CHALMERS

Exam in DAT 105 (DIT 051, DIT052) Computer Architecture

Time: 25-10-27 at 14:00 – 18:00 Johanneberg

Person in charge of the exam: Angelos Arelakis, Phone: 0763-103557

Supporting material/tools: Chalmers approved calculator

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: Only a Chalmers approved calculator is allowed

NOTE 2: Bonus points from Real-stuff studies and Quizzes will be added to the exam results for approved exams used solely for higher grades and will be kept for one year.

NOTE 3: Answers must be given in English

GOOD LUCK!
Angelos Arelakis

[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.]

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: angelos@chalmers.se

ASSIGNMENT 1

A computer design team at Apple is tasked to come up with a new processor design for the next product that can offer higher performance than the existing product on the market. They have good ideas for how to do that, but the lead architect is on sick leave; thus, you have been called to help them to come back with which option they should choose.

The current product is a multicore architecture with each processor (core) attached to a first-level cache, where all first-level caches are connected to a second-level cache. The second level cache is connected to off-chip memory. AI inference is executed with floating point instructions.

They have analyzed the bottlenecks in the existing product and have found the following:

- Floating-point operations take 10 cycles
- The cache hit time in the first level cache is 2 cycles
- The cache miss penalty from the first level to the second level cache is 20 cycles
- The cache miss penalty from the second level cache to the off-chip memory is 50 cycles
- The miss rate from level one to level two and from level two to the off-chip memory is 10% and 50%, respectively
- The CPI for instructions other than floating-point or memory instructions is 1
- The relative frequency of instructions is 25% floating point, 25% memory instructions
- The operating frequency is 2 GHz

The team has measured the execution times of three programs (P1-P3) on the existing product and found that they are 4, 2 and 6 seconds, respectively.

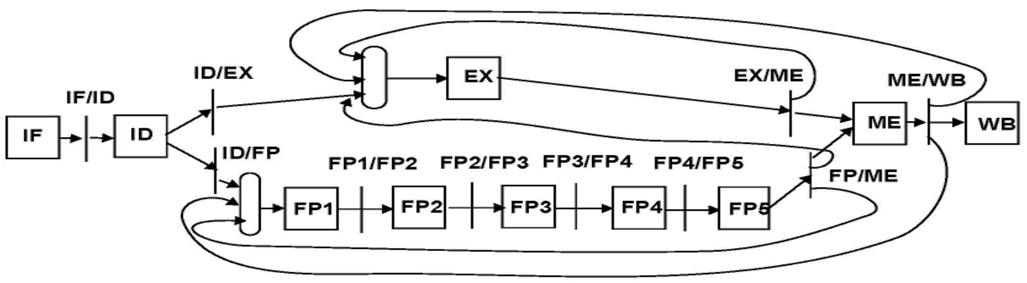
A) What is the execution time of P1-P3 on i) a processor that can speedup AI execution operations by a factor of two and ii) a processor that can speedup the miss-penalty from the second level cache to the off-chip memory by a factor of two. **(8 points)**

B) Determine the best choice of i) and ii) given the geometric means of the two choices. **(2 points)**

C) Which of the two alternatives (i) and ii) would be chosen if we would be able to completely remove the overheads associated with i) floating-point operations and ii) the overhead of memory instructions, respectively? **(2 points)**

ASSIGNMENT 2

We consider in this assignment a pipeline with a 5-stage 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. The pipeline also implements delayed branch meaning one can schedule two instructions in after the branch that will always execute even if the branch is taken.



