UPPSALA UNIVERSITET

Department of Information Technology

INSTRUCTIONS

Please check that you have the correct exam!

This sheet should always be submitted, even if you haven't solved any of the exam problems.

Each solution should be written on a separate paper.

Write your exam code on each paper.

Please use only *one side* of the papers and do not use a pencil with red colour.

Sort the solutions in question order, with question 1 first, before you submit them.

FRONT SHEET FOR EXAMS	**DATE:** 2022-05-25
Course name AVDARK 1DT024	
Your exam code	
Semester when you were first registered to the course:	Programme:
Time for submitting the exam:	Table number:

No.	Solved problems (mark with X)	Points earned
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		
11		
12		
13		
14		
15		
16		
17		
18		
19		
20		
21		
22		
23		
24		
25		
26		
27		
28		
29		
30		

Comments from the teacher

Σ

☐ Exam with bonus points: Grade is not shown.[1]

5 ≥ 61 4 ≥ 49 3 ≥ 36

Exam grade:

1 The final result (points including bonus points and grade) can be found in Ladok after certification.

Advanced Computer Architecture, 10.0 c

Course code:1DT024

Semester: Spring 2022

Place: Fyrislundsgatan 80, sal 1
Time: 25th of May (08.00 - 13.00)
Allowed help: Calculator
Language: Answers in English (Swedish can be used if needed, but try to have the main points of your answer also in English)

General information

- There are 18 questions in this exam (+ 1 guest-lecture bonus at the end)
- 12 questions correspond to PARTICIPATION, LAB, or HANDIN bonuses.
- The maximum score is 72p (excluding the guest lecture bonus question).
- You need 36p to pass.
- You will automatically be given maximum points (3p) for the corresponding bonus questions if you have already received the corresponding bonus for the PARTICIPATION, LAB, or HANDIN. You will not have to provide any answers for that question but clearly state that for such a question you claim the bonus you have already received.
- In case you think that you have earned a bonus point that is not accounted for, we suggest that you solve that question (just to be safe), but state in the solution that you think you should have received a bonus point. We will sort that out later…

Grades:
At least 36p → grade 3
At least 49p → grade 4
At least 61p → grade 5

Formatting your Answers

Write your answers in a new document, stating clearly the question number and the sub-part of the question (if applicable) that you are answering.
You need to provide your solutions as a separate PDF file.
Not readable and/or not understandable answers result in zero points.
If any of the questions seem unclear, state your interpretation and assumptions clearly.
Note that some of the questions have several sub-questions (i.e. *"What is 'X' **and** how does it improve 'Y'... ")*. Please answer all of the sub-questions to get the full score.

Good luck!

// Stefanos, Per, Pavlos

1. (Bonus: Lab 1) Memory Optimization (3p)

Blocking can be used to improve the effectiveness of a cache.

a) Give a code example of how this technique is applied (first un-optimized and then optimized). **(2p)**
b) What single locality property (name the important one) is primarily improved using this scheme? **(1p)**

2. (Bonus: Hand-in 1) Caches (3p)

a) Make a drawing of a **physically addressed cache** with the following data

- Cache size: 20 MByte
- Cache line size: 256 Byte
- Organization: 10-way
- CPU word size: 8 Byte (this is the size of the data unit always delivered from the cache to the CPU)
- Physical address size: 48 bits (i.e., 256 TByte of address space can be addressed)

Clearly show the address bits ranges used for the indexing, the comparisons, and multiplex ("mux") selects, etc. Describe, either in words or in logic, the functionality of all logic needed to resolve a read access to the cache. **(2p)**

b) Which bit ranges would change if the cache size was decreased to 10 MByte while all the other parameters stayed the same? State the new bit ranges (you do not need to make completely new drawing) **(1p)**

3. (Bonus: Participation 1) Caches (3p)

a) What is the purpose of the TLB in address translation? **(1p)**
b) Describe how a lookup is performed in a TLB (max 4 sentences). **(1p)**
c) What information is stored in a TLB entry? **(1p)**

