Exercise 1

Introduction

Consider a fabrication process with 25cm wafers (diameter) and a defect rate of 0.5 defects/cm^2.

(a) Calculate the yield of the process, if the chip area is 0.75cm^2. How many functioning chips do you receive per wafer?

- Number of dies: $Dies\ per\ wafer \approx \frac{wafer\ area}{die\ area} = (\frac{\pi}{4} \cdot d^2)/die\ area \approx 654$
- Yield: $\frac{1}{(1+Defects\ per\ area \cdot Die\ area/2)^2} \approx 0.709$
- Functioning chips: $654 \cdot 0.709 \approx 463$

(b) Calculate the yield of the process, if the chip area is 1.5cm^2. How many functioning chips do you receive per wafer?

- Number of dies: 327
- Yield: 0.529
- Functioning chips: 172

(c) Assume that the costs per wafer are 3000$: What is the minimum sale price of the chips of (a) and (b) to prevent financial losses?

- (a) $3000\$/463 \approx 6.48\$$
- (b) $3000\$/172 \approx 17.45\$$

(d) Assume that the wafer diameter can be increased by 10cm while having the same defect rate: Recalculate the yield and the number of functioning chips for the chip area given in (a).

- Number of dies: 1282
- Yield: 0.709
- Functioning chips: 908

(e) Calculate the maximum acceptable cost per wafer, given that the sale price for the chips should not be increased when using the bigger wafers.

- $6.48\$ \cdot 908 \approx 5884\$$

(f) Assume that for the parameters of (a), the fabrication process can be improved such that the resulting yield is 0.85. Calculate the defect rate.

- Defect rate ≈ 0.2257

Note that the formulas presented in the lecture are approximated. In case you are interested in seeing more precise results and a graphical representation, you can check online calculators, e.g., https://caly-technologies.com/die-yield-calculator/

Exercise 2

Performance

A program uses 5% of its operations for floating point multiplications, 15% for floating point divisions and 30% for floating point additions. To speed up the execution of the program, the following suggestions are made:

(a) Speed up the addition by factor 4.

(b) Speed up the multiplication by factor 8.

(c) Speed up the addition and division by factor 1.5, respectively.

(d) Speed up the multiplication and division by factor 2, respectively.

Show the results for all options above.

- (a) $T_i = 0.3 \cdot T \cdot 1/4 + 0.7 \cdot T = 0.775 \cdot T$
- (b) $T_i = 0.05 \cdot T \cdot 1/8 + 0.95 \cdot T = 0.95625 \cdot T$
- (c) $T_i = 0.3 \cdot T \cdot 1/1.5 + 0.15 \cdot T \cdot 1/1.5 + 0.55 \cdot T = 0.85 \cdot T$
- (d) $T_i = 0.05 \cdot T \cdot 1/2 + 0.15 \cdot T \cdot 1/2 + 0.8 \cdot T = 0.9 \cdot T$

(e) Which option is the best one?

Option a is the best one.

Exercise 3

Performance

Consider two processors P1 and P2 implementing the same instruction set. They have the following characteristics:

- P1: 2.2GHz clock rate; 1.9 average CPI
- P2: 1.3GHz clock rate; 1.1 average CPI

(a) Calculate the performance in terms of instructions per second for processors P1 and P2.

$$instr\_per\_sec(P1) = 2.20 \cdot 10^9 / 1.9 = 1.16 \cdot 10^9$$
$$instr\_per\_sec(P2) = 1.30 \cdot 10^9 / 1.1 = 1.18 \cdot 10^9$$

(b) Assume that processor P1 executes a benchmark program in 30 seconds. Calculate the corresponding number of cycles and the corresponding number of instructions.

$$\#cycles(P1) = 2.20 \cdot 10^9 \cdot 30 = 6.60 \cdot 10^{10}$$
$$\#instr(P1) = 6.60 \cdot 10^{10} / 1.9 = 3.47 \cdot 10^{10}$$

(c) Assume that processor P2 executes the same benchmark program and requires the same number of instructions for its execution: Calculate the execution time.

$$ET(P2) = 6.60 \cdot 10^{10} \cdot 1.1 \cdot 1/1.3GHz = 29.4s$$

Assume a third processor P3 implementing a different instruction set, which has a 3.5Ghz clock rate, an average CPI of 1.5 and executes $8 \cdot 10^{10}$ instructions for executing the benchmark program.

