CHALMERS

2021-08-13

Exam in DAT 105 (DIT 051) Computer Architecture

Time: August 17, 14-18 (Canvas)

Person in charge of the exam: Per Stenström, Phone: 0730-346 340

Supporting material/tools: Chalmers approved calculator, textbook.

Exam Review: More information on this will be available via Canvas

Grading intervals:

- **Fail**: Result < 24
- **Grade 3**: 24 <= Result < 36
- **Grade 4:** 36 <= Result < 48
- **Grade 5:** 48 <= Result

NOTE 1: Bonus points from Real-stuff studies and Quizzes will be added to the exam results for approved exams used solely for higher grades.

NOTE 2: Answers must be given in English

NOTE 3: Read the document Instructions for Canvas exam, available at Canvas, carefully

GOOD LUCK!
Per Stenström

CHALMERS UNIVERSITY OF TECHNOLOGY *DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING*
412 96 Göteborg
Visiting address: Rännvägen 5
Phone: 031-772 1761 Fax: 031-772 3663
Org. Nr: 556479-5598
E-mail: pers@chalmers.se

[General disclaimer: If you feel that sufficient facts are not provided to solve a problem, either 1) ask the teacher when he visits the exam, or 2) make your own additional assumptions. Additional assumptions will be accepted if they are reasonable and required to solve the problem. Always make sure to motivate your answers.]

ASSIGNMENT 1

1A) We want to compare the performance of two computer systems A and B using execution statistics from three programs P1, P2, and P3. For all systems, including the reference machine R, we assume that all instructions execute in a single cycle except for memory instructions. Memory instructions, whose data requests miss in the cache, experience a miss penalty of 100 nanoseconds.

The number of instructions executed for programs P1, P2 and P3 differ across systems. Similarly, the number of cache misses per thousand instructions (MPKI) also differ across the programs. The clock frequencies for the three systems also differ. All these statistics are shown in the tables below. Finally, P1 is run twice as often as P2 that is run twice as often as P3.

Instructions (Millions)	**P1**	**P2**	**P3**
System A	**1000**	**2000**	**1500**
System B	**4000**	**1000**	**2000**
System R	**1000**	**1000**	**1000**

MPKI	**P1**	**P2**	**P3**
System A	**10**	**20**	**15**
System B	**20**	**5**	**10**
System R	**5**	**5**	**5**

Clock freq. (GHz)	
System A	**1**
System B	**2**
System R	**1**

i) Derive the **weighted arithmetic mean** of the execution times to compare the performance of the two systems (A and B). **(4 points)**

ii) Derive the **harmonic means of the speedup** of A and B over the reference machine R. (**2 points**)

iii) Derive the **geometric mean** speedup over the reference machine R for the two systems A and B. (**2 points)**

1B) Multicore systems promise to speed up an application linearly with the number of processors. Now, if an application spends 25% of its execution in a part that cannot be executed in parallel, what is the maximum speedup that can be obtained on an ideal multicore system with an infinite number of processors? **(4 points)**

ASSIGNMENT 2

Assume a processor with a simple five-stage pipeline (Instruction fetch, Instruction decode, Execute, Memory access, Write back). Furthermore, assume that all memory accesses complete in a single cycle (cache hit time is one cycle), that pipeline forwarding is used whenever possible to resolve RAW hazards, and that there are two branch delay slots (all branch computations are completed in the EX stage). Integer multiply operations are performed in four steps; the first three in the Execute stage, and the fourth in the Memory Access stage, so multiply operations are fully pipelined like all other arithmetic instructions, but the result is not available until the end of the Memory access stage. Finally, for Branch instructions, branch conditions are established in the EX stage.

Let us consider the computation according to the code below.

```
for(int i=0; i<n; i=i+1) {
        x[i] = x[i]*x[i];
}
```

The x array is an array of integers (4 bytes). You may assume that register R1 already is loaded with the start address of the x array, i.e. the address of x[0]. The address of x[n] is likewise available in register R2.

