CHALMERS

2019-08-07

Exam in DAT 105 (DIT 051) Computer Architecture

Time: August 20, 2019 14 – 18 (in SB)

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

Supporting material/tools: Chalmers approved calculator.

Exam Review: On September 2 10-12 after appointment with Per Stenström

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) and Miss Penalties (MP) on two machines (A and B) 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 two machines (A and B) are also shown. The miss penalty is 100 nanoseconds for all three machines.

Program P1	**A**	**B**
CPI_0	0.5	1.0
MPKI	10	5
MP	100	200
Program P2	**A**	**B**
CPI_0	2.0	1.5
MPKI	5	10
MP	200	100

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

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

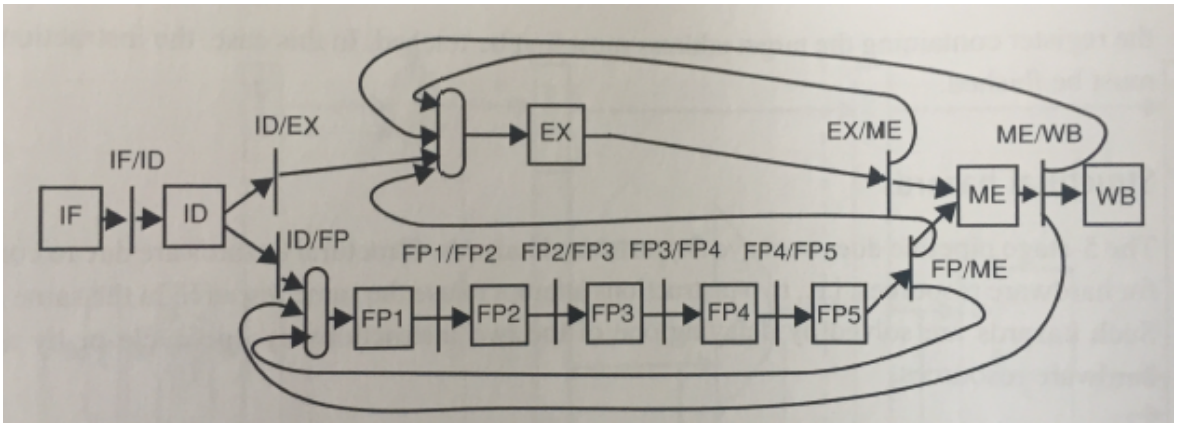
1B) Determine which of the machines is the fastest using **arithmetic means** (**4 points)**

1C) Determine which of the machines is the fastest using **arithmetic means** if the memory system is not having any (negative) impact on performance (**2 points)**

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

ASSIGNMENT 2

We consider in this assignment a pipeline with a 5-stage pipelined floating-point unit and a single-stage execution unit that executes integer, load/store and branch instructions. There are forwarding units from the output of each execution unit and from the memory stage.



2A) What is the operation latency and initiation interval of a) an integer ADD b) a floating-point Load and c) a floating-point ADD instruction? (**3 points)**

2B) Consider the following code:

```
LOOP: LD F1, 0(R1)
      ADD F4, F0, F1
      SD F4, 0(R1)
      ADDI R1, R1,#8
      SUBI R3, R3,#1
      BNE R1, R2, LOOP
```

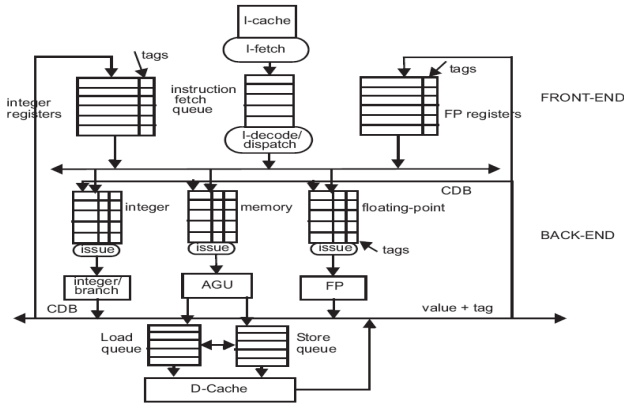
Construct a pipeline diagram that shows in what cycle every instruction enters a particular stage and use it to establish the execution time of a single iteration of the loop above.
(**3 points)**

2C) Now reorder instructions in a single iteration to minimize the stall cycles due to data hazards and establish the speedup in comparison with 2B). (**3 points)**

2D) Consider again the code in Assignment 2A. Use software pipelining to statically schedule the code to maximize instruction-level parallelism. Show the prologue, kernel and epilogue code for the software pipelined loop (**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 1 cycle to carry out an addition/subtraction and 5 cycles to carry out a division.