2A) How many cycles will the second instruction in each pair below have to wait until it can be issued for execution in the case the floating-point unit is pipelined and in the case it is not. Motivate your answer.

i) ADD.D F0,F1,F2 followed by ADD.D F3, F0,F1
ii) ADD.D F0,F1,F2 followed by ADD R1,R2,R3
iii) ADD R5,R2,R3 followed by ADD R4,R5,R2
iv) L.D F5, 1000(R10) followed by ADD.D F5,F4,F3

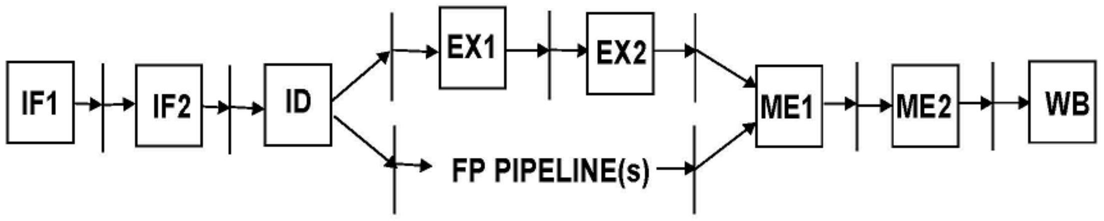
(4 points)

2B) Consider the same pipeline as in **2A)** with the floating-point functional unit *always* being pipelined. Derive the code for an unrolled loop, based on the code below, that can eliminate the cycles lost due to RAW and control hazards. To get all points, the code must correctly deal with register renaming and use the correct displacements for the load (LD) and store (SD) instructions. Initially, R3 contains 100. **(4 points)**

```
LOOP: L.D F1, 0(R1)
      ADD.D F4, F0, F1
      S.D F4, 0(R2)
      ADDI R1, R1,#8
      ADDI R2, R2,#8
      SUBI R3, R3,#1
      BNEZ R3, LOOP
```

2C) The model in 2A) is now changed to cope with higher clock frequencies as follows.

- The IF stage is replaced by two IF stages: IF1 and IF2
- The integer execution unit is replaced by two **pipelined** stages: EX1 and EX2
- The FP pipeline now consists of ten **pipelined** stages
- The ME stage is replaced by two **pipelined** stages



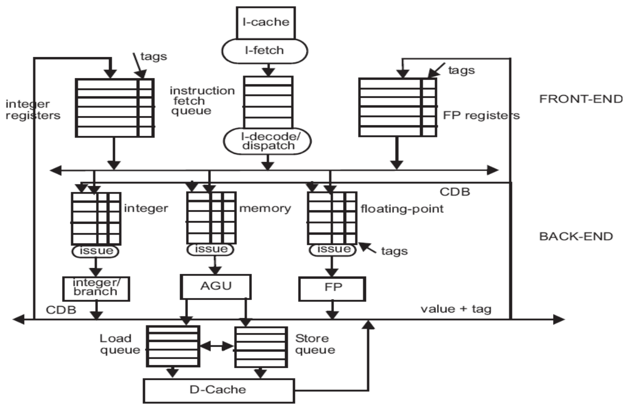
How many cycles will the second instruction in each pair below have to wait before it can be issued from the ID stage for execution in each of the cases, assuming full forwarding.

i) ADD.D F0,F1,F2 followed by ADD.D F3, F0,F1
ii) ADD.D F0,F1,F2 followed by ADD R1,R2,R3
iii) ADD R5,R2,R3 followed by ADD R4,R5,R2
iv) L.D F5, 1000(R10) followed by ADD.D F5,F4,F3

(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, resp. It takes 1 cycle to carry out the condition computation of a branch instruction.



3A) Explain *in detail* what happens in each of the various pipeline stages: Issue (IS), Execute (EX), and Write result (CDB). 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. **(5 points)**

ADD.D F1, F2, F3
DIV.D F4, F1, F2
SUB.D F2, F4, F6

	C1	C2	C3	C4	…	Cn
ADD	IS				…	
DIVD		IS			…	
SUBD			IS		…	

3B) We want to add support for speculative execution. Explain how a reorder buffer and another pipeline stage called COMMIT can allow the following instruction sequence to execute speculatively and what is done in each of the four stages. Show the new table up until the cycle that the branch instruction is on the top of the reorder buffer.