2A) The following is a first version of the code:

```
LOOP:       LD          R4, 0(R1)
            MUL         R5, R4, R4
            SD          R5, 0(R1)
            ADDI        R1, R1, 8
            BNE         R1, R2, LOOP
```

Disregarding the specific pipeline, list all RAW, WAR and WAW hazards in the code above. (**3 points)**

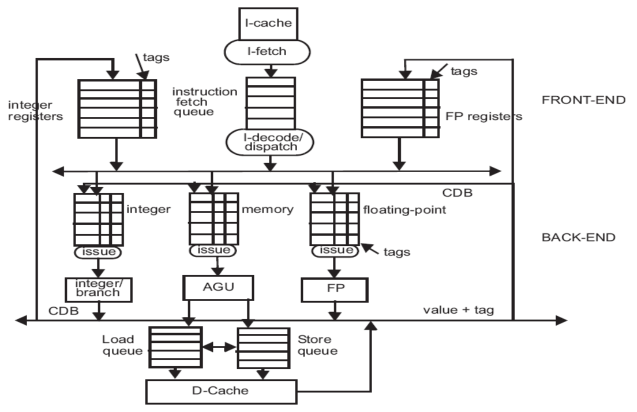
2B) Which of the hazards in the previous assignment will lead to performance loss for the specific pipeline? (**3 points)**

2C) Unroll the loop four times and schedule the instructions to minimize the number of cycles lost due to hazards. Which hazards will remain? **(3 points)**

2D) Explain what a 2-bit branch predictor is concerning the structure itself, how it is integrated in the example pipeline, how it is operated and when it will fail to correctly predict the branch outcome. **(3 points)**

ASSIGNMENT 3

The diagram below shows a pipeline with support for Tomasulo's algorithm. There are two functional units for adding floating-point numbers and a single functional unit for floating-point division. It takes 2 cycles to carry out an addition/subtraction and 5 cycles to carry out a division.



3A) Explain *in detail* what happens in each of the three pipeline stages: Issue, Execute, and Write result. In particular, explain how data hazards are resolved and in which cycle each instruction in the sequence below enters the different stages by filling out a pipeline diagram similar to the one below for the following instruction sequence. (**6 points)**

```
ADDD   F1, F2, F3              ; –O1
DIVD   F4, F1, F2              ; –O2
SUBD   F2, F4, F6              ; –O3
ADDD  F4, F1, F1               ; –O4
```

	Cycle 1	Cycle 2	Cycle 3	Cycle 4	Cycle 5	Cycle 6
O1	Issue					
O2		Issue				
O3			Issue			
O4				Issue		

3B) Consider the code below as it runs on a pipeline like the one above with support for speculative execution. Assume that the branch instruction is predicted to NOT be taken and that the prediction is validated to be correct when the last instruction has been speculatively executed. Explain in detail how the speculative processor keeps track of the register values during speculative execution and after how many cycles after the branch instruction has been validated, all register values are available in the register file. (**4 points)**

```
BNEZ   R1, LABEL
ADDD   F1, F2, F3
DIVD   F4, F1, F2
SUBD   F2, F5, F6
```

3C) Explain how branch target buffer works. (**2 points)**

ASSIGNMENT 4

4A) Explain which of the statements, below, are **not true** and why they are **not true.**
In a two-level **exclusive** memory-hierarchy the following holds:

i) A block in the upper level always exists in the lower level
ii) A block in the upper level cannot exist in the lower level
iii) A block in the lower level does not exist in the upper level
iv) A block in the lower level exists in the upper level

Note: A wrong answer cancels a correct answer. **(2 points)**

4B)
A computer architect wants to establish how many misses in each of the three categories there are for a 4-block, two-way associative cache using LRU and for a 4-block fully associative cache. A program does the following block references: 0 1 2 3 4 0 5 1 8 4 9 5.

Establish the number of cold, capacity (using the OPT replacement policy) and conflict misses for the two caches. (**4 points)**

4C) A computer supports sequential prefetching. Assume that a cache miss for each load instruction is generated every fourth iteration in the code below and that the cache miss penalty is 6 cycles. Which of the cache misses can be cancelled by sequential prefetching in the code below? (**4 points)**

```
LOOP:  LD.D  F1, 0(R1)
       LD.D  F2, 0(R2)
       ADD.D F4, F1,F2
       S.D   F4, 0(R1)
       SUBI R1, R1,#8
       BNEZ R1, LOOP
```

4D) Explain how a miss-handling status register (MSHR) manages to direct a response on a cache miss back to the processor's register file. (**2 points)**

ASSIGNMENT 5

5A) Time-multiplexed multithreading comes in two flavors: Fine-grain multithreading and coarse-grain multithreading. Explain when a switch to a new thread occurs assuming a simple five-stage pipeline for each of the two approaches. (**4 points**)

5B) What structures in a superscalar processor must be replicated to realize a simultaneous multithreaded processor? (**3 points**)