(d) Calculate the execution time.

$$ET(P3) = 34.3s$$

(e) Calculate the MIPS rating of P1, P2 and P3.

$$MIPS(P1) = 1160$$
$$MIPS(P2) = 1180$$
$$MIPS(P3) = 2333$$

(f) Given the results calculated so far: Show that frequency/clock rate is not a good performance metric.

$ET(P1) > ET(P2)$ (i.e., execution of the benchmark program on P1 takes longer than on P2) while $f(P2) < f(P1)$ (i.e., the clock rate of P2 is lower than the one of P1).

Exercise 4

Instructions/Processor/Pipelining

(a) Consider the following RISC-V assembly code (32-bit RISC-V version). It was written for a 5-stage RISC-V pipeline, where forwarding and handling of control hazards are implemented. Describe in one sentence as precisely as possible which functionality it implements.

```
1  <func>
2    lw      a5,4(a0)
3    lw      a4,8(a0)
4    nop
5    add     a5,a5,a4
6    lw      a4,0(a0)
7    nop
8    add     a5,a5,a4
9    lw      a0,12(a0)
10   nop
11   add     a0,a5,a0
12   srli    a0,a0,0x2
13   jalr    zero,0(ra)
```

The code loads four memory/array entries, calculates their sum and divides the sum by 4 → integer average of the entries of an array of size 4.

(b) Explain why the "nop" instructions at (4), (7) and (10) are required.

The nop instructions are required because the add instructions can't follow the load instructions directly as otherwise the old values would be used for processing → Load-use Data Hazard.

(c) Rewrite the code of (a) for a 5-stage RISC-V pipeline that neither supports forwarding nor hazard detection. Try to keep the performance as high as possible. Explain your changes.

Only rearranging

```
1  lw   a5,4(a0)
2  lw   a4,8(a0)
3  nop
4  nop
5  add  a5,a5,a4
6  lw   a4,0(a0)
7  nop
8  nop
9  add  a5,a5,a4
10 lw   a0,12(a0)
11 nop
12 nop
13 add  a0,a5,a0
14 nop
15 nop
16 srli a0,a0,0x2
17 jalr zero,0(ra)
18 nop
19 nop
20 nop
```

Using more registers (better solution)

```
1  lw   a5,4(a0)
2  lw   a2,8(a0)
```

Exercise 4 continued on next page...

```
3  lw   a3,0(a0)
4  lw   a4,12(a0)
5  add  a0,a5,a2
6  nop
7  add  a5,a3,a4
8  nop
9  nop
10 add  a0,a0,a5
11 nop
12 nop
13 srli a0,a0,0x2
14 jalr zero,0(ra)
15 nop
16 nop
17 nop
```

(d) Optimize the code of (a) with respect to the code size. Consider a 5-stage RISC-V pipeline, where forwarding and handling of control hazards are implemented.

Using more registers and rearranging code

```
1 lw   a5,4(a0)
2 lw   a2,8(a0)
3 lw   a3,0(a0)
4 lw   a4,12(a0)
5 add  a0,a5,a2
6 add  a0,a0,a3
7 add  a0,a0,a4
8 srli a0,a0,0x2
9 jalr zero,0(ra)
```

"Undoing" loop unrolling (better solution)

```
1      addi a4,a0,16
2      addi a5,zero,0
3  L1: lw   a3,0(a0)
4      addi a0,a0,4
5      add  a5,a5,a3
6      bne  a0,a4,L1
7      srli a0,a5,0x3
8      jalr zero,0(ra)
```