BNEZ R1, LABEL
ADD.D F1, F2, F3
DIV.D F4, F1, F2
SUB.D F2, F4, F6

Assume that the branch instruction is predicted to NOT be taken and that the prediction is NOT correct. **(3 points)**

3C) Find the incorrect statement(s) and for each one, explain why it is incorrect. **(4 points)**

For the dynamically scheduled pipeline with Tomasulo algorithm and speculative execution:
(i) Load instructions are divided into two sub instructions.
(ii) The BPB and BTB mechanisms are added in the fetch stage to predict the outcome of branch instructions.
(iii) The ROB holds the committed values of the registers of the instructions.
(iv) The RAT keeps track of the status of the registers. The status is either completed or committed, and based on the status the hardware knows where it can fetch the correct values from for the registers when it issues the instructions in the pipeline.

ASSIGNMENT 4

4A) A non-blocking cache uses Miss Status Handling Registers (MSHRs) for storing information about outstanding Primary and Secondary Misses. Explain what information is stored in an MSHR entry and the definition of Primary and Secondary misses especially which of them will NOT launch a miss request and WHY. (**3 points)**

4B) The 3C classification model can be used to assess the impact of some of the fundamental cache organizational parameters on performance. i) Explain what miss component is affected by only the cache size and ii) what the role of cache associativity is with respect to the 3C model. (**2 points)**

4C) Software prefetching is typically supported by a prefetch instruction `PF disp(Rx)`, where the argument specifies the effective address as disp + (Rx).

```
LOOP: LW R1,0(R1)
      ADD R3,R2,R1
      SW R3,0(R1)
      ADDI R1,R1,#4
      SUBI R4,R4,#1
      BNEZ R4,LOOP
```

to bring data into cache on time for subsequent iterations of the code using the following assumptions:

- All instructions including the prefetch instruction take a single cycle to execute. Pipeline hazards only happen for:
  - Load-use: 1 extra cycle for cache hits or cache misses; in the case of cache misses, there is extra latency incurred which is the memory access penalty;
  - conditional instructions, e.g., BNEZ: 2 cycles
- Each memory access to a block takes 41 cycles
- There are eight words per block and a word is 4 bytes

Define the PF instruction that will be added by the compiler and describe the reasoning behind.

(**4 points)**

4D) Explain how hardware-based sequential prefetching would be able to launch prefetch requests in the code in **4C)** by explaining how it does so. (**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-back write policy*. R_i and 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:

R_1
$W_1=4$
R_2
$W_2=5$
R_2
R_1

i) What is returned by the two last read operations from processor 1 and 2?
ii) Does this conform with the programmer's expectation?

(2 points)

5B) Explain how a MSI cache coherence protocol will fix the problem in **5A** by defining the finite-state machine with the three states MSI completely? **(6 points)**

5C) A computer architect has as a baseline a simple five stage in-order pipeline with a single functional unit and where data hazards are handled with data forwarding. Explain in detail how the pipeline must be changed in terms of new architectural resources to support *fine-grain (*sometime called *interleaved) multithreading* with four threads. (**4 points).**

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

Solutions to Exam 251027

ASSIGNMENT 1

1A)

Let's first establish how much time that the existing product (denoted R below) spends on
i) Execution of floating-point operations (AI inference)
ii) Handling misses to memory

The execution time of a program on R is
$T = T_{FP}+T_{MEM}+T_{other}$, where T_{FP}, T_{MEM}, and T_{other} are the execution times for handling floating-point, memory accesses and others, respectively.

