ELEC3441: Computer Architecture
Second Semester, 2023–24

Homework 3 (r1.0)

`Due: 27 Apr, 2024, 11:59pm`

Instruction: Submit your answers electronically through Moodle.

There are 3 parts in this homework. Part A includes questions that aim to help you with understanding the lecture materials. They resemble the kind of questions you will encounter in the final exam. Part A is an **individual** portion of this homework. You should be completing and turning in *your own work*.

Part B of this homework are hands-on exercises that require you to design and evaluate processor systems using various software and hardware tools, including the RISC-V compilation tool chains. They are designed to help you understand real-world processor design and the use of various tools to help you along the way. Part B is a group work. Each group should turn in only 1 set of project file and 1 project report.

Part C is an **individual** work. It is an open-ended report that asks you to perform a simple survey.

The following summarizes the 3 parts.

Part	Type	Indv/Grp
A	Basic problem set	Individual
B	Hands-on	Group of 2 to 3
C	Open-ended Survey Report	Individual

In all cases, you are encouraged to discuss the homework problems offline or online using Piazza. However, you should not ask for or give out solution directly as that defeat the idea of having homework exercise. Giving out answers or copying answers directly will likely constitute an act of plagiarism.

Part A: Problem Set

A.1 Page Table Size

You are considering the design of the virtual memory system in your company's new 64-bit high performance embedded processor with the following features:

- 64-bit architecture
- 33-bit physical address

Your system is hardwired with 8 GiB high performance memory. Each running process has its own page table to map their 64-bit virtual addresses to the physical memory locations. There is NO swap space on board. A process is killed with an out of memory error if it requests a page of memory from the OS when there is no physical space available.

The *maximum* memory address space allocated to each process is as follows. Depending on the need of the program, the process may not use up all the spaces that are allocated. Only used pages are allocated by the OS.

Segment	**Address**
code	`0x0000_0000_1000_0000 - 0x0000_0000_101F_FFFF`
stack	`0x0000_A000_0000_0000 - 0x0000_A000_0000_FFFF`
heap	`0xD000_0000_0000_0000 - 0xD000_0000_1FFF_FFFF`

A.1.1 Based on the above information, what is the maximum amount of memory (in bytes), that a process may use?

A.1.2 Let the size of a page be 2^p. What is the maximum number of pages that a process may be allocated in each segment?

A.1.3 In this part, consider the design of a simple 1-level *linear page table* that covers the entire 64-bit address space.

Let the size (in bytes) of a page table entry (PTE) be:

$$\left\lceil \frac{\text{sizeof(PPN)} + 3}{8} \right\rceil$$

Assuming $p = 12$, what is the size of this linear page table for each process?

A.1.4 In order to reduce the total amount of storage required, you are considering to use a 2-level page table organization instead. In this scheme, the first level of page table is preallocated when a process is launched, while the second level is allocated on demand.

The first level of your page table contains 8192 entries. Given your page size is 2^p, how many PTEs are there in *each* second level page table?

A.1.5 Using your above 2-level page table, what is the maximum storage needed to implement all the necessary page tables if a process allocated all the possible pages?

A.1.6 Assume instead that a process only uses the *lower* 25 % of its address space in all 3 segments. In this case, what is the amount of memory needed to start the 2-level page table?

A.1.7 Given the above 25 % memory usage pattern, design an optimal 2-level page table system that would minimize the total size of page table. Assume $p = 12$.

A.1.8 As an OS designer, you have the choice of picking the location for the various areas. Suggest new scheme for locating the three segment, `code`, `stack` and `heap`, *together* with a design of 2-level page table system that may minimize the amount of storage needed for page table entries. If the location of the segments have no effect, please also explain why.

A.2 TLB Trace

In this exercise you will experiment with the interaction between the TLB and the page table in a VM system.

Assume the following system configuration:

- 16-bit virtual and 24-bit physical address
- 4 KiB page size
- 4-entry, *fully associative* TLB, true LRU.

Initially, the TLB and page table contains the following entries. An invalid entry is marked as "empty". An entry marked with disk$n$ has a page stored in hard disk, where n is the disk page number. Assume in this question that physical page number = disk page number + 0xD10.