4. (Bonus: Lab 2) Synchronization (3p)

a) What is the advantage of a *queue-based* lock compared with a *Test and Test&Set* spinlock? **(1p)**
b) Write the pseudo-code for barrier synchronization, assuming that you have access to the synchronization primitives `lock(L)` and `unlock(L)`. **(2p)**

5. (Bonus: Hand-in 2) Memory ordering (3p)

Dekker's algorithm is described by the following parallel pseudo-code:

Initially:

```
A := 0;
B := 0;
```

Thread 0:

```
A := 1;
if (B==0) then
  print("Thread 0 won");
```

Thread 1:

```
B := 1;
if (A==0) then
  print("Thread 1 won");
```

a) What feature is implemented by Dekker's algorithm? (I.e., what did Dekker use it for?) **(1p)**
b) Under which of the three memory models: *sequential consistency*, *total store order* and/or *weaker consistency*, would it work as Dekker anticipated? **(1p)**
c) How would the code need to be changed in order to work for weaker models? **(1p)**

6. (Bonus: Participation 2) Coherence (3p)

Draw the two state transition diagrams (one for snooping the bus (*) and one for a CPU accessing the cache (**)) for an idealized MOSI cache coherence protocol where each state transition and output signal happens *"atomically"*. For each state transition arc between the coherence states, mark the arc with the input signal (i.e., what caused this transition) and output action (e.g., is Data provided by this cache?) **(3p)**

(*) Snooping transactions: ReadToShare (RTS), ReadToWrite (RTW), Invalidate (INV), Writeback (WB).

(**) CPU accesses: CPUread, CPUwrite, CPUreplacement

7. (Bonus: Lab 3). Scalable MP (3p)

a) What is false sharing and how does it differ from "true" sharing? **(1p)**
b) Under which of the parallel programming paradigms OpenMP, MPI and/or Pthreads, could an application typically experience false sharing and why? **(1p)**
c) How can you remove false sharing from an application? **(1p)**

8. (Bonus: Hand-in 3) Scalable MP (3p)

a) Draw the structure of a Dir_3B directory assuming that it tracks the coherence of 32 caches. Clearly show how many bits each field has. **(2p)**
b) Describe the difference between coherence protocols based on a Dir_3B directory and a Dir_3NB directory. **(1p)**

9. (Bonus: Participation 3) In-Order Pipelines (3p)

A 2-way SMT pipeline and a 2-way super-scalar pipeline can both start the execution of two instructions each cycle. State how they differ in terms of:

a) How they select the instructions to execute. **(1p)**
b) What hardware they need to replicate compared with a 1-way implementation. **(1p)**
c) What kind of *parallelism* they exploit. **(1p)**

10. (Bonus: Lab 4) (3p)

a) According to Flynn's taxonomy, what kind of parallelism is primarily explored with the SSE, MMX and AVX instructions of x86? Write the four-letter acronym and a brief explanation of how it works. **(2p)**
b) What is the other *common parallelism* expressed in Flynn's taxonomy and how does it differ from 10a? **(1p)**

11. (Bonus: Hand-in 4) CPU and Pipelines (3p)

There are three different classes of hazards: *Structural hazards*, *data hazards* and *control hazards*.
Please state which of them (a single one) is <u>mainly</u> removed or alleviated by the following architectural features and briefly describe how.

a) Branch prediction? **(1p)**
b) Register renaming? **(1p)**
c) Superscalar architecture (compared with single-issue architectures)? **(1p)**

12. (Bonus: Participation 4) CPU and Pipelines (3p)

a) Explain the most fundamental technique used by GPUs to hide memory latency? **(1p)**
b) How many threads can a high-performing GPU typically execute at the same time today? (10s, 100s or 1000s) **(1p)**
c) Name two popular "high-level languages" that are commonly used to program GPUs today. **(1p)**

13. Cache Coherence (6p)

All the three RISC CPUs (e.g. cores in a multicore) in a MOSI shared-memory sequentially consistent multiprocessor execute the following code.