$T= IC \times CPI \times Tcc = (IC_{FP} \times CPI_{FP} + IC_{MEM} \times CPI_{MEM} + IC_{other} \times CPI_{other}) \times Tcc$ (1)

The fractions of instructions in the three categories are given (FP=25%, MEM=25% and others=50%). For a given program with IC instructions, it holds that

$T= IC \times (0.25 \times CPI_{FP} + 0.25 \times CPI_{MEM} + 0.50 \times CPI_{other}) \times Tcc$ where
$CPI_{other} =1$ according to the assumptions
$CPI_{FP} = 10$ according to the assumptions
$CPI_{MEM}= 2 + MR_{level\text{-}one} \times (MP_{level\text{-}one} + MR_{level\text{-}two} \times MP_{level\text{-}two}) = 2 + 0.1 \times (20 + 0.5 \times 50) = 6.5$
$Tcc = (1 / (2 \times 10^9))$ s = 0.5 nanoseconds.
Given the execution time of the three programs, we can establish the number of instructions executed in each of the three programs:

- P1: Execution time 4 seconds: $IC = T/(0.25 \times CPI_{FP} + 0.25 \times CPI_{MEM} + 0.50 \times CPI_{other}) \times Tcc) = 4 / ((0.25 \times 10 + 0.25 \times 6.5 + 0.5 \times 1) \times 0.5 \times 10^{-9})=1.7 \times 10^9$ instructions
- P2: Execution time 2 seconds: 0.85×10^9 instructions
- P3: Execution time 6 seconds: 2.6×10^9 instructions

Now we can establish the execution time for P1-P3 on a new product that can execute floating point operations twice as fast as the execution time for each of the programs is given.

P1: $T = 1.7 \times 10^9 \times (0.25 \times 10 / 2 + 0.25 \times CPI_{MEM} + 0.50 \times CPI_{other}) \times Tcc= 1.7 \times 10^9 \times (0.25 \times 5 + 0.25 \times 6.5+ 0.5 \times 1) \times 0.5 \times 10^{-9} = 2.87$ seconds
P2: T = 1.43 second
P3: T = 4.39 seconds

For the product that speedups the handling of off-chip accesses by a factor of two we can calculate the $CPI_{MEM} = 2 + 0.1 \times (20 + 0.5 \times 50 / 2) = 5.3$

- P1: $T = 1.7 \times 10^9 \times (0.25 \times 10 + 0.25 \times CPI_{MEM} + 0.50 \times CPI_{other}) \times Tcc = 1.7 \times 10^9 \times (0.25 \times 10 + 0.25 \times 5.3 + 0.50 \times 1) \times 0.5 \times 10^{-9} = 3.68$ seconds
- P2: T = 1.84 second
- P3: T = 5.62 seconds

Hence, it is more important to execute the AI execution operations faster.

[8 points – 4 point for showing a correct methodology, 2 points for showing the formulas, and 2 points for doing the math correctly and find the correct answer]

B)

Let's refer to the current product as R, the proposed system with accelerated AI execution operations as A and the proposed system with accelerated off-chip memory accesses as C and with the execution times of program Pi on the three systems as as $T_{R,Pi}$, $T_{A,Pi}$, and $T_{B,\ Pi}$ respectively. The geometric mean of system A is defined as

$G=((T_{R,P1}/ T_{A,P1})x(T_{R,P2}/ T_{A,P2})x(T_{R,P3}/ T_{A,P3}))^{1/3}= ((4/2.8)x(2/1.4)x(6/4.2))^{1/3}=1.4$

This is the geometric mean performance improvement of speeding up AI operations by a factor of two.

