1 Handwritten (68%)

Each 2%.

4.31 In this exercise we compare the performance of 1-issue and 2-issue processors, taking into account program transformations that can be made to optimize for 2-issue execution. Problems in this exercise refer to the following loop (written in C):

```
for(i=0;i!=j;i+=2)
   b[i]=a[i]-a[i+1];
```

A compiler doing little or no optimization might produce the following RISC-V assembly code:

```
      li    x12, 0
      jal   ENT
TOP:  slli  x5, x12, 3
      add   x6, x10, x5
      ld    x7, 0(x6)
      ld    x29, 8(x6)
      sub   x30, x7, x29
      add   x31, x11, x5
      sd    x30, 0(x31)
      addi  x12, x12, 2
ENT:  bne   x12, x13, TOP
```

The code above uses the following registers:

i	j	a	b	Temporary values
x12	x13	x10	x11	x5–x7, x29–x31

Assume the two-issue, statically scheduled processor for this exercise has the following properties:

1. One instruction must be a memory operation; the other must be an arithmetic/logic instruction or a branch.
2. The processor has all possible forwarding paths between stages (including paths to the ID stage for branch resolution).
3. The processor has perfect branch prediction.
4. Two instruction may not issue together in a packet if one depends on the other. (See page 324.)
5. If a stall is necessary, both instructions in the issue packet must stall. (See page 324.)

As you complete these exercises, notice how much effort goes into generating code that will produce a near-optimal speedup.

4.31.1 [30] <§4.10> Draw a pipeline diagram showing how RISC-V code given above executes on the two-issue processor. Assume that the loop exits after two iterations.

4.31.2 [10] <§4.10> What is the speedup of going from a one-issue to a two-issue processor? (Assume the loop runs thousands of iterations.)

4.31.3 [10] <§4.10> Rearrange/rewrite the RISC-V code given above to achieve better performance on the one-issue processor. Hint: Use the instruction "`beqz x13,DONE`" to skip the loop entirely if `j = 0`.

4.31.4 [20] <§4.10> Rearrange/rewrite the RISC-V code given above to achieve better performance on the two-issue processor. (Do not unroll the loop, however.)

4.31.5 [30] <§4.10> Repeat Exercise 4.31.1, but this time use your optimized code from Exercise 4.31.4.

4.31.6 [10] <§4.10> What is the speedup of going from a one-issue processor to a two-issue processor when running the optimized code from Exercises 4.31.3 and 4.31.4.

4.31.7 [10] <§4.10> Unroll the RISC-V code from Exercise 4.31.3 so that each iteration of the unrolled loop handles two iterations of the original loop. Then, rearrange/rewrite your unrolled code to achieve better performance on the one-issue processor. You may assume that `j` is a multiple of 4.

4.31.8 [20] <§4.10> Unroll the RISC-V code from Exercise 4.31.4 so that each iteration of the unrolled loop handles two iterations of the original loop. Then, rearrange/rewrite your unrolled code to achieve better performance on the two-issue processor. You may assume that `j` is a multiple of 4. (Hint: Re-organize the loop so that some calculations appear both outside the loop and at the end of the loop. You may assume that the values in temporary registers are not needed after the loop.)

4.31.9 [10] <§4.10> What is the speedup of going from a one-issue processor to a two-issue processor when running the unrolled, optimized code from Exercises 4.31.7 and 4.31.8?

4.31.10 [30] <§4.10> Repeat Exercises 4.31.8 and 4.31.9, but this time assume the two-issue processor can run two arithmetic/logic instructions together. (In other words, the first instruction in a packet can be any type of instruction, but the second must be an arithmetic or logic instruction. Two memory operations cannot be scheduled at the same time.)

Each 2%.

5.5 For a direct-mapped cache design with a 64-bit address, the following bits of the address are used to access the cache.

Tag	Index	Offset
63–10	9–5	4–0

5.5.1 [5] <§5.3> What is the cache block size (in words)?

5.5.2 [5] <§5.3> How many blocks does the cache have?

5.5.3 [5] <§5.3> What is the ratio between total bits required for such a cache implementation over the data storage bits?

