CHALMERS

2019-01-04

Exam in DAT 105 (DIT 051) Computer Architecture

Time: January 19, 2019 14 – 18

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

Supporting material/tools: Chalmers approved calculator.

Exam Review: On January 31, 2019 10-12 in Room 4128

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

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

The tables below show the CPI assuming a single-cycle memory system (CPI_0) and number of Misses-Per-Kilo-Instructions (MPKI) on two machines (A and B) and a reference machine (R) with the **same** Instruction Set Architecture (ISA) for two single-threaded programs, P1 and P2, respectively, where P1 executes twice as many instructions as P2 which executes 1 million instructions. The operating frequencies of the three machines (A, B and R) are also shown. The miss penalty is 100 nanoseconds for all three machines.

Program P1	**A**	**B**	**R**
CPI_0	0.5	1.0	1
MPKI	10	10	1
Program P2	**A**	**B**	**R**
CPI_0	2.0	1.5	1
MPKI	10	10	1

Clock freq. (GHz)	
Machine A	**1**
Machine B	**1.2**
Machine R	**1**

1A) Calculate the execution times for P1 and P2 on A, B and R (**4 points)**

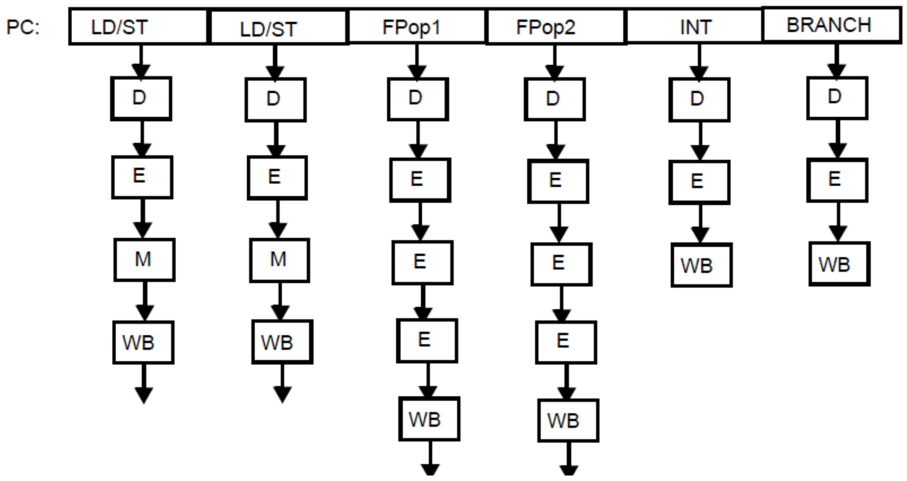
1B) Determine which of the machines is the fastest using **geometric means** (**4 points)**

1C) Assume that 10% of the execution time of a program is spent on executing the division instruction. What is the speedup obtained on *the entire program* if an ingenious design of a division functional unit would result in a speedup of division instructions by a factor 1000? **(2 points)**

1D) Why is arithmetic means of the execution time unreliable as a metric to compare performance of two machines? (**2 points)**

ASSIGNMENT 2

We consider in this assignment a VLIW architecture that can issue two memory, two floating-point, one integer, and one branch instruction each cycle according to the pipeline organization below. There are no forwarding units.



2A) Consider the following code:

```
LOOP: LD F1,0(R1)
      ADD F4, F0,F2
      SD F4,0(R1)
      SUBI R1,R1,#8
      BNE R1, R2, LOOP
```

Show how the instructions in one iteration of the loop above should be scheduled on the VLIW pipeline to maximize instruction level parallelism. (**2 points)**

2B) Consider again the code in Assignment 2A. Use loop unrolling seven times to statically schedule the code to maximize instruction level parallelism. (**6 points)**

2C) Consider the following code:

```
for (i = 0; i < N; i++)
   {A[i+2] = A[i] + 2;
    B[i] = A[i+2] + 1;}
```

This is translates into

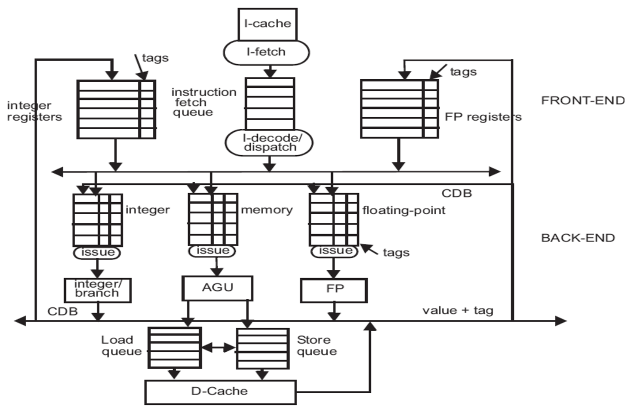
```
Loop:     L.S F0, 0(R2)          ; –O1
          ADD.S F3, F0, F1       ; –O2
          ADD.S F4, F3, F1       ; –O3
          S.S. F3, 16(R2)        ; –O4
          S.S. F4, 0(R3)         ; –O5
```

```
ADDI R2,R2, 8
ADDI R3,R3, 8
BNE R2,R4, Loop
```