Exercise 5

Instructions/Processor/Pipelining

Consider the following RISC-V assembly code (32-bit RISC-V version).

```
<func>:
        addi a5,zero,0
        nop
        nop
        nop
label1: bne  a5,a2,label2
        nop
        nop
        nop
        jalr zero,0(ra)
        nop
        nop
        nop
label2: addi a3,zero,0
        nop
        nop
        nop
        addi a4,zero,0
        nop
        nop
        nop
label3: slli a6,a4,0x2
        nop
        nop
        nop
        add  a6,a0,a6
        nop
        nop
        nop
        lw   a6,0(a6)
        nop
        nop
        nop
        add  a3,a3,a6
        nop
        nop
        nop
        addi a4,a4,1
        nop
        nop
        nop
        bge  a5,a4,label3
        nop
        nop
        nop
        slli a4,a5,0x2
        nop
        nop
        nop
        add  a4,a1,a4
        nop
        nop
        nop
        sw   a3,0(a4)
        nop
        nop
        nop
        addi a5,a5,1
        nop
        nop
        nop
        jal  zero,label1
        nop
        nop
        nop
```

(a) Describe in one sentence as precisely as possible which functionality it implements. Hint: ***a0*** *holds the base address of an array containing the input,* ***a1*** *holds the base address of an array for the output,* ***a2*** *contains the number of array entries.*

> Takes an array of size reg(a2) as an input and calculates for each index i the sum of the entry itself and all its "predecessors" (i.e., entries with indices 0..i-1) in the array and stores it at index i in the output array (running sum).

(b) The code above was written for a basic 5-stage RISC-V pipeline without forwarding and hazard detection. Optimize the code for an enhanced version of the pipeline, where forwarding and handling of control hazards are implemented. You may re-arrange and remove instructions, but you are not allowed to add or modify instructions. Explain your optimizations and justify potentially remaining "nop" instructions.

Optimized

```
        addi a5,zero,0
label1: bne  a5,a2,label2
        jalr zero,0(ra)
label2: addi a3,zero,0
        addi a4,zero,0
label3: slli a6,a4,0x2
        add  a6,a0,a6
        lw   a6,0(a6)
        addi a4,a4,1
```

Exercise 5 continued on next page...

```
10          add  a3,a3,a6
11          bge  a5,a4,label3
12          slli a4,a5,0x2
13          add  a4,a1,a4
14          sw   a3,0(a4)
15          addi a5,a5,1
16          jal  zero,label1
```

There are two aspects to take care of: (1) Realizing that all nops except for (30) can be removed directly (load-use data hazard, which is not solved by forwarding). (2) Then, this nop can be removed by swapping the add instructions directly following the lw.

(c) Describe which further improvements can be made by rewriting the code above completely. Write the corresponding assembly code.

Rewritten

```
1  10218: 00261613 slli a2,a2,0x2
2  1021c: 00000793 addi a5,zero,0
3  10220: 00000713 addi a4,zero,0
4  10224: 00c79463 bne  a5,a2,1022c
5  10228: 00008067 jalr zero,0(ra)
6  1022c: 00f506b3 add  a3,a0,a5
7  10230: 0006a683 lw   a3,0(a3)
8  10234: 00d70733 add  a4,a4,a3
9  10238: 00f586b3 add  a3,a1,a5
10 1023c: 00e6a023 sw   a4,0(a3)
11 10240: 00478793 addi a5,a5,4
12 10244: fe1ff06f jal  zero,10224
```

Instead of calculating the individual sums from scratch for every entry in the output array, the already existing result from the output array can just be used. Then, only the sum of the entry with index i from the input array and entry with index i-1 from the output array has to be calculated, forming the result for index i in the output array.