Beginning from power on, the following byte-addressed cache references are recorded.

Address											
Hex	00	04	10	84	E8	A0	400	1E	8C	C1C	B4	884
Dec	0	4	16	132	232	160	1024	30	140	3100	180	2180

5.5.4 [20] <§5.3> For each reference, list (1) its tag, index, and offset, (2) whether it is a hit or a miss, and (3) which bytes were replaced (if any).

5.5.5 [5] <§5.3> What is the hit ratio?

5.5.6 [5] <§5.3> List the final state of the cache, with each valid entry represented as a record of <index, tag, data>. For example,

```
<0, 3, Mem[0xC00]-Mem[0xC1F]>
```

Each 2%.

5.10 In this exercise, we will look at the different ways capacity affects overall performance. In general, cache access time is proportional to capacity. Assume that main memory accesses take 70 ns and that 36% of all instructions access data memory. The following table shows data for L1 caches attached to each of two processors, P1 and P2.

	L1 Size	L1 Miss Rate	L1 Hit Time
P1	2 KiB	8.0%	0.66 ns
P2	4 KiB	6.0%	0.90 ns

5.10.1 [5] <§5.4> Assuming that the L1 hit time determines the cycle times for P1 and P2, what are their respective clock rates?

5.10.2 [10] <§5.4> What is the Average Memory Access Time for P1 and P2 (in cycles)?

5.10.3 [5] <§5.4> Assuming a base CPI of 1.0 without any memory stalls, what is the total CPI for P1 and P2? Which processor is faster? (When we say a "base CPI of 1.0", we mean that instructions complete in one cycle, unless either the instruction access or the data access causes a cache miss.)

For the next three problems, we will consider the addition of an L2 cache to P1 (to presumably make up for its limited L1 cache capacity). Use the L1 cache capacities and hit times from the previous table when solving these problems. The L2 miss rate indicated is its local miss rate.

L2 Size	L2 Miss Rate	L2 Hit Time
1 MiB	95%	5.62 ns

5.10.4 [10] <§5.4> What is the AMAT for P1 with the addition of an L2 cache? Is the AMAT better or worse with the L2 cache?

5.10.5 [5] <§5.4> Assuming a base CPI of 1.0 without any memory stalls, what is the total CPI for P1 with the addition of an L2 cache?

5.10.6 [10] <§5.4> What would the L2 miss rate need to be in order for P1 with an L2 cache to be faster than P1 without an L2 cache?

5.10.7 [15] <§5.4> What would the L2 miss rate need to be in order for P1 with an L2 cache to be faster than P2 without an L2 cache?

Each 2%.

5.16 As described in Section 5.7, virtual memory uses a page table to track the mapping of virtual addresses to physical addresses. This exercise shows how this

table must be updated as addresses are accessed. The following data constitute a stream of virtual byte addresses as seen on a system. Assume 4 KiB pages, a four-entry fully associative TLB, and true LRU replacement. If pages must be brought in from disk, increment the next largest page number.

Decimal	4669	2227	13916	34587	48870	12608	49225
hex	0x123d	0x08b3	0x365c	0x871b	0xbee6	0x3140	0xc049

TLB

Valid	Tag	Physical Page Number	Time Since Last Access
1	0xb	12	4
1	0x7	4	1
1	0x3	6	3
0	0x4	9	7

Page table

Index	Valid	Physical Page or in Disk
0	1	5
1	0	Disk
2	0	Disk
3	1	6
4	1	9
5	1	11
6	0	Disk
7	1	4
8	0	Disk
9	0	Disk
a	1	3
b	1	12

5.16.1 [10] <§5.7> For each access shown above, list

- whether the access is a hit or miss in the TLB,
- whether the access is a hit or miss in the page table,
- whether the access is a page fault,
- the updated state of the TLB.

5.16.2 [15] <§5.7> Repeat Exercise 5.16.1, but this time use 16 KiB pages instead of 4 KiB pages. What would be some of the advantages of having a larger page size? What are some of the disadvantages?

5.16.3 [15] <§5.7> Repeat Exercise 5.16.1, but this time use 4 KiB pages and a two-way set associative TLB.