CPU1	CPU2	CPU3
A = 1;	**while (A==0) {};**	**while (B==0) {};**
while (C ==0) {};	**B = 1;**	**C = A;**

The CPUs execute ALU instructions (register-to-register) and memory instructions, moving values between memory and registers. There are two kinds of memory instructions in this example: LD and ST.
CPU3 will start slightly ahead of the others and will execute its first memory instruction before CPU2 starts its first memory instruction. CPU1 will start last. Then, the CPUs will take turns and execute one memory instruction each (this is an artificial constraint for exam purposes only).
Initially A, B and C reside only in memory and have the values 0. They belong to different cachelines.
After the parallel execution is done for all of the CPUs, the cachelines that are still in the caches will get replaced by data brought in by some other execution. This activity should also be reflected in your answer.

The following four bus transaction types can be seen on the snooping bus connecting the CPUs (cores):

- RTS: ReadtoShare (reading the data with the intention to read it)
- RTW: ReadToWrite (reading the data and requesting write permission)
- WB: Writing data back to memory
- INV: Invalidating other caches copies

The Solution Sheet below shows every state change and/or value change of A, B and C in each CPU's cache **according to one correct interleaving of the memory accesses**. For each line, the bus transaction that occurs on the bus (if any) as well as which device is providing the corresponding data (if any) are shown.

Unfortunately for you the coherence protocol in this solution is fairly "broken" and introduces a few bugs compared to a proper-functioning protocol. Some (but not all) bus transactions, states, values, and data-providers are wrong. You must find all the bugs and correct them. Strike out the incorrect entry and write beneath it the correct one (make it clear).

CPU action	Bus Transaction (if any)	State/value after the CPU action									Data is provided by [CPU 1, 2, 3 or Mem] (if any)
		CPU1			CPU2			CPU3			
		A	B	C	A	B	C	A	B	C	
Initially		I	I	I	I	I	I	I	I	I	
CPU3: LD B	RTS(B)								S/0		Mem
CPU2: LD A	RTS(A)				S/0						Mem
CPU1: ST A	RTW(A)	S/1			S/1						CPU2
CPU3: LD B	-										-
CPU2: LD A	RTS(A)	O/1			S/1						CPU1
CPU1: LD C	RTS(C)			S/0							Mem
CPU3: LD B	-										-
CPU2: ST B	INV(B)					O/1			I		CPU1
CPU1: LD C	-										-
CPU3: LD B	RTS(B)					M/0			S/1		CPU2
CPU2: -											
CPU1: LD C	-										-
CPU3: LD A	RTS(A)							S/1			Mem
CPU2: -											
CPU1: LD C	-										-
CPU3: ST C	INV(C)			S/1						M/1	Mem
CPU2: -											
CPU1: LD C	RTS(C)			S/1						S/1	CPU3
…											
CPU1: REPL A	-	I									-
CPU1: REPL C	WB(C)			I							CPU1
CPU2: REPL A	WB(A)				I						CPU2
CPU2: REPL B	-					I					-
CPU3: REPL A	-							I			-
CPU3: REPL B	WB(B)								I		CPU3
CPU3: REPL C	-									I	-