Exercise 6

Instructions/Processor/Pipelining

(a) Consider the following RISC-V assembly code (32-bit RISC-V version). Complete the missing machine code.

```
<func>:
  101f4:  00052503  lw   a0,0(a0)
  101f8:  ________  lw   a5,0(a1)
  101fc:  ________  bltu a0,a5,10208
  10200:  ________  sub  a0,a0,a5
  10204:  ________  bgeu a0,a5,10200
  10208:  ________  jalr zero,0(ra)
```

```
<func>:
  101f4:  00052503  lw   a0,0(a0)
  101f8:  0005a783  lw   a5,0(a1)
  101fc:  00f56663  bltu a0,a5,10208
  10200:  40f50533  sub  a0,a0,a5
  10204:  fef57ee3  bgeu a0,a5,10200
  10208:  00008067  jalr zero,0(ra)
```

(b) Describe in one sentence as precisely as possible which functionality the code shown in (a) implements.

The code returns the remainder of the division of two values a and b by using repeated subtractions, i.e., it calculates a%b (where a is loaded by the first 'lw' and b is loaded by the second 'lw').

(c) Consider the following RISC-V machine code (32-bit RISC-V version). Complete the missing assembly instructions.

```
<func>:
   101f4:  00052503  lw   a0,0(a0)
   101f8:  0005a783  ____________
   101fc:  00f57463  ____________
   10200:  00008067  ____________
   10204:  40f50533  ____________
   10208:  ff5ff06f  ____________
```

```
<func>:
   101f4:  00052503  lw   a0,0(a0)
   101f8:  0005a783  lw   a5,0(a1)
   101fc:  00f57463  bgeu a0,a5,10204
   10200:  00008067  jalr zero,0(ra)
   10204:  40f50533  sub  a0,a0,a5
   10208:  ff5ff06f  jal  zero,101fc
```

(d) Describe in one sentence as precisely as possible which functionality the code shown in (c) implements.

As in (a), the code also returns a%b, but with a sligthly different set of instructions.

Exercise 7

Memory

You are given a processor system with a cache having the following design parameters:

- four-way set-associative
- LRU
- block size: 1 word
- number of sets: 4

(a) For the following byte addresses, write down (a) the block address, (b) the set, (c) the tag, (d) if the access results in a hit or a miss and (e) the tag of the evicted entry.

Byte address	Block address	Set	Tag	Hit/Miss	Evicted
192_{10}					
0_{10}					
480_{10}					
448_{10}					
192_{10}					
80_{10}					
488_{10}					
480_{10}					
256_{10}					
60_{10}					
8_{10}					
448_{10}					
8_{10}					
72_{10}					
344_{10}					
168_{10}					

Exercise 7 continued on next page...

Byte address	Block address	Set	Tag	Hit/Miss	Evicted
192_{10}	48	0	12	M	
0_{10}	0	0	0	M	
480_{10}	120	0	30	M	
448_{10}	112	0	28	M	
192_{10}	48	0	12	HIT	
80_{10}	20	0	5	M	0
488_{10}	122	2	30	M	
480_{10}	120	0	30	HIT	
256_{10}	64	0	16	M	28
60_{10}	15	3	3	M	
8_{10}	2	2	0	M	
448_{10}	112	0	28	M	12
8_{10}	2	2	0	HIT	
72_{10}	18	2	4	M	
344_{10}	86	2	21	M	
168_{10}	42	2	10	M	30

(b) Show the cache state after the last access.

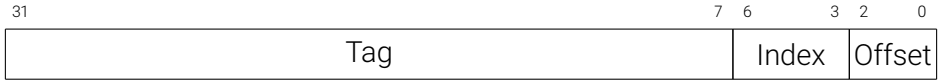
Set	Valid	Tag	Valid	Tag	Valid	Tag	Valid	Tag
0								
1								
2								
3								

Set	Valid	Tag	Valid	Tag	Valid	Tag	Valid	Tag
0	V	28	V	5	V	30	V	16
1								
2	V	10	V	0	V	4	V	21
3	V	3						

Exercise 8

Memory

(a) Assume a system with a 2-way set associative cache using byte addressing. The partitioning of the main memory address looks as follows:

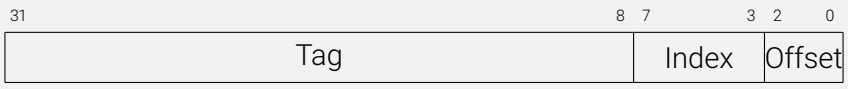


Calculate the block size, the number of blocks, the number of sets, the capacity (only data) and the overall size of the cache.