The same methodology can be applied to speed up off-chip memory accesses and the performance improvement will not be as much.
$G=((T_{R,P1}/ T_{C,P1})x(T_{R,P2}/ T_{C,P2})x(T_{R,P3}/ T_{C,P3}))^{1/3}= ((4/3.7)x(2/1.9)x(6/5.6))^{1/3}=1.07$

[2 points – 1 point for each of the two GM calculations]

C)
If we could cancel the performance overhead of floating point operations on A, the execution time for P1-P3 on A would be:

- P1: $T = 1.7 \times 10^9 \times (0.25 \times CPI_{MEM} + 0.50 \times CPI_{other}) \times Tcc = 1.7 \times 10^9 \times (0.25 \times 6.5+ 0.50 \times 1) \times 0.5 \times 10^{-9} = 1.81$ seconds
- P2: T = 0.9 second
- P3: T = 2.76 seconds

On the other hand, if we could cancel the performance overhead of off-chip accesses on B, the execution time for P1-P3 on B would be:

- P1: $T = 1.7 \times 10^9 \times (0.25 \times 10 + 0.50 \times CPI_{other}) \times Tcc = 1.7 \times 10^9 \times (0.25 \times 10 + 0.50 \times 1) \times 0.5 \times 10^{-9} = 2.55$ seconds
- P2: T = 1.28 second
- P3: T = 3.9 second

It is therefore alternative i because all programs execute faster than in ii.

[2 points – 2 points for a correct solution, 0 for an incorrect one]

ASSIGNMENT 2

2A)

i) ADD.D F0,F1,F2 followed by ADD.D F3, F0,F1

Pipelined: Since the two instructions have operand dependences (RAW with respect to F0), they must execute serially and the second instruction has to wait for 4 stall cycles.
Non-pipelined: The second instruction will have to wait for 4 cycles until the first one has left the floating-point unit.

(The table is added for clarification purposes but it is not required in the answer)

Instruction	Pipeline												
	1	2	3	4	5	6	7	8	9	10	11	12	13
ADD.D F0,F1,F2	IF	ID	FP1	FP2	FP3	FP4	FP5	ME	WB				
ADD.D F3,F0,F1		IF	ID	ID	ID	ID	ID	FP1	FP2	FP3	FP4	FP5	ME

ii) ADD.D F0,F1,F2 followed by ADD R1,R2,R3

Pipelined: Since the two instructions have no operand dependences, they can execute in parallel and the second instruction does not have to wait.
Non-pipelined: The second instruction will use the integer unit and the first one will use the floating point unit so no cycles lost

iii) ADD R5,R2,R3 followed by ADD R4,R5,R2

Pipelined: There is a register dependency (RAW hazard) with respect to R5. However, the forwarding will take care of in the integer pipeline so no lost cycles.
Non-pipelined: The instruction pair does not use the floating point unit and is unaffected by whether it is pipelined or not.

iv) L.D F5, 1000(R10) followed by ADD.D F5,F4,F3

Pipelined: The two instructions have a name dependency on the destination register (WAW with respect to F5) however the two instructions follow a different part of the pipeline for the execution stage. The LD takes the top path and the ADD takes the bottom path. However, since the bottom path is longer and the ADD follows the LD, the WAW hazard is guaranteed to be avoided without needing to stall the ADD instruction at all. Hence, 0 cycles of waiting time.
Non-pipelined: Similar to the pipelined version.

[4 points – 1 point for each correct answer]

2B)

The original loop

```
LOOP: L.D F1, 0(R1)        (1) – no hazard but loads F1
      ADD.D F4, F0, F1     (2) – RAW with prev. L.D
      S.D F4, 0(R2)        (5) – RAW with prev. ADD.D
      ADDI R1, R1,#8       (1) – no hazard but modifies R1: 1 cycle
      ADDI R2, R2,#8    (1) – no hazard but modifies R2: 1 cycle
      SUBI R3, R3,#1    (1) – no hazard but modifies R3: 1 cycle
      BNEZ R3, LOOP     (3) – control hazard: 3 cycles
```

The unrolled loop:

```
LOOP: L.D F1, 0(R1)
       L.D F2, 8(R1)
      ADD.D F4, F0, F1
      ADD.D F5, F0, F2
      ADDI R1, R1,#16
      ADDI R2, R2,#16
      SUBI R3, R3,#2
      BNEZ R3, LOOP
      S.D F4, -16(R2)
      S.D F5, -8(R2)
```