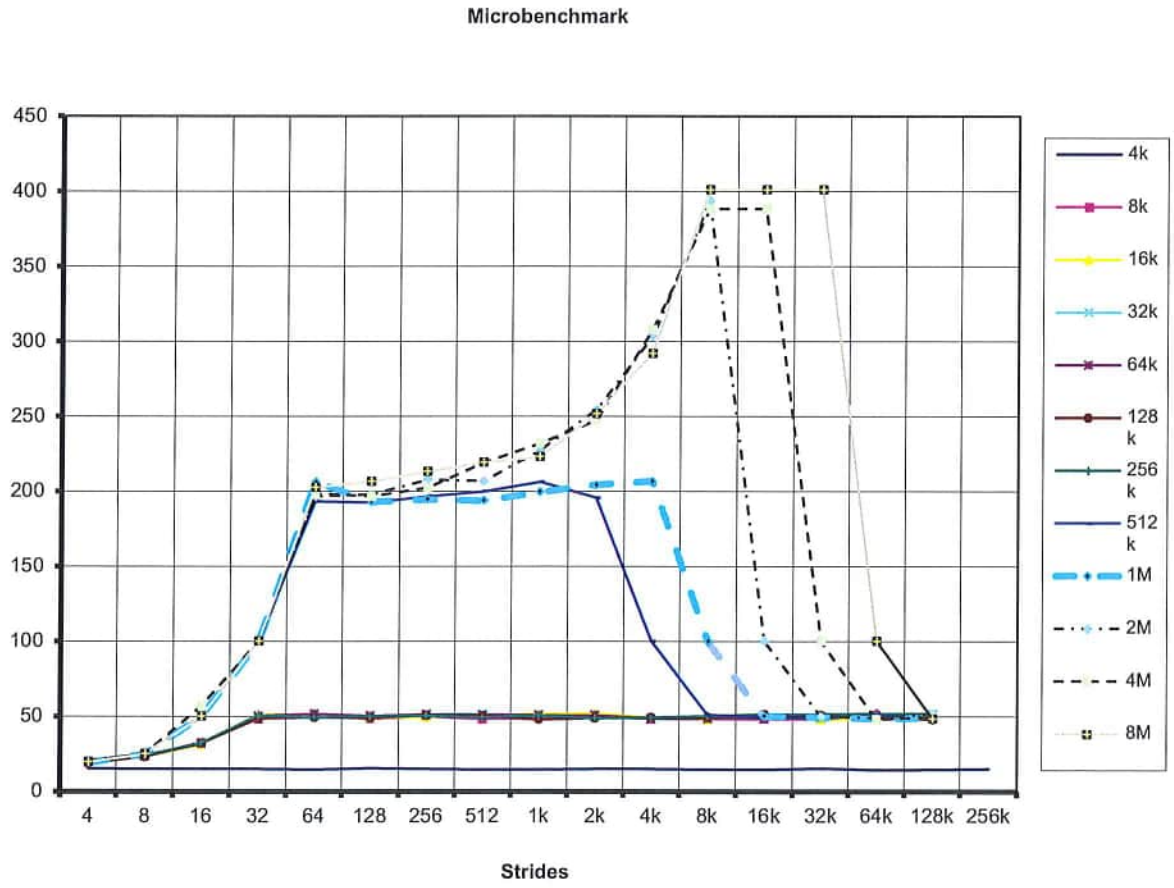
14. Microbenchmarks (6p)

A simple microbenchmark code below is used to draw curves that characterizes the properties of the underlying memory system. Initially we assume that both cache and TLB are fully associative.

```
for (times = 0; times < Max; times++)          /* many times*/
  for (i = 0; i < ArraySize; i = i + Stride)
    dummy = A[i];                              /* touch an item in the array */
```

The curve below shows the average access time to a memory system as a function of Stride for different ArraySizes.

a) What is the cache line size and cache size for the 1st level cache? **(1p)**
b) What is the cache line size and cache size for the 2nd level cache? **(1p)**
c) What is the page size and how many entries does the TLB have? **(2p)**
d) How would the graph change if the TLB was made 8-way set-associative instead of fully associative? **(2p)**



(*) The lowest curve contains only the 4k curve
Curves 8k – 256k look very similar
Curves 512k – 1M look similar between stride=64 and stride = 1k

15. Coherence (6p)

There are 2 parts: a (3p) and b (3p).