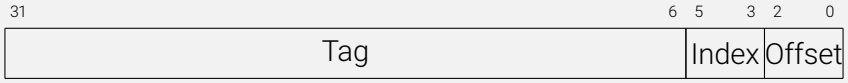
- Block size: $2^3 Byte = 8Byte$
- Number of blocks: $2 \cdot 2^4 = 32$
- Number of sets: $2^4 = 16$
- Capacity: $\#blocks \cdot block\ size = 32 \cdot 8Byte = 256Byte$
- Overall size: $\#blocks \cdot (block\ size + tag + valid\ bit) = 32 \cdot (64bit + 25bit + 1bit) = 360Byte$

(b) Given the system shown in (a): Give five alternative cache designs with the same capacity (only data) while keeping the same block size. Additionally, show the partitioning of the main memory address, respectively.

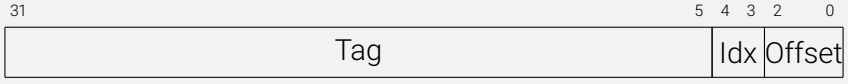
- Direct mapped/1-way set associative (32 sets)



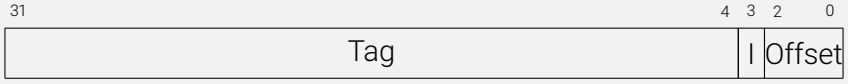
- 4-way set associative (8 sets)



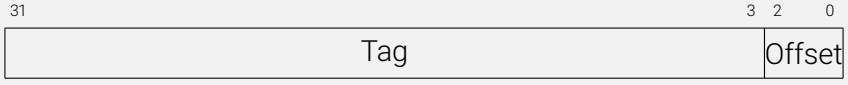
- 8-way set associative (4 sets)



- 16-way set associative (2 sets)



- Fully associative/32-way set associative (1 set)



Exercise 9

Memory

(a) Assume that the accesses to memory addresses shown in the tables below are given. For those accesses compare different cache designs (by filling the following tables). The cache is initially empty, byte addressing is used and the replacement strategy is LRU.

(I) A direct-mapped cache with a capacity (data) of 512 bytes and a block size of 8 bytes.

Byte address	Block address	Set	Tag	Hit/Miss	Evicted
11332_{10}					
11344_{10}					
10818_{10}					
11840_{10}					
11328_{10}					
11856_{10}					

(II) A 2-way set-associative cache with a capacity (data) of 512 bytes and a block size of 8 bytes.

Byte address	Block address	Set	Tag	Hit/Miss	Evicted
11332_{10}					
11344_{10}					
10818_{10}					
11840_{10}					
11328_{10}					
11856_{10}					

(I) BLK_SIZE: 8; CAPACITY: 512Byte; ASSOC: 1
TAG: 23bit | INDEX: 6bit | OFFSET: 3bit
#BLOCKS: 64; #SETS: 64; OVERALL SIZE: 704Byte

Byte address	Block address	Set	Tag	Hit/Miss	Evicted
11332_{10}	1416	8	22	M	-
11344_{10}	1418	10	22	M	-
10818_{10}	1352	8	21	M	22
11840_{10}	1480	8	23	M	21
11328_{10}	1416	8	22	M	23
11856_{10}	1482	10	23	M	22

Exercise 9 continued on next page...