Show the code for the kernel in a software pipelined version of the code that maximizes instruction level parallelism. **(4 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 a two-level branch predictor works. (**4 points)**

ASSIGNMENT 4

4A) A computer architect wants to establish the relative performance between a system with a blocking and a non-blocking cache using software prefetching. In software prefetching, prefetch instructions are appropriately inserted in the code shown below. Assume that five instructions are executed per iteration where the CPI for each instruction is one assuming that it doesn't cause a cache miss. On the other hand, if the instruction causes a cache miss, CPI is 100. In addition, prefetch instructions result in CPI=2. Both caches have a block size of four words.

```
for (i=0; i<1000; i++)
     C+=A[i];
```

- Show how the code is annotated with prefetch instructions to hide all cache misses.
- How many MSHRs are needed to make software prefetching maximally effective?
- How much faster does the program run on the system with a non-blocking cache using software prefetching?

A convincing explanation that is easy to follow is needed for full points. (**6 points)**

4B)

In the table below, we show MPKI (Misses-Per-Kilo-Instruction) for a number of organizations. The fraction of memory instructions, out of all executed instructions, is 10%. From the data, determine the cold miss rate, the capacity miss rate and the conflict miss rate for a direct-mapped cache. (**3 points)**

Cache organization	**MPKI**
16-KB direct mapped cache	10
16-KB 2-way assoc. cache	8

16-KB fully associative cache	6
Cache with infinite size	2

4C) How does sequential prefetching work? **(3 points)**

ASSIGNMENT 5

5A) Consider a multicore system comprising a number of processors (cores) on a chip that are connected to a single-level private cache. The private caches use the *write-through* write policy. X_i=R_i and X_i=W_i, mean a read and a write request to the *same* address from processor *i,* respectively, where W_i=*C* means that the value C is written by processor *i*. Now consider the following access sequence:

R_1

R_2

W_1=0

W_2=1

R_1

R_2

What is returned by the second read operation from processor 1 and what is the reason that the correct value is not returned given the cache write policy assumed? How can we modify the cache controller to make sure that the right value is returned?
(6 points)

5B)
Assume a write-back cache and the MSI-protocol. What bus transaction will cause a state transition from state I to state S and what transaction will cause a transition from state S to state I? **(2 points)**

5C) Explain the concept of interleaved (fine-grain) multithreading. Consider a five-stage pipeline. What additional mechanisms and pipeline stages must be added to support interleaved multithreading? How many cycles are lost on a thread switch? **(4 points)**

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

Solutions Exam 2019-01-19

ASSIGNMENT 1

1A) Calculate the execution times for P1 and P2 on A, B and R (**4 points)**

Execution times P1

- $E_{A,P1}$ = IC x (CPI_0 + MPKI x MP/10^3) x Tc. IC=2 x 10^6, CPI_0=0.5, MPKI=10, MP=100ns/1ns=100, T_c=1ns. $E_{A,P1}$= 2 x 10^6 x (0.5 + 10x100/10^3) x 1ns = 2 x 10^6 x (0.5+1) x 1 ns = **<u>3 ms</u>**
- $E_{B,P1}$ = IC x (CPI_0 + MPKI x MP/10^3) x Tc. IC=1 x 10^6, CPI_0=1, MPKI=10, MP=100ns/0.83ns=120, T_c=0.83ns. $E_{B,P1}$= 1 x 10^6 x (1 + 10x120/10^3) x 0.83ns = 1 x 10^6 x (1+1.2) x 0.83 ns = **<u>1.8 ms</u>**
- $E_{R,P1}$ = IC x (CPI_0 + MPKI x MP/10^3) x Tc. IC=1 x 10^6, CPI_0=1, MPKI=1, MP=100ns/1 ns=100, T_c=1 ns. $E_{R,P1}$= 1 x 10^6 x (1 + 1x100/10^3) x 1 ns = 1 x 10^6 x 1.1 x 1 ns = **<u>1.1 ms</u>**

