renamed onto a set of physical registers. The initial register map and free list are given. Show the rename mappings after each instruction has been dispatched by filling out the table below, rewriting each instruction with renamed source and destination registers and updating the rename mappings in the table appropriately. The first instruction has been done for you. Assume that registers are allocated from the left of the free list, and that freed registers are returned to the right of the free list. (16 points) ___

| | (Alloc here)-> | <-(Free here) |
|---|---|---|
| **Free list** | P8, P9, P10, P11 | |

| Cycle | | Original Instruction | Renamed Instruction | Rename mappings after dispatch | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | R1 | R2 | R3 | R4 | R5 | R6 | R7 | R8 |
| Dispatched | Committed | Initial register mappings | | P1 | P2 | P3 | P4 | P5 | P6 | P7 | P0 |
| 1 | 3 | R5 <= R1 + R5 | P8 <= P1 + P5 | | | | | P8 | | | |
| 1 | 3 | R2 <= mem (R3 + R1) | P9 <= mem(P3 + P1) | | P9 | | | | | | |
| 2 | 4 | R1 <= R3 + 4 | P10 <= P3 + 4 | P10 | | | | | | | |
| 2 | 5 | R7 <= R1 + R2 | P11 <= P10 + P9 | | | | | | | P11 | |
| 4 | 6 | R4 <= R4 – 1 | P5 <= P4 - 1 | | | | P5 | | | | |
| 4 | 6 | R4 <= R4 + R8 | P2 <= P5 + P0 | | | | P2 | | | | |
| 6 | 9 | R7 <= R4 – R7 | P1 <= P2 - P11 | | | | | | | P1 | |
| 6 | 9 | R4 <= mem(R1 + R6) | P7 <= mem(P10 + P6) | | | | P7 | | | | |
| 8 | 10 | R3 <= R5 – R4 | P4 <= P8 - P7 | | | P4 | | | | | |

Does the allocator ever have to stall to wait for registers to be added to the free list?
Explain why or why not? (4 points)

No, there are enough number of registers when the instructions start executing. The maximum number of instructions that are being concurrently executed.

**3. Tomosulo's Algorithm (25 points)**

Simulate the execution of the following code snippet using Tomosulo's algorithm. Show the contents of the reservation station entries, register file busy, tag (the tag is the RS ID number), and data fields for each cycle (make a copy of the table below for each cycle that you simulate). Indicate which instruction is executing in each functional unit in each cycle. Also indicate any result forwarding across a common data bus by circling the producer and consumer and connecting them with an arrow

i: R4 ← R0 + R8
j: R2 ← R0 * R4
k: R4 ← R4 + R8
l: R8 ← R4 * R2

Assume dual dispatch and dual CDB(Common Data Bus). Add latency is one cycle, and multiply latency is 3 cycles. An instruction can begin execution in the same cycle that it is dispatched, assuming all dependencies are satisfied.(20 points)

NOTE: add latency is one cycle

CYCLE #1: Dispatched instructions : i,j

RS

| | | Tag | Sink | Tag | Source |
|---|---|---|---|---|---|
| i | 1 | 0 | 6.0 | 0 | 7.8 |
| | 2 | | | | |
| | 3 | | | | |
| | | | i | | Adder |

RS

| | | Tag | Sink | Tag | Source |
|---|---|---|---|---|---|
| j | 4 | 0 | 6.0 | 1 | |
| | 5 | | | | |
| | | | Mult/Div | | |

FLR

| | Busy | Tag | Data |
|---|---|---|---|
| 0 | | | 6.0 |
| 2 | Yes | 4 | 3.5 |
| 4 | Yes | 1 | 10.0 |
| 8 | | | 7.8 |

CYCLE#2 Dispatched instructions : k,l

RS

| | | Tag | Sink | Tag | Source |
|---|---|---|---|---|---|
| | 1 | | | | |
| k | 2 | 0 | 13.8 | 0 | 7.8 |
| | 3 | | | | |
| | | | k | | Adder |

RS

| | | Tag | Sink | Tag | Source |
|---|---|---|---|---|---|
| j | 4 | 0 | 6.0 | 0 | 13.8 |
| l | 5 | 2 | | 4 | |
| | | | j | | Mult/Div |

FLR

| | Busy | Tag | Data |
|---|---|---|---|
| 0 | | | 6.0 |
| 2 | Yes | 4 | 3.5 |
| 4 | Yes | 2 | 13.8 |
| 8 | Yes | 5 | 7.8 |

CYCLE#4(Skipped 3): Dispatched instructions :

RS

| | Tag | Sink | Tag | Source |
|---|---|---|---|---|
| 1 | | | | |
| 2 | | | | |
| 3 | | | | |
| | | Adder | | |

RS

| | | Tag | Sink | Tag | Source |
|---|---|---|---|---|---|
| j | 4 | 0 | 6.0 | 0 | 13.8 |
| l | 5 | 0 | 21.6 | 4 | |
| | | | j | | Mult/Div |

FLR

| | Busy | Tag | Data |
|---|---|---|---|
| 0 | | | 6.0 |
| 2 | Yes | 4 | 3.5 |
| 4 | | 0 | 21.6 |
| 8 | Yes | 5 | 7.8 |