(II): BLK_SIZE: 8Byte; CAPACITY: 512Byte; ASSOC: 2;
TAG: 24bit | INDEX: 5bit | OFFSET: 3bit
#BLOCKS: 64; #SETS: 32; OVERALL SIZE: 712Byte

Byte address	Block address	Set	Tag	Hit/Miss	Evicted
11332_{10}	1416	8	44	M	-
11344_{10}	1418	10	44	M	-
10818_{10}	1352	8	42	M	-
11840_{10}	1480	8	46	M	44
11328_{10}	1416	8	44	M	42
11856_{10}	1482	10	46	M	-

(b) Find two different better alternative cache designs instead of the ones presented in part (a), which achieve a better hit rate for the given accesses. In general, valid solutions have to vary different cache design parameters, respectively, and can only change one design parameter at a time compared to the configuration in (a).I or (a).II. Explain why (or show that) your solution achieves a better hit rate.

Option 1: Increase the associativity
BLK_SIZE: 8Byte; CAPACITY: 512Byte; ASSOC: 4
TAG: 25bit | INDEX: 4bit | OFFSET: 3bit
#BLOCKS: 64; #SETS: 16; OVERALL SIZE: 720Byte
Note that higher associativities work as well, but giving e.g., a 4-way and an 8-way set associative cache as an answer does not fulfill the requirements of the question as the same design parameter (i.e., associativity) is varied in both cases.
Explanation: Hit for 11328_{10} as no eviction happens by accessing 10818_{10} or 11840_{10} due to the higher associativity.

Byte address	Block address	Set	Tag	Hit/Miss	Evicted
11332_{10}	1416	8	88	M	-
11344_{10}	1418	10	88	M	-
10818_{10}	1352	8	84	M	-
11840_{10}	1480	8	92	M	-
11328_{10}	1416	8	88	HIT	-
11856_{10}	1482	10	92	M	-

Option 2: Increase the capacity
BLK_SIZE: 8Byte; CAPACITY: 1024Byte; ASSOC: 1
TAG: 22bit | INDEX: 7bit | OFFSET: 3bit
#BLOCKS: 128; #SETS: 128; OVERALL SIZE: 1392Byte
Explanation: Hit for 11328_{10} as with the higher capacity, the accesses to 10818_{10} and 11840_{10} (causing evictions earlier) map to a different set.

Exercise 9 continued on next page...

Byte address	Block address	Set	Tag	Hit/Miss	Evicted
11332_{10}	1416	8	11	M	-
11344_{10}	1418	10	11	M	-
10818_{10}	1352	72	10	M	-
11840_{10}	1480	72	11	M	10
11328_{10}	1416	8	11	HIT	-
11856_{10}	1482	74	11	M	-

Exercise 10

Memory

(a) In the lecture it was shown that oftentimes a better hit rate can be achieved when increasing cache associativity. Is this always the case? If yes, explain why. If no, show a counterexample.

One counterexample can be shown comparing a direct-mapped with a 2-way set associative cache for the following access pattern:

Design 1: Associativity: 1 (i.e., direct-mapped); Capacity: 16Byte; Block size: 4Byte

Byte address	Block address	Set	Tag	Hit/Miss	Evicted
352_{10}	88	0	22	M	-
360_{10}	90	2	22	M	-
280_{10}	70	2	17	M	22
354_{10}	88	0	22	HIT	-

Design 2: Associativity: 2 (i.e., 2-way set-associative); Capacity: 16Byte; Block size: 4Byte

Byte address	Block address	Set	Tag	Hit/Miss	Evicted
352_{10}	88	0	44	M	-
360_{10}	90	0	45	M	-
280_{10}	70	0	35	M	44
354_{10}	88	0	44	M	45

(b) Assume that you are given a system and you are asked to find out the cache design parameters. The only information given is the following:

- Byte addressing is used
- Block size: 4, 8, 16, 32 or 64Byte
- Associativity: 1-, 2-, 4-, or 8-way set-associative
- Capacity (data): 2048Byte or 4096Byte
- Replacement: LRU

The only way to find out the actual design parameters is to give the system a sequence of accesses (with an initially empty cache) and observing the hit rate of the cache after finishing the complete sequence. Explain your solution.
(I) Propose a sequence of accesses for finding out the block size of the cache.