Execution times P2

- $E_{A,P2}$ = IC x (CPI_0 + MPKI x MP/10^3) x Tc. IC=2 x 10^6, CPI_0=2, MPKI=10, MP=100ns/1ns=100, T_c=1ns. $E_{A,P2}$= 2 x 10^6 x (2 + 10x100/10^3) x 1ns = 2 x 10^6 x (2+1) x 1 ns = **<u>6 ms</u>**
- $E_{B,P2}$ = IC x (CPI_0 + MPKI x MP/10^3) x Tc. IC=1 x 10^6, CPI_0=1.5, MPKI=10, MP=100ns/0.83ns=120, T_c=0.83ns. $E_{B,P2}$= 1 x 10^6 x (1.5 + 10x120/10^3) x 0.83ns = 1 x 10^6 x (1.5+1.2) x 0.83 ns = **<u>2.2 ms</u>**
- $E_{R,P2}$ = IC x (CPI_0 + MPKI x MP/10^3) x Tc. IC=1 x 10^6, CPI_0=1, MPKI=1, MP=100ns/1 ns=100, T_c=1 ns. $E_{R,P2}$= 1 x 10^6 x (1 + 1x100/10^3) x 1 ns = 1 x 10^6 x 1.1 x 1 ns = **<u>1.1 ms</u>**

1B) Determine which of the machines is the fastest using **geometric means** (**4 points)**

Solution:

- **A:** Speedup(P1 vs R) = 3/1.1 = 2.7. Speedup(P2 vs R) = 6/5.5 = 1.1 Geommean=(2.7 x 1.1)$^{1/2}$ = 2.5

- **B:** Speedup(P1 vs R) = 1.8/1.1 = 1.6. Speedup(P2 vs R) = 2.2/1.1 = 2 Geommean=$(2.5x\ 2)^{1/2}$ = 2.2

Hence, A (Geommean = 2.5) is faster than B (Geommean = 2.2)

1C) Assume that 10% of the execution time of a program is spent on executing the division instruction. What is the speedup obtained on *the entire program* if an ingenious design of a division functional unit would result in a speedup of division instructions by a factor 1000? (**2 points)**

Speedup with enhanced functional unit:

SP = 1/(0.9 + 0.1/1000) = 1.1 (about 10% faster). This is a good example of not spending significant engineering efforts on a rare case.

1D) Why is arithmetic means of the execution time unreliable as a metric to compare performance of two machines? (**2 points)**

Arithmetic means are sensitive to outliers. So, if one computer outperforms another one for nine out of ten applications but underperforms significantly for one, arithmetic means may suggest that the one that is performing best in most of the cases performs worse.

ASSIGNMENT 2

2A) Consider the following code:

```
LOOP: LD F1,0(R1)
      ADD F4, F0,F2
      SD F4,0(R1)
      SUBI R1,R1,#8
      BNE R1, R2, LOOP
```

Show how the instructions in one iteration of the loop above should be scheduled on the VLIW pipeline to maximize instruction level parallelism (ILP). (**2 points)**

Solution:

LD/ST	LD/ST	FPop1	FPop2	INT	BRANCH
LD F1,0(R1)					
		ADD F4, F0,F2			
SD F4,0(R1)				SUBI R1,R1,#8	
					BNE R1, R2, LOOP

Not much ILP is exploited.

2B) Consider again the code in Assignment 2A. Use loop unrolling seven times to statically schedule the code to maximize instruction level parallelism. (**6 points)**

See pipeline diagram of Figure 3.19 in the textbook, on page 144

2C) Consider the following code:

```
for (i = 0; i < N; i++)
    {A[i+2] = A[i] + 2;
     B[i] = A[i+2] + 1;}
```

This translates into

```
Loop:       L.S F0, 0(R2)           ; –O1
            ADD.S F3, F0, F1        ; –O2
            ADD.S F4, F3, F1        ; –O3
            S.S. F3, 16(R2)         ; –O4
            S.S. F4, 0(R3)          ; –O5
            ADDI R2,R2, 8
            ADDI R3,R3, 8
            BNE R2,R4, Loop
```

Show the code for the kernel in a software pipelined version of the code that maximizes instruction level parallelism. **(4 points)**

See schedule in Table 3.26 on page 150 in the textbook.

ASSIGNMENT 3

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
```

Solution:

The pipeline diagram:

	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					
O4				Issue	Issue	Exec	Exec	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 and O4 are independent of previous instructions and can start executing directly since WAR (O3 with respect to O2) and 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) 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          ;- O1
DIVD   F4, F1, F2          ;- O2
SUBD   F2, F5, F6          ;- O3
```

Solution:

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 predictor works. **(4 points)**

See textbook, page 121 with respect to Figure 3.19.

ASSIGNMENT 4

4A) A computer architect wants to establish the relative performance between a system with a blocking and a non-blocking cache using software prefetching. In software prefetching, prefetch instructions are appropriately inserted in the code shown below. Assume that five instructions are executed per iteration where the CPI for each instruction is one assuming that it doesn't cause a cache miss. On the other hand, if the instruction causes a cache miss, CPI is 100. In addition, prefetch instructions result in CPI=2. Both caches have a block size of four words. In the non-blocking cache, there are 8 miss-status-holding registers.

```
for (i=0; i<1000; i++)
     C+=A[i];
```

- Show how the code is annotated with prefetch instructions to hide all cache misses.
- How many MSHRs are needed to make software prefetching maximally effective?
- How much faster does the program run on the system with a non-blocking cache using software prefetching?