3A) Consider the program below

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

What data hazards (RAW, WAR and WAW) exist in this code sequence? **(3 points)**

3B) 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)**

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

3C) Explain how a two-bit branch predictor works. (**3 points)**

ASSIGNMENT 4

4A)

A computer architect wants to establish the relative performance between a system with a blocking and a non-blocking cache for the following program.

```
TOY:      LW R1, 0(R2)
          ADDI R2,R2,#4
          BNE R2,R4,TOY
```

The following holds:

- The cache-block size is 16 bytes (four words)
- The cache hit time is a single cycle and the miss penalty for both caches is 200 cycles.
- CPI=1 for non-memory instructions
- The number of MSHRs is 16

Determine the number of cycles it takes to execute 16 iterations of the loop on the blocking and non-blocking cache. (**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	20
16-KB 2-way assoc. cache	16
16-KB fully associative cache	12
Cache with infinite size	4

4C) How does software 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
W_1=3
W_2=4
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 S to state M and what transaction will cause a transition from state S to state M? **(2 points)**

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

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

Solutions Exam 2019-08-20

ASSIGNMENT 1

1A)

P1:
A: Exectime = 2 x 10^6 x (0.5 + 0.01 x 100) x 10^{-9} = 3.0 ms
B: Exectime = 2 x 10^6 x (1.0 + 0.005 x 200) x 10^{-9}/1.2 = 3.3 ms

P2:
A: Exectime = 1 x 10^6 x (2.0 + 0.005 x 200) x 10^{-9} = 3.0 ms
B: Exectime = 1 x 10^6 x (1.5 + 0.10 x 100) x 10^{-9}/1.2 = 9.6 ms

1B)

A: (3.0 + 3.3)/2 = 3.15 ms
B: (3.3 + 9.6)/2 = 6.5 ms
Hence, A is fastest.

1C)

P1:
A: Exectime = 2 x 10^6 x 0.5 x 10^{-9} = 1.0 ms
B: Exectime = 2 x 10^6 x 1.0 x 10^{-9}/1.2 = 0.61 ms

P2:
A: Exectime = 1 x 10^6 x 2.0 x 10^{-9} = 2.0 ms
B: Exectime = 1 x 10^6 x 1.5 x 10^{-9}/1.2 = 1.3 ms

A: (1.0 + 2.0)/2 = 1.5 ms
B: (0.61 + 1.3)/2 =0.95 ms

B is now fastest

1D) The geometric means tend to make the means less affected by outliers.

ASSIGNMENT 2

2A)

Operation latency:
Integer ADD: 0
Floating-point ADD: 4

Initiation interval:
Integer ADD: 1
Floating-point ADD: 1 (because it is pipelined)

2B)

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12
LD F1, 0(R1)	IF	ID	EX	ME	WB							
ADD F4, F0, F1		IF	ID	ID	EX	EX	EX	EX	EX	ME	WB	
SD F4, 0(R1)			IF	ID	ID	ID	ID	ID	ID	EX	ME	WB
ADDI R1, R1,#8				IF	ID	EX	ME	WB				
SUBI R3, R3,#1					IF	ID	EX	ME	WB			
BNE R1, R2, LOOP						IF	ID	EX	ME	WB		

2C) Now reorder instructions in a single iteration to minimize the stall cycles due to data hazards and establish the speedup in comparison with 2B). **(3 points)**

LOOP: LD F1, 0(R1)
ADD F4, F0, F1
SD F4, 0(R1)
ADDI R1, R1,#8
SUBI R3, R3,#1
BNE R1, R2, LOOP

We can move the red-marked instructions up between the ADD and the SD

LOOP: LD F1, 0(R1)
ADD F4, F0, F1
ADDI R1, R1,#8
SUBI R3, R3,#1
SD F4, -8(R1)
BNE R1, R2, LOOP

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12

LD F1, 0(R1)	IF	ID	EX	ME	WB							
ADD F4, F0, F1		IF	ID	ID	EX	EX	EX	EX	EX	ME	WB	
ADDI R1, R1,#8			IF	ID	EX	ME	WB					
SUBI R3, R3,#1				IF	ID	EX	ME	WB				
SD F4, 0(R1)					IF	ID	ID	ID	ID	EX	ME	WB
BNE R1, R2, LOOP						IF	ID	EX	ME	WB		

Comparing the two programs, they will execute in the same amount of time. In both cases, all instructions are ready in cycle 12 (C12).

2D)

The dependency distance between the LD and the ADD is one cycles and the dependency distance between the ADD and the SD is four cycles as opposed to one cycle if forwarding could provide the operand in the next cycle.