5.16.4 [15] <§5.7> Repeat Exercise 5.16.1, but this time use 4 KiB pages and a direct mapped TLB.

5.16.5 [10] <§§5.4, 5.7> Discuss why a CPU must have a TLB for high performance. How would virtual memory accesses be handled if there were no TLB?

Each 2%.

6.7 Consider the following portions of two different programs running at the same time on four processors in a *symmetric multicore processor* (SMP). Assume that before this code is run, both x and y are 0.

Core 1: `x = 2;`

Core 2: `y = 2;`

Core 3: `w = x + y + 1;`

Core 4: `z = x + y;`

6.7.1 [10] <§6.5> What are all the possible resulting values of `w`, `x`, `y`, and `z`? For each possible outcome, explain how we might arrive at those values. You will need to examine all possible interleavings of instructions.

6.7.2 [5] <§6.5> How could you make the execution more deterministic so that only one set of values is possible?

Each 2%.

6.9 Consider the following three CPU organizations:

CPU SS: A two-core superscalar microprocessor that provides out-of-order issue capabilities on two *function units* (FUs). Only a single thread can run on each core at a time.

CPU MT: A fine-grained multithreaded processor that allows instructions from two threads to be run concurrently (i.e., there are two functional units), though only instructions from a single thread can be issued on any cycle.

CPU SMT: An SMT processor that allows instructions from two threads to be run concurrently (i.e., there are two functional units), and instructions from either or both threads can be issued to run on any cycle.

Assume we have two threads X and Y to run on these CPUs that include the following operations:

Thread X	Thread Y
A1 – takes three cycles to execute	B1 – take two cycles to execute
A2 – no dependences	B2 – conflicts for a functional unit with B1
A3 – conflicts for a functional unit with A1	B3 – depends on the result of B2
A4 – depends on the result of A3	B4 – no dependences and takes two cycles to execute

Assume all instructions take a single cycle to execute unless noted otherwise or they encounter a hazard.

6.9.1 [10] <§6.4> Assume that you have one SS CPU. How many cycles will it take to execute these two threads? How many issue slots are wasted due to hazards?

6.9.2 [10] <§6.4> Now assume you have two SS CPUs. How many cycles will it take to execute these two threads? How many issue slots are wasted due to hazards?

6.9.3 [10] <§6.4> Assume that you have one MT CPU. How many cycles will it take to execute these two threads? How many issue slots are wasted due to hazards?

6.9.4 [10] <§6.4> Assume you have one SMT CPU. How many cycles will it take to execute the two threads? How many issue slots are wasted due to hazards?

2 Programming (32%)

In this homework, we are going to examine the cache effect. The tool we'll use is rocket-chip.
You can either build rocket-chip yourself or use the image provided

```
docker pull ntuca2020/hw5 # size ~ 8.28G
docker run --name=test -it ntuca2020/hw5
cd /root
ls
```

Folder structure for this homework:

```
emulator/                          // link to rocket-chip emulator
  |-- benchmarks/                  // link to riscv-tests benchmark
  |    |-- Makefile                // complie all benchmarks
  |    |-- qsort/                  // qsort benchmark folder
  |    |-- qsort.riscv             // riscv executable
  |    |-- qsort.riscv.dump        // objdump riscv executable
  |    |-- mt-matmul/              // mt-matmul benchmark
  |    |-- mt-matmul.riscv         // riscv executable
  |    |-- mt-matmul.riscv.dump    // objdump riscv executable
  |    |-- mt-matmul_4/            // for part2
  |    |     `-- matmul.c          <-- need to be handed in
  |    |-- mt-matmul_4.riscv       // riscv executable
  |    |-- mt-matmul_4.riscv.dump  // objdump riscv executable
  |    |-- ...                     // other benchmarks
  |    `-- common
  |          |-- ...
  |          `-- crt.S             // specify number of cores available
  |-- system/                      // link to rocket-chip system
  |    |-- test.scala              // first part SoC settings
  |    |-- HW5.scala               <-- used for matrix multiplication and need to be handed in
  |    `-- *.scala                 // other default scala settings
  |-- build.sh                     // build all settings
  |-- test.sh                      // test all settings
  |-- spike_test.sh                // can test on spike first
  |-- Config1                      // Configuration1
  |-- generated-src_Config1        // Layout, RTL, mappings, dts, etc, for Config1
  |-- ...
  `-- Makefile                     // Build the configuration
```