A convincing explanation that is easy to follow is needed for full points. (**6 points)**

Solution:

- Show how the code is annotated with prefetch instructions to hide all cache misses.

According to the assumptions a loop iteration containing a prefetch instruction (CPI=2) and five other instructions (CPI=1) takes 7 cycles assuming that all load instruction hits. However, one of the five instructions is a load instruction and since the block size is four words, the load instruction in every fourth iteration misses. A cache miss takes 100 cycles. The question is how many iterations in advance we must launch a prefetch instruction.

$i \times 7 > 100 \Leftrightarrow i = 15$

```
for (i=0; i<1000; i++){
     prefetch(A[i+15]
     C+=A[i];

}
```

- How many MSHRs are needed to make software prefetching maximally effective?

We need 15 MSHRs because there are 15 outstanding prefetch instructions. However, since only every fourth load will miss we could optimize the program to launch a prefetch instruction only once every fourth iteration. This is however beyond the scope of the assignment.

- How much faster does the program run on the system with a non-blocking cache using software prefetching?

The program without prefetching:

Consider four iterations as there is a miss every fourth iteration:

Four iterations execute 4 x 4 + 3= 19 instructions that take a single cycle and 1 load instruction that takes 100 cycles. In total: 119 cycles

The program with prefetching:

Four iterations execute 4 x 5 = 20 instructions and four prefetch instructions that take a single cycle and 4 prefetch instructions that take 8 cycles. In total: 28 cycles

4B)

In the table below, we show MPKI (Misses-Per-Kilo-Instruction) for a number of organizations. The fraction of memory instructions, out of all executed instructions, is 10%. From the data, determine the cold miss rate, the capacity miss rate and the conflict miss rate for a direct-mapped cache. (**3 points)**

Cache organization	**MPKI**
16-KB direct mapped cache	10
16-KB 2-way assoc. cache	8
16-KB fully associative cache	6
Cache with infinite size	2

Solution:

Let's calculate the miss rates. In one thousand instructions 10% are memory instructions, that is, 100. So in 100 instructions an infinite-sized cache experience 2 misses. This is a miss rate of 2%. So we have the following miss rates:

Cache organization	**MR**
16-KB direct mapped cache	10
16-KB 2-way assoc. cache	8
16-KB fully associative cache	6

Cache with infinite size	2

The cold miss rate is 2%, the capacity miss rate is (6-2=) 4% and the conflict miss rate is (10-4 -2=) 4%.

4C) How does sequential prefetching work? **(3 points)**

See textbook, page 209

ASSIGNMENT 5

5A) Consider a multicore system comprising a number of processors (cores) on a chip that are connected to a single-level private cache. The private caches use the *write-through* write policy. X_i=R_i and X_i=W_i, mean a read and a write request to the *same* address from processor *i*, respectively, where W_i=*C* means that the value C is written by processor *i*. Now consider the following access sequence:

R_1
R_2
$W_{1=}0$
$W_{2=}1$
R_1
R_2

What is returned by the second read operation from processor 1 and what is the reason that the correct value is not returned given the cache write policy assumed? How can we modify the cache controller to make sure that the right value is returned?
(6 points)

Solution:

In a write-through cache, the memory is updated but not the content of any cache that has a copy of that block. The first two reads from P1 and P1 creates copies of the original content of the block. The next two writes, from P1 and P2 updates their respective private caches and memory with the values 0 and 1, from P1 and P2, respectively, in that order. Finally, the read from P1 returns the 0, not 1, which is incorrect as the last write, in the order, writes 1, not 0, breaking the correctness. The first write, by P1, should have invalidated the block in P2's cache.

5B)
Assume a write-back cache and the MSI-protocol. What bus transaction will cause a state transition from state I to state S and what transaction will cause a transition from state S to state I? **(2 points)**

See textbook on page 254.

I->S: Any cache miss will result in a read request on the interconnect (bus in this case) that will return an up-to-date copy from memory.

S->I: This state transition is triggered by another processor that writes to a block that is present (Upgrade) or not present (BusRdX) in its cache. It will result in invalidating the local copy in the cache resulting in downgrading the block to Invalid (I).

5C) Explain the concept of interleaved (fine-grain) multithreading. Consider a five-stage pipeline. What additional mechanisms and pipeline stages must be added to support interleaved multithreading? How many cycles are lost on a thread switch? **(4 points)**

See textbook, pages 438-439 in the textbook.