One of the simplest protocols for coherence is MSI (Modified, Shared, Invalid). It provides the basic functionality for the Single-Writer-Multiple-Reader Invariant in cache coherence. However, there are protocols with more states, for example protocols that include an Exclusive "E" and/or Owned "O" state.

a. The Shared state "S" says nothing about how many other sharers there might be in the system. In contrast, an Exclusive "E" state implies that it is the only sharer in the system.
   1) How do the M and E states differ?
   2) How do S and E states differ?
   3) From which of the MSI states (Modified, Shared, Invalid) do you reach the E state and with what BUS () and PROCESSOR () transactions? Draw two state diagrams, one for to show exactly these cases (do not bother with irrelevant states or transactions). If a state diagram is empty (no states), indicate it.
   4) Why adding an Exclusive state to MSI, to form the MESI protocol, is useful? Give an example of a situation where a protocol with the E state (i.e., MESI) performs better than the basic MSI.

b. The Modified "M" state implies (among other things) that the cacheline is different than the data in memory. An Owned "O" state also implies that the cacheline is different than the data in memory.
   1) How do the M and the O states differ?
   2) How do the S and O states differ?
   3) From which of the MSI states (Modified, Shared, Invalid) do you reach the O state and with what BUS (BUS_RTS, BUS_RTW, BUS_WB, BUS_INV) and CPU (CPUread, CPUwrire, CPUreplace) transactions? Draw two state diagrams, one for BUS transactions, one for CPU transactions, to show exactly these cases (do not bother showing irrelevant states or transactions). If a state diagram is empty (no states), indicate it.
   4) Why adding an Owned state to MSI, to form the MOSI protocol, is useful? Give an example of a situation where a protocol with the O state (i.e., MOSI) performs better than the basic MSI.

16. Consistency, TSO & DRF (6p)

There are 2 parts: a (3p) and b (3p).

a. TSO (3p)

Consider the following code:

Core 0	Core 1
Initially	x=0; y=0;
`ld ra,y`	`st x,1`
`ld rb,x`	`st y,1`

Program order

There are two variables in memory: `x` and `y`.
Before the code executes both variables are 0.
Core 0 then issues two loads, `ld ra,y` and `ld rb,x`, loading registers `ra` and `rb` with `y` and `x` respectively.
Core 1 issues two stores, `st x,1` and `st y,1`, storing ones in `x` and `y` respectively.

After both cores have executed we can inspect registers `ra` and `rb` (recall that `ra` was loaded with `y` and `rb` with `x`).There are only four possible outcomes we can observe:

case	ra (loaded with y)	rb (loaded with x)	Memory order example that yields this result
1	0	0	(TSO-valid) ld y → ld x → st x → st y
2	0	1	
3	1	0	
4	1	1	

Assume TSO (Total Store Order) as the consistency model.
Recall that in TSO in the memory order we must observe program order ("$^{po}\rightarrow$") in the following cases:
ld y $^{po}\rightarrow$ ld x
st a $^{po}\rightarrow$ ld b
st x $^{po}\rightarrow$ st y

Which one of the above four cases (outcomes) is NOT VALID in TSO? (The first case is given as an example.)
For the invalid outcome, show how TSO is *violated* with a specific (TSO-invalid) memory order example (see the example for a TSO-valid case in the table).

b. DRF (3p)

Consider the following piece of parallel code (two threads T0 and T1):

```
T0          T1

a := 1      b := 1

print b     print a
```

A ***Data Race*** is two conflicting accesses on the same data, without being separated by synchronization. ***Data-Race Free (DRF)*** code is code that does not have data races. The code above is obviously not DRF and your job is to make it DRF using locks.

1. You try the following options from A to D. Which ones are DRF and which ones have data races? Show the data races with arrows.

A

```
T0              T1

a := 1          b := 1

Lock(x)         Lock(y)
print b         print a
Unlock(x)       Unlock(y)
```

B

```
T0              T1

a := 1          b := 1

Lock(l)         Lock(l)
print b         print a
Unlock(l)       Unlock(l)
```

C

```
T0              T1

Lock(l)         Lock(l)
a := 1          b := 1
Unlock(l)       Unlock(l)

Lock(l)         Lock(l)
print b         print a
Unlock(l)       Unlock(l)
```

D

```
T0              T1

Lock(x)         Lock(y)
a := 1          b := 1
Unlock(x)       Unlock(y)

Lock(y)         Lock(x)
print b         print a
Unlock(y)       Unlock(x)
```

2. You show the results to your teammates, but they are not happy. Can you do a DRF version with only one lock and one unlock per thread?