Part 1: Observing cache behavior (17%)

Run `test.sh` and fill in cycle counts for each benchmark and each setting in the following form (2%)

	dhrystone	median	multiply	qsort	rsort	towers	vvadd
Configuration 1	(4)				(3)		(1)
Configuration 2							(1)
Configuration 3					(2),(3)		
Configuration 4					(2)		
Configuration 5							
Configuration 6	(4)						
Configuration 7	(4)						
Configuration 8							
Configuration 9							
Configuration 10							
Configuration 11							
Configuration 12	(5)						
Configuration 13	(5)						

Tabelle 1: Benchmark on different configurations

Answer the following questions (answers should be based your observation on the cache configurations and the program behavior)

- Why are (1) the same or different? (2%)
- Why are (2) the same or different? (2%)
- Why are (3) the same or different? (2%)
- Why are (4) the same or different? (2%)
- Why are (5) the same or different? (2%)
- See the `pmp.c` in `/root/emulator/benchmarks/pmp`, what does this program want to do? And how does it make it? (2%)
- Change the number of cores available in `crt.S` file (line 125) in `/root/emulator/benchmarks/common` and recompile the `mt-matmul` program (for this question, matrix size is 32x32).
  - Report the cycle count of configuration17 on 1-core, configuration19 on 2-core, and configuration20 on 4-core (1%)
  - Describe whether the cycle count decreases linearly, why or why not. (2%)

Part 2: Cache and matrix multiplication (15%)

In this part, we revisit the matrix multiplication. You are asked to implement **64x64 matrix multiplication** on **4-core, 128-B L1-D$, 128-B L1-I$** (no L2). The size of cache is fixed so that you can only change way-set setting in L1.

Change the dataset in **/root/emulator/benchmarks/mt-matmul/mt_matmul.c** to the one with 64x64 (**dataset2.h**).

The cache setting is specified in **/root/emulator/system/HW5.scala** and you can build the simulator using

```
make -j8 CONFIG=freechips.rocketchip.system.HW5Config
```

in `/root/emulator`.

The matrix multiplication program is located at **/root/emulator/benchmarks/mt-matmul/matmul.c**. Each thread will enter this function with its thread id and local storage (**128KB**) and exit once the task is finished. You may want to see the files under `mt-matmul/` and `common/`.

The distribution of the workload and the cache behavior should be considered when you implement matrix multiplication. We will score based on the cycle count coming out from your **HW5.scala** and **matmul.c**.

Grading:

- Correctness (2%)
- Based on cycle count
  - Ranking: Top 5 (8%)
  - Ranking: $6 \sim 20$ (6%)
  - Ranking: $21 \sim 40$ (4%)
  - Ranking: $41 \sim 80$ (2%)
  - Ranking: > 80 (0%)
- Report on how you make your matrix multiplication and maybe some cache miss rate statistics using spike (5%)

Bonus: Architecture and Security (5%)

Although it is important to design a high-performance architecture, it is also crucial to design a secure architecture. Read the "Spectre Attacks: Exploiting Speculative Execution" (or you may want to reference the original paper here) and answer the questions.

- How to perform "exploiting conditional branch misprediction" attack? (1%)
- How to perform "poisoning indirect branches" attack? (1%)
- How to mitigate Spectre Attacks? (at least 3 methods) (3%)

Submission

- Zip and upload your file to ceiba in the following format:

```
r09922028/              <-- zip this folder
    |-- matmul.c        // Matrix multiplication program
    |-- HW5.scala       // Cache configuration
    `-- HW5.pdf         // Including handwritten part, programming part 1 answers,
                           report on part 2 multi-core matrix-multiplication,
                           and bonus part if any
```

- Deadline: 1/11, 21:00
- **No late submission!!!**
- If there's any question, please send email to ntuca2020@gmail.com.
- TA hour for this homework: Wed 15:00 17:00 and Thur 15:00 17:00