TLB

Idx	VPN	PPN	Time
3	0x8	0x992	0028
2	0x2	0xB07	0011
1	0xA	0x289	0009
0	0xF	0x28A	0022

Page Table

VPN	PPN or Disk
0xF	0x28A
0xE	empty
0xD	empty
0xC	disk2
0xB	empty
0xA	0x289
0x9	empty
0x8	0x992
0x7	empty
0x6	empty
0x5	0x22E
0x4	empty
0x3	0xA9C
0x2	0xB07
0x1	empty
0x0	empty

A.2.1 Given the above VM setup, what is its TLB reach?

A.2.2 The following sequence of memory accesses are issued:

0x2468, 0x246C, 0x3070, 0xA500, 0x2470, 0x5CCC, 0xC000, 0x8004, 0x5CD0,

Answer the following:

1. For each memory access, is it a hit/miss in TLB, a hit/miss in the page table, a page fault?
2. What is the final state of the TLB and Page Table after the above accesses?

A.2.3 What are some of the advantages and disadvantages of using a larger page size?

A.2.4 You have the option to improve system performance by using an 8-entries direct map TLB. Assume the clock speed is the same for both cases, discuss what are the tradeoffs for using the new TLB compared to the original design. Give concrete examples on when (and if) one proposal will perform better than the other. State your assumptions.

A.3 TLB and Cache

based on a previous final exam question You are designing a paged memory system for a high-performance 32-bit processor for datacenter. The initial virtual memory subsystem design has the following characteristics:

- 32-bit virtual address
- 35-bit physical address
- 4 KiB page size
- Fully associated TLB, true LRU
- A single linear page table for each user process
- Page fault penalty: 10 000 ns
- On TLB hit, TLB access takes 0.5 ns
- Page tables are not cached
- Single level cache: *physically tagged*, direct map, 4 word line size, 1 MiB capacity,
- Cache hit time: 0.5 ns
- Cache miss penalty: 400 ns
- Clock rate: 1 GHz

A.3.1 Answer the following:

1	Number of bits needed for a virtual page number	bits
2	Number of bits needed for a physical page number	bits
3	Number of bits needed for page offset	bits

A.3.2 What is the TLB reach of the system?

A.3.3 If a TLB miss, the processor must reload the entry from the page table in memory. Given the above characteristic of this processor, how long does it take to update an entry to the TLB?

A.3.4 Assume your data cache always hit. After profiling a program B, you found that out of all the memory accesses, 3 % results in TLB **MISS**. Among the misses, 1 % results in a page fault. As a result of these address translations, what is the average memory access time (in seconds) for B?

A.3.5 You found that in general the data cache has a miss rate of 5 %. Your processor always complete address translations before accessing the data cache. That is, the system handles any TLB misses or page faults before accessing the cache. If a cache misses, the penalty is added on top of the time needed to translate an address.

Given this access mechanism, what is the average memory access time (AMAT) of the system, which includes both time needed for address translation and cache access?

A.3.6 Your project partner argues that since there are plenty of cache memory on board, it is possible to remove the TLB entirely. Instead, the processor can treat the in-memory page table just as any other data memory locations and allow them to be cached in the data cache. Your partner further argues that since most of the access to the page table entires will be a *hit* on the cache, overall average memory access time will not be affected significantly.

Explain the following:

(i) Can a paged virtual memory system function correctly without a TLB?

(ii) If it works, is the performance expected to be similar, better, worse than the original scheme with TLB?

(iii) If it does not work, give counter example on why TLB is needed for a VM system to be functionally correct.

When making your argument for this question, you can assume the hit rate of the data cache is 92 %, which includes both page table entries and any other normal data access. Also, you can assume that overall probability of a page fault is 1 %.

A.4 Dense-Sparse Vector Multiplications

(from previous final exam) When performing multiplications on vectors, in theory, one can speed up the process by skipping the costly multiplication operation if the value of an input element is zero (0). Define the sparsity ratio of a vector, r, where $0 \le r \le 1$, as the percentage of zeros in a vector, i.e., $r = 1$ means the vector has 100 % zero, and $r = 0$ means all the elements are non-zeros. Consider the following 2 programs that compute the dot product of two vectors. In program A, all the elements are computed regardless of their values. In program B, it skips the multiplication if the element from B is a zero.