17. OOO (6p)

Assume this simple C code and its pseudocode assembly code:

```
This code writes zeroes and ones in x, a 4x4 matrix of doubles (8-byte) in Row-major
order.
for (i = 0; i < 4 )
   for (j = 0; j < 4) {
       if (j > i)
          x[i,j] = 1;
       else
          x[i,j] = 0;
   }
...
```

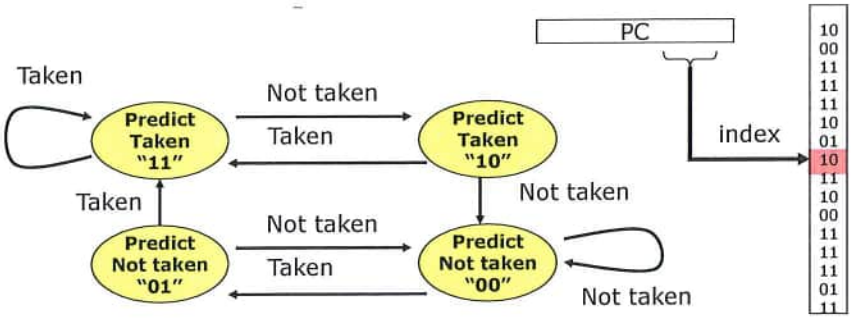
0	1	1	1
0	0	1	1
0	0	0	1
0	0	0	0

Instruction PC (label:)	Instructions				Comment
1 loop:	MULI	R5,	Ri,	#32	R5=i*4*8
2	ADD	R6,	Rj,	R5	R6=j+i*4*8
3	BLTEQ	Rj,	Ri,	if	j <= i ? if
4 else:	ST	R0,	x(R6)		x[j+i*4*8] = 0
5	J	endif			Jump endif
6 if:	ST	R1,	x(R6)		x[j+i*4*8] = 1
7 endif:	ADDI	Rj,	Rj,	#8	j++
8	BLTEQ	Rj,	#24,	loop	j <= 3*8 ? loop
9	ADDI	Rj,	R0,	#0	j=0
10	ADDI	Ri,	Ri,	#8	i++
11	BLTEQ	Ri,	#24,	loop	i <= 3*8 ? loop
12	i1				Instr. 1
13	i2				Instr. 2

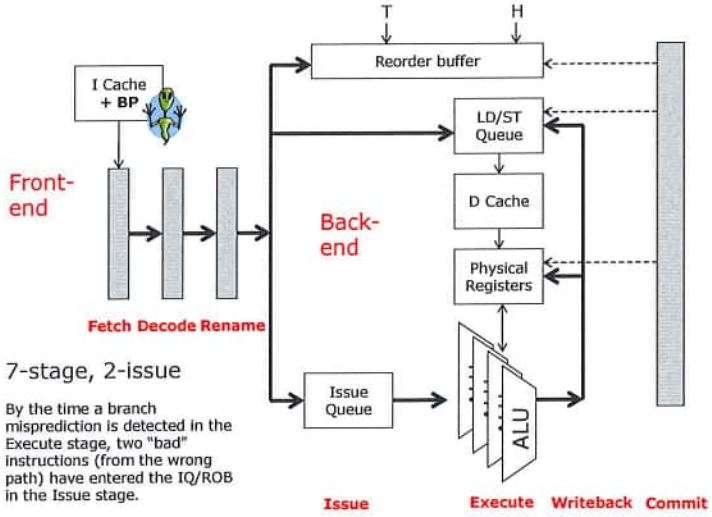
Notes:

2. All registers are 64 bits (8 bytes)
3. Assume `R0 == 0` and `R1 == 1`
4. Assume that before the code, variable i is 0 and assigned to architectural register Ri, variable j is 0 and assigned to Rj.
5. To load x[i,j] you need to do load a register from address $x + (j + i * 4) * 8$
6. Assume the following instructions:
   - `MULI R1, R2, #immediate // R1 = R2 * immediate`
   - `ADD R1, R2, R3 // R1 = R2 + R3`
   - `ADDI R1, R2, #immediate // R1 = R2 + immediate`
   - `LD R1, x(R2)  // load R1 from address x + R2`
   - `ST  R1, x(R2)  //  store R1 to address x + R2`
   - `BLTEQ R1, R2, Label  // goto label if R1 <= R2`
   - `BLTEQ R1, #immediate, Label  // goto label if R1 <= immediate`

Assume the following 2-bit branch predictor. Assume that all entries in the predictor are initially set to 00.



Assume the following dynamic superscalar architecture:



For the first four iterations of the inner loop body, fill in the contents of the reorder buffer (ROB) in the ROB table (next page) using the following rules:

- Insert instructions in the ROB according to the **branch prediction** for each branch.
- Indicate the prediction for each branch and whether it turns out to be correct or not.
- By the time a branch misprediction is detected in the Execute stage, **two** "bad" instructions (from the wrong path) have entered the IQ/ROB in the Issue stage.
- When you squash instructions in the wrong path, cross out (~~cross out~~) the corresponding row in the ROB table and continue filling the ROB from the correct path by re-adjusting the Tail pointer (i.e., the ROB position)
- **Rename the registers** of the instructions **of the first two iterations** as follows:
  R2 → P16 indicates new renaming that assigns architectural register R2 to the physical register P16
  R2 : P16 indicates an existing renaming that you use in a instruction --- in this example R2 has been assigned in the past to P16
  (you can keep your own register rename table so you don't lose track of the past renamings)
- The first three instructions are *partially* filled-in to get you started (you need to complete the renaming of 2 and 3)

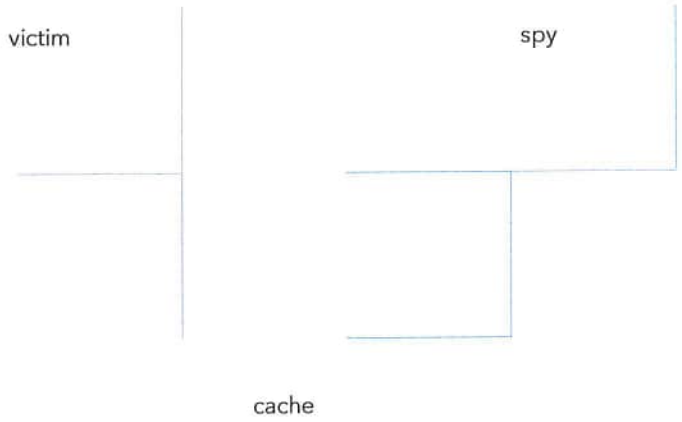
ROB position	PC	Renamed Instruction	Register Renaming	Branch prediction T / NT (correct/wrong)
1	1	`MULI P1, Pi, #32`	R5→P1 , Ri:Pi	
2	2	`ADD …`		
3	3	`BLTEQ …`		NT (correct)

18. Security, Caches, and Speculative Execution (6p)

Three parts: a (1p), b (2p), and c (3p).

A cache side-channel leaks information via changes in the cache made by a victim program, changes that can be detected by an adversary.

a. What kind of program information can leak out of a cache side-channel from a non-speculative execution of a program?
b. How can an adversary extract this information from the cache? Hint: Think what the adversary has to do to detect specific changes in the cache --- assume that the adversary accesses the cache before and after the victim, but cannot directly read the victim's own cachelines in the cache. Use the diagram to help your description.