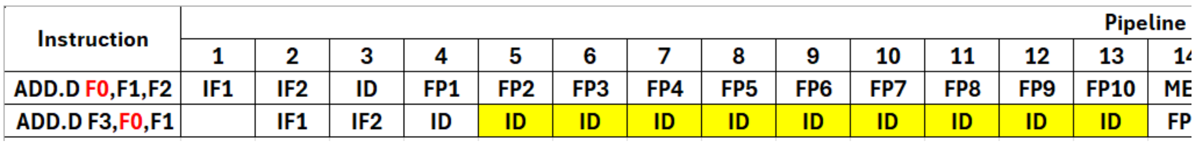
Unrolling the loop twice is enough to remove all hazards assuming that the pipeline implements delayed branches. Otherwise, more unrolling would be needed. Note that we have introduced more registers (F2 and F5), changed displacements (adding 16 instead of eight), adjusted displacements associated with the store instructions to compensate for pushing them down under the ADDI R2… instructions.

[4 points – All points for a perfect solution. Up to 2 points for finding the correct number of unrolled loops and do the code motion correctly]

2C)

i) ADD.D F0,F1,F2 followed by ADD.D F3, F0,F1
There is a RAW hazard with respect to F0. The second instruction must wait for 9 cycles to start executing.

(The table is added for clarification purposes but it is not required in the answer)



Instruction	Pipeline													
	1	2	3	4	5	6	7	8	9	10	11	12	13	1
ADD.D F0,F1,F2	IF1	IF2	ID	FP1	FP2	FP3	FP4	FP5	FP6	FP7	FP8	FP9	FP10	ME
ADD.D F3,F0,F1		IF1	IF2	ID	ID	ID	ID	ID	ID	ID	ID	ID	ID	FP

ii) ADD.D F0,F1,F2 followed by ADD R1,R2,R3
The two instructions work on different register files and there is no data hazard. The second instruction can start executing in the next cycle in both cases.

iii) ADD R5,R2,R3 followed by ADD R4,R5,R2
The second instruction must wait for 1 cycle to resolve the RAW hazard with respect to R5.

(The table is added for clarification purposes but it is not required in the answer)

Instruction	Pipeline												
	1	2	3	4	5	6	7	8	9	10	11	12	13
ADD R5,R2,R3	IF1	IF2	ID	EX1	EX2	ME1	ME2	WB					
ADD R4,R5,R2		IF1	IF2	ID	ID	EX1	EX2	ME1	ME2	WB			

iv) L.D F5, 1000(R10) followed by ADD.D F5,F4,F3
There is a WAW hazard with respect to F5. The L.D uses the upper pipeline where the EX and ME stage have been super-pipelined with 2 EX and 2 ME stages. The subsequent ADD instruction will follow the bottom path which is longer than the super-pipelined EX path, thus similar to the answer of questions 2A, the WAW hazard is guaranteed to be avoided without needing to stall the ADD instruction at all.

[4 points – 1 point for each correct answer]

ASSIGNMENT 3

3A)

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13
ADDD	IS	EX	EX	CDB									
DIVD		IS	IS	IS	EX	EX	EX	EX	EX	CDB			
SUBD			IS	IS	IS	IS	IS	IS	IS	IS	EX	EX	CDB

For I1 there is a tag, say T1, which will be associated with the destination operand (F1). Similarly, the destination operand of I2 (F4) will be assigned a tag (T2). I2 will wait for I1 to get ready by waiting for T1 and data to show up on the CDB. This happens in C4 and I2 will start executing in C5. I3 will be assigned a tag (T3) for F2 to avoid the WAR hazard with respect to I2. In C10, I2 will send T2 along with data on CDB and I3 can start executing in C11 and will reach the Write Result stage in C13.