5C) Given an MSI cache coherence protocol, with the three states Modified, Shared and Invalid. Let two processors with their private caches have a copy of block X loaded into their caches. In the access sequence below, X_i=R_i and X_i=W_i, mean a read and a write request to the *same* address X from processor *i,* respectively, where W_i=*C* means that the value C is written by processor *i*. Now consider the following access sequence assuming that X is not present in any cache from the beginning and that X originally contains the value 0:

$W_{1=}1$
R1
R2
$W_{2=}2$
R2

What is the final state of block X in each of the two caches? **(5 points)**

****** GOOD LUCK! ******

Solutions to the exam in DAT105/DIT 051 2021-08-17

ASSIGNMENT 1

1A)

i)

Let's first determine the weights. P1 runs twice as often as P2 which runs twice as often as P3. Hence the weights are 4/7, 2/7 and 1/7 for P1, P2 and P3, respectively.

The execution time of a program on a machine can be determined by using T = IC x CPI x Tc, where IC is given in the first table, CPI = 1 + MPKI x 100 ns/Tc/1000, where MPKI and 1/Tc (=f) are given in the second and third table, resp.

Execution time (seconds)	**P1**	**P2**	**P3**
System A	**2×10^{-6}**	**6×10^{-6}**	**3.75×10^{-6}**
System B	**1×10^{-5}**	**1×10^{-6}**	**1.5×10^{-6}**

A: Weighted average execution time: $(2 \times 4 + 6 \times 2 + 3.75 \times 1) \times 10^{-6}/7 = 3.3 \times 10^{-6}$ seconds
B: Weighted average execution time: $(0.1 \times 4 + 1 \times 2 + 1.5 \times 1) \times 10^{-6}/7 = 0.6 \times 10^{-6}$ seconds

ii)

We must first determine the execution time on the reference machine to calculate the speedup.

For the execution time, we use the same methodology as in i).

Execution time (seconds)	**P1**	**P2**	**P3**
R	**1.5×10^{-6}**	**1.5×10^{-6}**	**1.5×10^{-6}**

The speedup of machine X over R is Execution time(R)/Execution time (X) and is given in the table below:

Speedup over R	**P1**	**P2**	**P3**
System A	**0.75**	**0.25**	**0.4**
System B	**15**	**1.5**	**1**

Harmonic means of the speedup for A: $3/(1/0.75 + 1/0.25 + 1/0.4) = 0.4$
Harmonic means of the speedup for B: $3/(1/15 + 1/1.5 + 1/1) = 1.7$

iii)

Geometric mean speedup is defined as SQRT^n(s1xs2x…xsn) where SQRT^n is the n:th square.

Geometric mean speedup for A: SQRT^3(0.75x0.25x0.4) = 0.42
Geometric mean speedup for A: SQRT^3(15x1.5x1) = 2.8

1B)

Use Amdahl's Law $\lim_{n\to\infty} 1/(0.25 + 0.75/n) = 4$

ASSIGNMENT 2

2A)

```
I1: LOOP:     LD       R4, 0(R1)
I2:           MUL      R5, R4, R4
I3:           SD       R5, 0(R1)
I4:           ADDI     R1, R1, 8
I5:           BNE      R1, R2, LOOP
```

RAW: I1->I2; I2->I3; I4->I5
WAR: I3->I4
WAW: None

2B)

RAW: I1->I2; I2->I3; I4->I5

I1->I2 will lead to loss of a single cycle because the operand cannot be forwarded until the end of the memory stage. I2->I3 will also lead to a single cycle loss because multiply will extend to the end of the memory stage and can first be forwarded after that. I4 ->I5 does not lead to any loss because of forwarding.

WAR: I3->I4

Instructions read their operands in the D stage in execution order so I4 cannot write to R1 before I3 has read R1.

2C) Here is the loop unrolled four times where all lost cycles due to RAW hazards have been eliminated.

```
LOOP:   LD R4,0(R1)
        LD R6,8(R1)
        LD R7,16(R1)
        LD R8,24(R1)
        MUL R5,R4,R4
        MUL R9,R6,R6
        MUL R10,R7,R7
        MUL R11,R8,R8
        SD R5, 0(R1)
        SD R9, 8(R1)
        SD R10, 16(R1)
        ADDI R1,R1,32
        BNE R1,R2, LOOP
        SD R10, -16(R1)
        SD R11, -8(R1)
```

2D)

A branch predictor consists of a table that is indexed by the program counters (or a subset of the bits in it) in which a branch prediction is stored. A 2-bit predictor will predict untaken for say states 00 and 01 and taken for 10 and 11. Starting with state 00, on a correct prediction no state change happens. On a misprediction a transition to state 01 happens and the branch is still predicted as untaken. Hence, two mispredictions need to happen in a row for the predictor to predict taken.