Program A

```
int vecA[N];
int vecB[N];
int p = 0;
for (i = 0; i < N; i++) {
    a = vecA[i];
    b = vecB[i];
    p += a * b;
}
```

```
   1: add   x9, x0, x0   # stores result
   2: add   x8, x0, x0
loop: lw    x3, 0(x1)
   4: lw    x4, 0(x2)
   5: mult  x5, x3, x4   # long multiplication
   6: add   x9, x9, x5
   7: addi  x1, x1, 4
   8: addi  x2, x2, 4
   9: addi  x8, x8, 1
  10: bne   x8, x10, loop # x10 contains N
```

Program B

```
int vecA[N];
int vecB[N];
int p = 0;
for (i = 0; i < N; i++) {
    a = vecA[i];
    b = vecB[i];
    if (b != 0) {
        p += a * b;
    }
}
```

```
   1: add   x9, x0, x0   # stores result
   2: add   x8, x0, x0
loop: lw    x3, 0(x1)
   4: lw    x4, 0(x2)
   5: beq   x4, x0, skip
   6: mult  x5, x3, x4
   7: add   x9, x9, x5
skip: addi  x1, x1, 4
   9: addi  x2, x2, 4
  10: addi  x8, x8, 1
  11: bne   x8, x10, loop # x10 contains N
```

You are running Program A and B in a processor with the following properties:

- Your processor uses a branch prediction scheme. A correctly predicted branch takes 1 cycle. A mispredicted branch takes m cycles to complete.
- Your processor uses a naive "predict not-taken" scheme, i.e., speculate and kill.
- A multiplication instruction takes x cycles to complete.
- You can assume the last `bne` instruction in both programs (line 10 in A and line 11 in B) are always predicted correctly.
- All other instructions take 1 cycle to complete.

You may ignore cache in this question except in Part (d).

A.4.1 How many cycles does it take to complete Program A?

A.4.2 How many cycles does it take to complete Program B?

A.4.3 If $x = 50$, $m = 10$, then under what condition would Program B be faster than Program A?

A.4.4 An alternative design for sparse vector dot product, Program C, is shown below. Program C uses 2 vectors to store `vecB`: an index vector `vecB_idx` is an array that stores the index position of all non-zero values in `vecB`, while the value vector `vecB_val` stores the value of that position. For example, the following sparse vector:

`vecB`	0	0	0	0	75	0	0	23	0	0

is encoded as two arrays like below, together with a variable `vecB_nzlen` that stores the number of non-zeros in `vecB`:

`vecB_idx`	4	7
`vecB_val`	75	23
`vecB_nzlen`	2	

Program C

```
int vecA[N];
int vecB_idx[N];
int vecB_val[N];
p = 0;
for (i=0; i<vecB_nzlen; i++) {
    idx = vecB_idx[i];
    a = vecA[idx];
    b = vecB_val[i];
    p += a * b;
}
```

```
   1: add   x9, x0, x0    # stores result
   2: add   x8, x0, x0
loop: lw    x4, 0(x2)     # load vecB_val[i]
   4: lw    x12, 0(x11)   # load vecB_idx[i]
   5: slli  x12, x12, 2
   6: add   x12, x1, x12  # get &vecA[idx]
   7: lw    x3, 0(x12)    # load vecA[idx]
   8: mult  x5, x3, x4    # long multiplication
   9: add   x9, x9, x5
  10: addi  x11, x11, 4   # incr vecB_idx
  11: addi  x2, x2, 4     # incr vecB_val
  12: addi  x8, x8, 1
  13: bne   x8, x10, loop # x10 = vecB_nzlen
```

Now, compare the performance of the 3 implementations (Program A, B, and C) for different values of N and r? Depending on the actual application, N may range from 10 to 10 million, while r can range from 0 % to 99.99 %. You can assume $m = 10$ and $x = 50$.

You *do not* need to count the exact number of cycles for performance comparison in this part. Instead, focus on the relative strengths and weaknesses of the 3 implementations with rough estimation. You should formulate your arguments from the perspective of the *Iron Law.* For microarchitectural features, consider the effects of branches, having long multiplication instructions, cache behaviors, as well as memory usage in your discussion. *Note:* This is an open-ended question.