The idea is to start with address 0 and test for accesses potentially going to the same block, based on the block sizes, which are known to be an option, i.e.,

Access #	1	2	3	4	5	6	7	8
Address	0	2	4	8	16	32	64	128

Exercise 10 continued on next page...

Assume that a hit rate of 2/8 is observed: This means that the accesses to 2 and 4 result in a hit, respectively, as the block is brought into the cache by the access to address 0. The block size can't be smaller than 8, as otherwise a lower hit rate would be observed. The block size also can't be larger than 8, as otherwise the access to address 8 would result in a hit as well and therefore a larger hit rate would be observed. Consequently, the block size must be 8Byte.

(II) Propose a sequence of accesses for finding out the associativity of the cache.

The idea is to design the accesses in a way, that all accesses map to the same set and access different blocks, independent of the block size and the size of the cache (when staying within the boundaries stated above). This can be achieved in the following way:

Access #	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
Address	0	4096	0	8192	12288	0	16384	20480	24576	28672	0

The following cases can occur:

- Hit rate 0/11: Direct-mapped (all accesses map to the same set, therefore each access (except for the first one) leads to an eviction)
- Hit rate 1/11: 2-way set-associative (the second access to 0 results in a hit, as there is a free entry left in the set when accessing 4096, where the corresponding block can be stored; the following accesses to 0 result in a miss due to the following evictions)
- Hit rate 2/11: 4-way set-associative (the second and the third access to 0 result in a hit, when accessing 0 for the fourth time, the corresponding block is evicted as four other blocks were accessed in between)
- Hit rate 3/11: 8-way set-associative (the second, the third and the fourth access to 0 result in a hit)

(III) Propose a sequence of accesses for finding out the capacity of the cache.

The idea is to test for evictions that would happen in the cache of the smaller size, but not in the one with the larger size. Therefore, based on the associativity found in the previous analysis, one of the following cases can be considered to find the cache size:

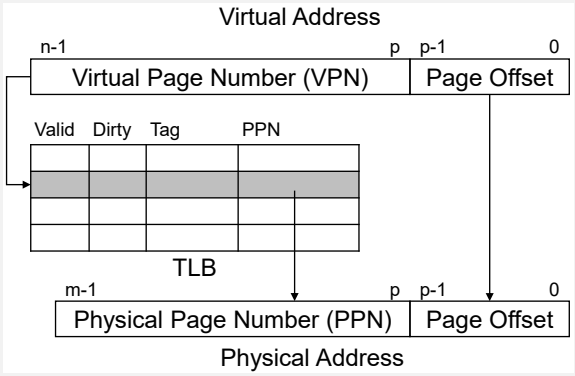
- DM: 0, 2048, 0 $\rightarrow$ Addresses 0 and 2048 map to the same set for the smaller cache size (no hit for last access; hit rate 0/3) and to different sets for the larger cache size (hit for last access; hit rate 1/3)
- 2W: 0, 4096, 1024, 0 $\rightarrow$ Addresses 0 and 1024 map to the same set for the smaller cache size (no hit for last access; hit rate 0/4) and to different sets for the larger cache size (hit for last access; hit rate 1/4). The access to address 4096 maps to the same set as address 0 in both cases, so that for the following access to 1024 either an eviction occurs in set 0 (smaller cache) or not (larger cache).
- 4W: 0, 4096, 8192, 12288, 512, 0 $\rightarrow$ Same idea as above, but more blocks have to be brought to set 0, so that the access to address 512 either leads to an eviction or not.
- 8W: 0, 4096, 8192, 12288, 16384, 20480, 24576, 28672, 256, 0 $\rightarrow$ Same idea as above, but more blocks have to be brought to set 0, so that the access to address 256 either leads to an eviction or not.

Exercise 11

Memory

Assume a system with Virtual Memory (VM). The Virtual Address width is 14 bit and the page size is 256Byte. The Physical Address (PA) width is 12bit. A 2-way set-associative TLB with overall 4 blocks/entries and a block size of 1 page table entry is implemented, which uses LRU replacement.