c. A speculative side-channel attack uses a cache side-channel in an entirely new way. The following code (in simple C) is a huge security threat in a core with speculative execution (branch prediction speculation) because it can directly leak the value of the variable "secret" via the cache side-channel. How does it do that? Detail you answer explaining step-by-step what happens in the core and in the caches. (Assume that cacheline size is 64 bytes.)

```
char x;  // undefined
int array1_size = 100;
char array1[100];  // an array of 100 bytes: array1[0]...array1[99]
                   // the largest valid index for array1 is 99
char secret = 13; // a secret byte is located in memory just after the end of array1
                  // and happens to map on exactly array1[100]
...
char array2[16384]; // a probe array

main(){
...

// attack
x = 100;
flush_cache;  // clear the cache (and write back to memory its dirty contents)
              // assume that this is part of the attack
if (x < array1_size)
   y = array2[ array1[x] * 64 ];
...
}
```

19. Bonus Bonus Question (3p)

In the following figures a simple convolution graph workload is shown executed in a CPU/GPU style system and in a Spatially-mapped Dataflow system.
Name and concisely describe three advantages of the dataflow system over the CPU/GPU system. Keep the descriptions of any advantage short and to the point (a couple of sentences at most, would suffice).

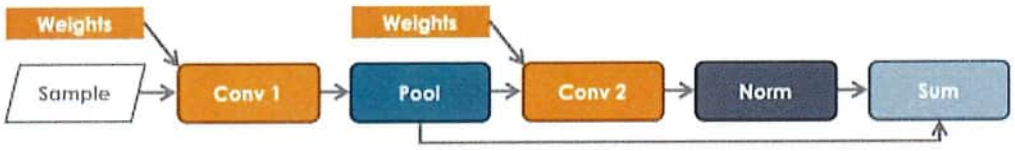


Figure 2 - Simple convolution graph

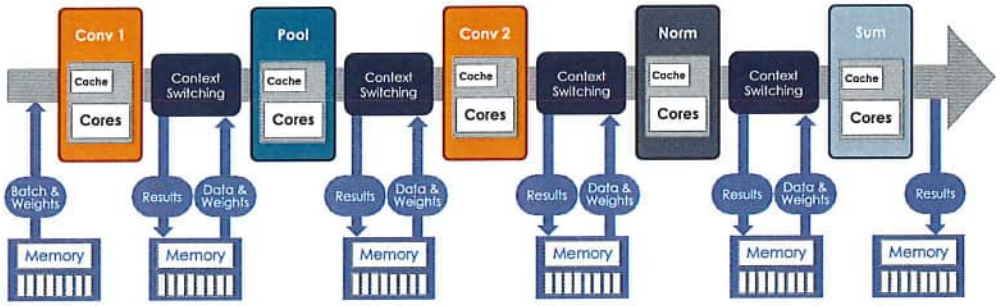


Figure 3 - Core-based kernel by kernel execution

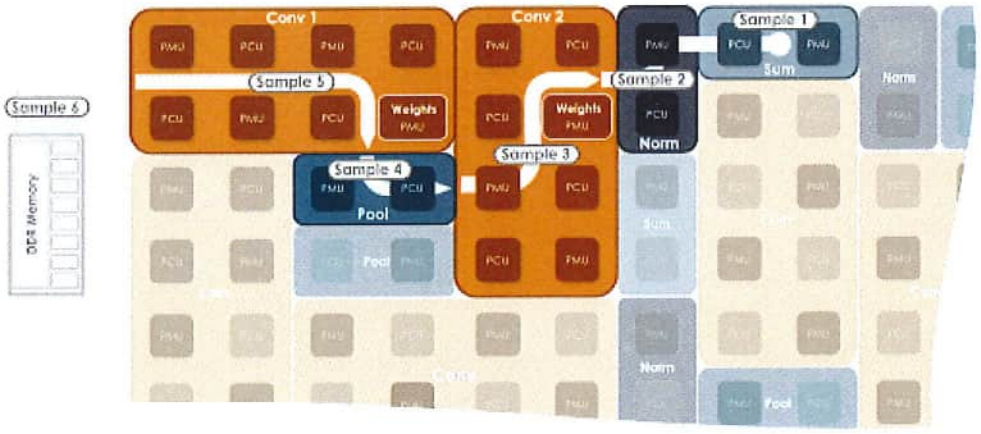


Figure 4 - RDU dataflow execution

Dataflow Advantages:

-
-
-