CYCLE#5: Dispatched Instruction(s):

RS

| | Tag | Sink | Tag | Source |
|---|---|---|---|---|
| 1 | | | | |
| 2 | | | | |
| 3 | | | | |
| | Adder | | | |

RS

| | | Tag | Sink | Tag | Source |
|---|---|---|---|---|---|
| | 4 | | | | |
| 1 | 5 | 0 | 21.6 | 0 | 82.8 |
| | | | 1 | | Mult/Div |

FLR

| | Busy | Tag | Data |
|---|---|---|---|
| 0 | | | 6.0 |
| 2 | | | 82.8 |
| 4 | | | 21.6 |
| 8 | Yes | 5 | 7.8 |

CYCLE#:7 (Skipped 6) Dispatched Instruction(s):

RS

| | Tag | Sink | Tag | Source |
|---|---|---|---|---|
| 1 | | | | |
| 2 | | | | |
| 3 | | | | |
| | Adder | | | |

RS

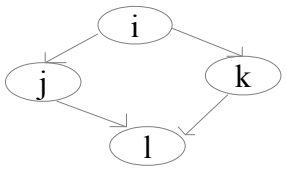
| | | Tag | Sink | Tag | Source |
|---|---|---|---|---|---|
| | 4 | | | | |
| 1 | 5 | 0 | 21.6 | 0 | 82.8 |
| | | | 1 | | Mult/Div |

FLR

| | Busy | Tag | Data |
|---|---|---|---|
| 0 | | | 6.0 |
| 2 | | | 82.8 |
| 4 | | | 21.6 |
| 8 | Yes | 5 | 7.8 |

b. Determine whether or not the code executes the dataflow limit for the above problem. Explain why or why not. (5 points)

Critical path is 1 + 3 + 3 = 7 cycle. Therefore, Tomasulo executes this at the dataflow limit.




## 4. Caches (15 points)

Consider a processor with 32-bit virtual addresses, 4KB pages and 36-bit physical-addresses. Assume memory is byte-addressable(i.e 32-bit virtual address specifies a byte in memory).

a. L1 Instruction cache: 64 Kbytes, 128 byte blocks, 4-way set-associative, indexed and tagged with virtual address.

b. L1 Data cache: 32 Kbytes, 64 byte blocks, 2-way set-associative, indexed and tagged with physical-address, write-back

c. 4-way set-associative TLB with 128 entries in all. Assume that TLB keeps a dirty bit, a reference bit, and 3 permission bits(read, write, execute) for each entry.

Specify the number of offset, index, and tag bits for each of these structures in the table below. Also, compute the total size in number of bit cells for each of the tag and data arrays.

| Structure | Offset bits | Index bits | Tag bits | Size of tag array | Size of data array |
|---|---|---|---|---|---|
| I-cache | 7 | 7 | 18 | 512 * (18 tag + 1 valid) = 9728 | 64 KB |
| D-cache | 6 | 8 | 22 | 512 * (22 + 1 valid + 1 dirty) = 12,288 bits | 32 KB |
| TLB | 12 | 5 | 15 | 128 * (15 tag + v) = 2048 bits | 128 * (24 + d + r + 3p) = 3712 bits |

5. **Memory Dataflow (10 points)**

A. In an out-of-order speculative microdataflow processor, a store operation is committed to the memory subsystem only when it is the oldest unfinished instruction. Based on this, can you conclusively eliminate the possibility of WAW and WAR memory data hazard?Explain(5 points)

There is no WAW hazard as stores happen in order. Stores happen only when all the previous loads to that particular address are finished. So, there is no WAR hazard.

.

B. How does load forwarding and load bypassing improve the performance of a scalar in-order pipelined processor (even though loads and stores are executed in order in the same stage)? Explain(5 points)

The in-order scalar processor can make use of store buffer and not let the store cache misses affect the execution. In load forwarding technique, data is forwarded from completed (not retired) stores to later loads. Load bypassing allows loads to bypass store buffer as they are critical for faster execution.

**6. Cache Performance Analysis(20 points)**

Assume the following:

- L1 instruction cache with 98% per instruction hit rate
- L1 data cache with 94% per instruction hit rate
- Shared L2 cache with 35% local miss rate (i.e. 35% misses per reference to the L2)
- L1 miss penalty of 12 cycles to L2 hit
- L2 miss penalty of:
    - 150 cycles latency to request word from memory
    - 8 cycles per 8B bus transfer, 8x8B = 64B block transferred
    - Hence 64 cycles transfer plus 1 cycle to fill L2
    - Total penalty 150+64+1 = 215 cycles

Compute the following CPI adders corresponding to the misses occurring at each cache:

L1 instruction cache CPI adder (assuming perfect L2):

.02 miss/inst x 12 cyc/miss = .24 CPI

L1 data cache CPI adder (assuming perfect L2):

.06 miss/inst x 12 cyc/miss = .72 CPI

L2 CPI adder:

(.02 L1miss/inst + .06 L1miss/inst) x .35 L2miss/L1miss x 215 cyc/L2miss = 6.02 CPI