I1: ADD.D F1, F2, F3
I2: DIV.D F4, F1, F2
I3: SUB.D F2, F4, F6

[5 points – 1 point for explaining the ADD, 2 for the DIV.D and 2 for the SUB.D.]

3B)

	C1	C2	C3	C4

BNEZ	IS	EX	CDB	CT
ADD.D		IS	EX	EX
DIV.D			IS	IS
SUB.D				IS

Branch prediction is done in the instruction fetch stage. Assuming that branch not taken is predicted the subsequent instructions will be speculatively executed until the branch instruction is committed four cycles later. In this case, the branch is predicted not taken so the subsequent instructions will be fetched. Each speculatively executed instruction is inserted in the reorder buffer in the order it appears in the program. When the branch instruction has been executed and has reached the top of the ROB buffer, it will be checked if the prediction was correct or not. This is the COMMIT stage where instructions can **only** commit their results to the register file if the speculation was correct, otherwise if the speculation was wrong the pipeline must be flushed.

In this stage (C4 in the table above), it is discovered that the prediction was wrong thus all instructions afterwards have been misspeculated. This means that the branch instruction including all misspeculated instructions will leave the ROB without committing the results. This is referred to as flushing the pipeline. After flushing the pipeline, the next instruction to execute will be the mispredicted branch that will now be correctly executed.

[3 points – 1 point for showing the correct table, 2 points for providing an answer similar to above].

3C)

(i) Load instructions are divided into two sub instructions

This is incorrect. Load instruction is kept as one instruction as it only needs to calculate the address and read from memory. It is store instructions that are divided into two sub-instructions, one for calculating the address and another for writing the data to memory.

(ii) The BPB and BTB mechanisms are added in the fetch stage to predict the outcome of branch instructions

This is correct.

(iii) The ROB holds the committed values of the registers of the instructions

This is incorrect. The ROB holds the temporary register values of the completed instructions.

(iv) The RAT keeps track of the status of the registers. The status is either completed or committed, and based on the status the hardware knows where it can fetch the correct values from for the registers when it issues the instructions in the pipeline.

This is incorrect. The RAT implements the register renaming and keeps track of where the register values are in the pipeline – committed, completed, or pending. The RAT is installed in the dispatch stage, therefore the hardware knows where it will fetch the correct register values when the instruction is dispatched, not when it is issued.

[4 points – 1 point for each correct answer]

ASSIGNMENT 4

4A)

An MSHR entry contains two types of information: the effective address of the location and the destination register. For example, if LD R5, 100(R6) results in a miss, the effective address 100 + (R6) will be stored as well as R5. A *primary miss* means that the memory instruction does not find the data in the cache or any entry in the MSHR with the same effective address. Then a miss request is launched. On the other hand, if an entry with the same effective address exists, the miss request results in a *secondary miss.*

[3 points – 1 point for describing the MSHR, 1 point for answering with respect to the Primary miss, 1 point for answering with respect to the Secondary miss]

4B)

The cache size will primarily affect the *capacity miss rate.* This component is established by measuring the cache miss rate on a fully-associative cache with an optimal replacement algorithm (OPT). If we reduce the associativity some hits will be converted to misses because blocks that are in high demand conflict with each other for a location in the cache.

[2 points – 1 point for describing the Cache capacity misses, 1 point for describing the impact of associativity]

4C)

Since there are eight words per memory block and they are visited one by one in the first eight iterations, there will be one miss followed by seven hits. We would like to hide the latency of the miss that happen in the nineth iteration and onwards.
Therefore, we need to establish the number of cycles to execute the first eight iterations. Each iteration has six instructions with one cycle each based on the assumptions.
On the top of this, a cache miss takes 41cycles and each load instruction has one load-use hazard cycle.
Finally, the branch instruction needs two extra cycles to resolve the branch target.