(a) Illustrate the detailed subdivision of the Virtual Address (also considering the TLB) and show the translation to (and the subdivision of) the Physical Address.



Virtual Address: | VPN: 6bit [13..8] | page offset: 8bit [7..0] |
→ TLB access: VPN: | Tag [13..9] | Index [8] |
Physical Address: | PPN: 4bit [11..8] | page offset: 8bit [7..0] |

(b) The following virtual (byte) addresses are accessed: 0x628, 0x308, 0x9FC, 0x1A0
Given (A) the page table and (B) the TLB below: Fill the corresponding tables based on the information given.

*For the **Page Table**:*

- *Note down the final state of the page table after all accesses are completed.*
- *If a page must be brought from disk, assume it is brought to the next highest page number.*

Page Table (V: Valid; PP#: Physical page number)

	V	PP#/Disk
0	1	10
1	1	8
2	0	Disk
3	0	Disk
4	1	5
5	1	9
6	1	7
7	0	Disk
8	0	Disk
9	1	6
10	0	Disk

V	PP#/Disk

Exercise 11 continued on next page...

*For the **TLB**:*

- *Note down if the access results in a TLB hit (Yes/No) and if it causes a page fault (Yes/No).*
- *Show the state of the TLB after each access. In case of multiple invalid entries in a set, evict the leftmost entry.*

TLB (V: Valid; PP#: Physical page number; LAT: Last Access Time (higher number means more recent access))

Initial State

Set	V	Tag	PP#	LAT	V	Tag	PP#	LAT
0	1	2	5	1	1	0	10	0
1	1	4	6	0	0			

Accessed VM address: 0x628 → TLB Hit? | Page Fault?

Set	V	Tag	PP#	LAT	V	Tag	PP#	LAT
0								
1								

Accessed VM address: 0x308 → TLB Hit? | Page Fault?

Set	V	Tag	PP#	LAT	V	Tag	PP#	LAT
0								
1								

Accessed VM address: 0x9FC → TLB Hit? | Page Fault?

Set	V	Tag	PP#	LAT	V	Tag	PP#	LAT
0								
1								

Accessed VM address: 0x1A0 → TLB Hit? | Page Fault?

Set	V	Tag	PP#	LAT	V	Tag	PP#	LAT
0								
1								

Page Table (V: Valid; PP#: Physical page number)

	V	PP#/Disk	V	PP#/Disk
0	1	10	1	10
1	1	8	1	8
2	0	Disk	0	Disk
3	0	Disk	1	11
4	1	5	1	5
5	1	9	1	9
6	1	7	1	7
7	0	Disk	0	Disk
8	0	Disk	0	Disk
9	1	6	1	6
10	0	Disk	0	Disk

Exercise 11 continued on next page...

TLB (V: Valid; PP#: Physical page number; LAT: Last Access Time (higher number means more recent access))

Initial State

Set	V	Tag	PP#	LAT	V	Tag	PP#	LAT
0	1	2	5	1	1	0	10	0
1	1	4	6	0	0			

Accessed VM address: 0x628 → TLB Hit? N | Page Fault? N

Set	V	Tag	PP#	LAT	V	Tag	PP#	LAT
0	1	2	5	0	1	3	7	1
1	1	4	6	0	0			

Accessed VM address: 0x308 → TLB Hit? N | Page Fault? Y

Set	V	Tag	PP#	LAT	V	Tag	PP#	LAT
0	1	2	5	0	1	3	7	1
1	1	4	6	0	1	1	11	1

Accessed VM address: 0x9FC → TLB Hit? Y | Page Fault? N

Set	V	Tag	PP#	LAT	V	Tag	PP#	LAT
0	1	2	5	0	1	3	7	1
1	1	4	6	1	1	1	11	0

Accessed VM address: 0x1A0 → TLB Hit? N | Page Fault? N

Set	V	Tag	PP#	LAT	V	Tag	PP#	LAT
0	1	2	5	0	1	3	7	1
1	1	4	6	0	1	0	8	1