ASSIGNMENT 3

3A)

The pipeline diagram:

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13
O1	Iss	Ex	Ex	CDB									
O2		Iss	Iss	Iss	Ex	Ex	Ex	Ex	Ex	CDB			
O3			Iss	Iss	Iss	Iss	Iss	Iss	Iss	Iss	Ex	Ex	CDB
O4				Iss	Iss	Ex	Ex	CDB					

1. O1 flows through the pipeline without encountering any hazards
2. O2 has a RAW hazard with respect to O1 and cannot execute until Cycle 5 when the result from O1 is broadcast over the CDB.
3. O3 has a RAW hazard with respect to O2 and cannot execute until Cycle 11.

4. O4 is independent of previous instructions and can start executing directly since WAW (O4 with respect to O2) hazards are resolved through renaming. Note that O2 will not write back to the registerfile as register F4 is linked to the destination operand of O4.

3B)

The reorder buffer (ROB) is the main mechanism to keep track of register values when instructions are speculatively executed. It buffers speculatively executed instructions **in the order they appear in the program,** that is, in program order. Each entry has room for the status of each instruction whether it is speculatively executed or committed. When an instruction is committed, it will be removed from the reorder buffer in the next cycle. When an instruction is speculatively executed, the entry also contains the value of the destination register, if it is available.

Now, when the branch instruction is validated as correctly predicted, it will be removed from the ROB in the next cycle. This happens, according to the assumptions, when the last instruction has been executed.

Let's make a pipeline diagram to track the execution of the last three instructions:

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11
O1	Issue	Exec	Exec	CDB							
O2		Issue	Issue	Issue	Exec	Exec	Exec	Exec	Exec	CDB	
O3			Issue	Exec	Exec	CDB					

We note that from the point the branch instruction has retired from the ROB (in C7 according to the assumptions), it takes another **four cycles** until the result is written back to the registerfile.

3C) Explain how a two-level branch target buffer works. (**4 points)**

A branch target buffer stores the target address for branches so that taken branches can start executing already in the instruction fetch stage. It is in the IF stage and is indexed by the program counter.

ASSIGNMENT 4

4A)

i) A block in the upper level always exists in the lower level (**WRONG** because in an exclusive cache a block can only be in one level)

ii) A block in the upper level cannot exist in the lower level (**CORRECT**)

iii) A block in the lower level does not exist in the upper level (**CORRECT**)

iv) A block in the lower level exists in the upper level

(**WRONG** because in an exclusive cache a block can only be in one level)

4B)

Cold misses: All unique block references: 0,1,2,3,4,5,8 and 9: **8 misses**
Capacity misses: same as misses in a fully associative cache with OPT as replacement policy minus number of cold misses.
Missing references: 0 1 2 3 4 (replaces 2) 5 (replaces 3) 8 (replaces 0) 9 (replaces 1): **8 misses**
Hence 8-8 = 0 capacity misses
Conflict misses: same as misses in a two-way associative cache with LRU minus the number of capacity and cold misses.
Missing references: 0 1 2 3 4(replaces 0) 0 (replaces 2) 5 (replaces 1) 1 (replaces 3) 8 (replaces 0) 4 (replaces 0) 9 (replaces 1) 5 (replaces 9): **12 misses**
Hence 12 – 8 – 0 = <u>**4 conflict misses**</u>.

4C)

Sequential prefetching will bring in two blocks: the one accessed and the consecutive one, on a miss. For an iteration in which the load instructions miss, the next consecutive blocks will be available 6 cycles later. As there are 6 instructions between two consecutive loads, the prefetched blocks arrive on time.

4D)

The memory address for the request and the destination register is stored in a MSHR. Hence, on the response from memory, the memory address is used as a key to locate the MSHR that contains the destination register identity so the operand can be inserted there.

ASSIGNMENT 5

5A)

In fine-grain multithreading a thread switch happens every clock cycle whereas in coarse-grain multithreading a thread switch happens when a long-latency event is encountered such as a last-level cache miss.

5B)

Assuming a multiple-issue superscalar pipeline, multiple program counters and multiple branch predictors and branch target buffers are needed. In addition, the register file must also be replicated.

5C)

Let's denote the state as a tuple (S1, S2) where Si is the state of the block in the cache of processor i.

$W_{1=}1$. (M,I)
R1
R2 (S,S)
$W_{2=}2$ (I,M)
R2

Hence, the state of the block in processor 1’s cache is Invalid whereas the block in processor 2’s cache is in state Modified.