In total, the first iteration takes 6+1+41+2=50 cycles.
The next seven iterations will hit in the cache thus take, 6+1+2= nine cycles each.
In total, the eight iterations will take 50 + 7x9 = 113 cycles which is far more than the time to bring in prefetched data.
At steady-state with prefetching this latency shall be reduced to 73 (first iteration: 6+1+2 +1 for the PF instruction, other iterations: 6+1+2; i.e., 10+7x9=73) cycles but that is still large enough to bring the next data.

We now have to determine the effective address for the nineth iteration. For the first iteration, the displacement is 0, then 1 up to 7 for the eight iterations, leading to eight for the nineth iteration. The displacement in PF disp(Rx) should be 8x4 = 32. Hence, one should insert PF 32(Rx) in the first iteration so that it prefetches a block ahead.

[5 points – All 5 points for a fully correct and well-motivated answer. 2 points for showing the methodology to solve this problem.]

4D)

When the load instruction is launched and it hits, there will be no action. However, when it misses, it will trigger a prefetch to request to the next data block. This data will be returned either in a dedicated prefetch buffer or in the cache.

[3 points – All points for answer that covers what happens upon a hit, upon a miss, when prefetch is triggered, what happen in the response.]

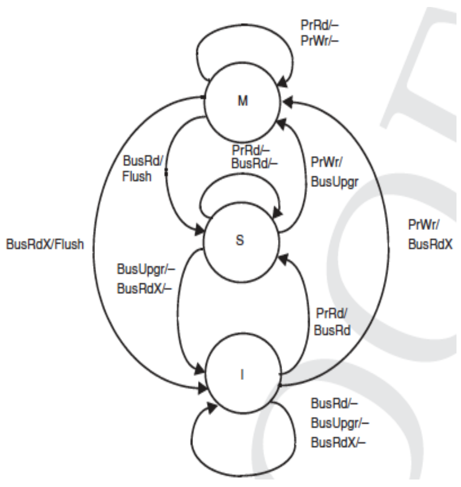
ASSIGNMENT 5

5A)

R_2 will return 5 because it reads from P_2s cache. R_1 will return 4 as it reads from P_1s cache. The programmer's expectation is that the reads will return the latest value which is 5.

[2 points – 1 point for answering correctly each sub-question]

5B)



The finite state machine (FSM) specifies how the MSI protocol works. There are 3 states in the FSM: M (Modified), S (Shared) and I (Invalidated); each state defines the actual state of a cache block. There is one FSM associated with P1's cache and one with P2's.

R1 from P1 will bring the block into state S (Shared) – I->S transition for P1 (PrRd/BusRd). The subsequent write request will bring P1's block into state M (Modified) – S->M transition (PrWr/BusUpgr).
Next, P2 will issue a read request. This will cause an I->S transition for P2 triggering a BusRd request on the bus. This will cause an M->S transition for P1 causing a Flush operation, as a result the data from P1's cache is copied into P2's cache.
Next, P2 will write 5 to the block. P2 will place its own unique copy into state M (S->M with PrWr/BusUpgr). This will further cause an invalidation to P1's copy (S->I).
The read from P2 will return 5 (PrRd and M->M transition).
As for P1's read, it will copy the block from P2's cache and place both FSMs in state S.

With MSI, all reads returns the latest writes.

[6 points – 2 points for providing the MSI FSM. 2 points for explaining what happens in the MSI for P1 and 2 points for explaining what happens in the MSI for P2.]

5C)

A five-stage in-order pipeline with the states IF ID EX MEM and WB will be augmented with a new state TS (for thread switch) that alternately in round-robin switches between the four threads as follows: T0, T1, T2, T3, T0, T1, T2, T3, T0, …, etc. The pipeline must also quadruple the number of program counters – one per thread – and support one register file per thread. Finally, a thread ID must follow the instruction so that in the case of pipeline flushing for a specific thread, only the stages that contain instructions of this thread are flushed.

[4 points – 1 point for each of the 4 modifications]