We refer to the LD, the ADD and the SD as OP1, OP2 and OP3, respectively, in the software pipelining diagram below:

	ITE1	ITE2	ITE3	ITE4	ITE5	ITE6	ITE7	ITE8	ITE9	ITE10
C1	OP1									
C2		OP1								
C3	OP2		OP1							
C4		OP2		OP1						
C5			OP2		OP1					
C6				OP2		OP1				
C7					OP2		OP1			
C8	OP3					OP2		OP1		
C9		OP3					OP2		OP1	
C10			OP3					OP2		OP1
C11				OP3					OP2	
C12					OP3					OP2

Prologue: C1 – C7
Kernel: C8 – C10
Epilogue (portions of it): C11 – C12

ASSIGNMENT 3

3A)

RAW (pairs): (O1,O2), (O2,O4)
WAR (pairs): (O1,O2), (O1,O3), (O3,O4)
WAW (pairs): (O2,O3)

3B)

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

	C1	C2	C3	C4	C5	C6	C7	C8	C9
O1	IS	EX	WR						
O2		IS	EX	EX	EX	EX	EX	WR	
O3			IS	EX	WR				
O4							IS	EX	WR

Note that because of the RAW hazard between O2 and O4, O4 cannot start executing until C8. The operand from O1 is forwarded via the Common Data Bus to O2 in C3. The WAR and WAW hazards are eliminated through register renaming so O3 can start executing immediately. O4 is though delayed due to RAW w.r.t. O2 through F2.

3C)

A two-bit predictor encodes four states (let's call them 1, 2, 3 and 4) with the two bits where the first two states predict untaken and the last two states predict taken. For as long as the prediction is correct it stays in a certain state. It can take up to two mispredictions for the prediction to change.

ASSIGNMENT 4

4A)

Blocking cache:

There will be a miss every four iterations because there are four words in a cache block and every word is accessed. The time to execute four iterations is: 4 x 12 x 1 + 200 cycles = 248 cycles. Hence, 16 iterations take 4 x 248 = 992 cycles.

Non-blocking cache:

The program will not be stalled by the cache because in 16 iterations, there are 16 loads and hence the 16 MSHRs are sufficient. Note that R1 is not needed and hence there is not any RAW hazards. Consequently, 16 iterations take 16 x 3 x 1 = 48 cycles.

Speedup: 992/48 = 20!

4B)

In 1000 instructions there are 1000 instruction fetches and 100 memory accesses (10% of the instructions are memory instructions). Hence, there are in total 1100 memory accesses. The miss rate can now be calculated as the number of misses per thousand instructions (MPKI) divided by the number of memory accesses. This is calculated in the table below:

Cache organization	**MPKI**	**Miss rate**
16-KB direct mapped cache	20	0.018
16-KB 2-way assoc. cache	16	0.015
16-KB fully associative cache	12	0.011
Cache with infinite size	4	0.0036

Cold miss rate (miss rate for an infinite cache): 0.36%
Capacity miss rate (miss rate for a fully assoc. cache minus cold miss rate):
1.1%-0.36% = 0.74%
Conflict miss rate (miss rate of direct mapped – fully assoc): 1.8% - 1.1% = 0.7%

4C)

The ISA comprises a prefetch instruction that like a load specifies the effective address but not the destination register. Hence, it just generates a memory access that brings the corresponding memory block into the cache. The challenge is to schedule the prefetch instruction at a point in time where it can deliver the data in a timely fashion in anticipation of a future access to memory. Often, this is part of cyclic scheduling in loops by having the compiler analyze the loop so that a prefetch instruction can prefetch data sufficiently ahead of time so that the memory access latency can be entirely hidden.

ASSIGNMENT 5

5A)

The write policy is write-through; it updates the cache and the memory on a write. The second read by P1will however only see the writes from P1 and not from P2. Hence, it will return the value written by the second write from P1 (3). This is not the last written value in the sequence. It should have returned 4 – the last written value from P2.

5B) If a block is in the S(hared) state it means that memory is up to date. What brings that block into the M(odified) state is a write to the block. Then an Upgrade message will be sent to memory and to all other caches forcing them to invalidate the block. On a subsequent read miss from some other cache, a Bus read request is posted on the bus. This will be intercepted by the cache having the block in the M(odified) state which will respond with the block in what is called a Flush operation. The new state is then S(hared).

5C) In block or coarse-grain multithreading, a switch to another thread happens when encountering a long-latency operation such as a cache miss. Since that is detected in the memory stage in a five-stage pipeline, all instructions in the stages preceding that stage must be flushed. A thread switch stage is added between the instruction fetch stage