Part B: Hands-on Exercise

In this exercise, you will learn techniques to optimize an application for CPU and for GPU. The core of the matrix-matrix multiplication program is the following loop:

```
for (i = 0; i < N; i++){
    for (j = 0; j < N; j++) {
        for (k = 0; k < N; k++){
            c[i*N+j] += a[i*N+k] * b[k*N+j];
        }
    }
}
```

B.1 Optimizing Matrix-Matrix Multiplication for CPU

Obtain the homework files:

```
tux-1$ tar xzf ~elec3441/elec3441hw3.tar.gz
tux-1$ cd hw3
tux-1$ export HW3ROOT=$PWD
```

B.1.1 Loop Interchange There are no data or control dependencies between loop iterations, so it is possible to reorder operations arbitrarily. Depending on the order of the loops, one of the three elements `a[i*N+k]`, `b[k*N+j]`, `c[i*N+j]` stays constant during the whole inner loop. Compare the following three versions:

IJK

```
for (i = 0; i < N; i++){
    for (j = 0; j < N; j++) {
        x = 0;
        for (k = 0; k < N; k++){
            x += a[i*N+k] * b[k*N+j];
        }
        c[i*N+j] += x;
    }
}
```

KIJ

```
for (k = 0; k < N; k++){
    for (i = 0; i < N; i++){
        x = a[i*N+k];
        for (j = 0; j < N; j++) {
            c[i*N+j] += x * b[k*N+j];
        }
    }
}
```

JKI

```
for (j = 0; j < N; j++) {
    for (k = 0; k < N; k++){
        x = b[k*N+j];
        for (i = 0; i < N; i++){
            c[i*N+j] += a[i*N+k] * x;
        }
    }
}
```

B.1.2 Analyze the cache behavior of each version. Assume the matrices are very large and even one row will not fit in the cache. Matrices are stored in row-major order. One cache line fits four matrix elements. How many loads and stores does each iteration of the innermost loop need? How many cache misses per iteration will it produce for matrices A, B, C?

B.1.3 What will be the behavior for loop orders JIK, IKJ and KJI?

B.1.4 The file `test_mmm_inter.c` measures the time needed for the three above versions of matrix multiplication on various matrix sizes. Compile and run this code:

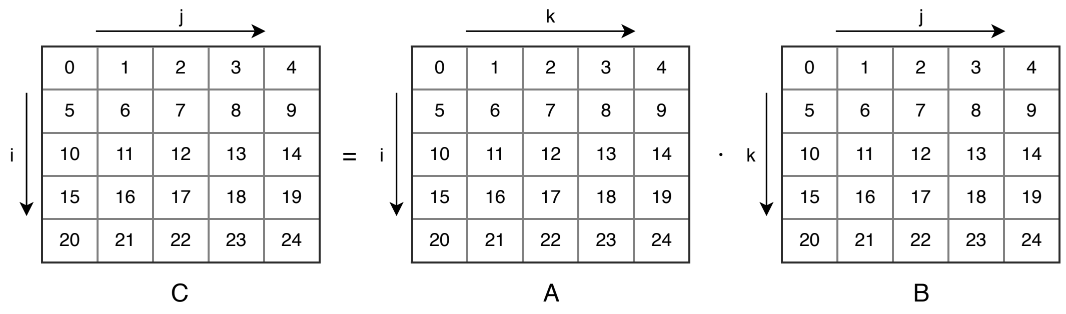
```
tux-1$ cd ${HW3ROOT}/mmm_cpu
tux-1$ gcc -O3 -o test_mmm_inter test_mmm_inter.c -lrt
tux-1$ ./test_mmm_inter
```

It will output the time taken in nanoseconds for the loop orders ijk, kij, jki for matrix sizes from 200×200 to 2000×2000. Plot the time taken per iteration (time/N^3) for each of the loop orders. Does the result match your analysis in B.1.2?

B.1.5 Can you make a guess at the L1 and L2 cache size of the machine based on this graph? You can change the values of BASE, ITER and DELTA to get a closer look for smaller matrix sizes.

B.1.6 Blocking Another way to apply the observation that the different iterations of the loop body can be executed in any order is cache blocking. After loading a small part of the matrices into the cache, to obtain the best performance as many operations using this data as possible should be executed before loading new data.

Examine the function `mmm_iijjkk_blocked` in `test_mmm_block.c`. In the following diagram, each numbered block is of size `block_size` $\times$ `block_size`. Highlight the blocks of A, B and C that will be accessed in the second row block iteration and third column block iteration, i.e. for values of `ii=1*block_size` and `jj=2*block_size`.



What inequality between the block size and the cache size needs to be fulfilled for this method to be efficient?

B.1.7 Compile and run `test_mmm_blocked.c`:

```
tux-1$ cd ${HW3ROOT}/mmm_cpu
tux-1$ gcc -O3 -o test_mmm_block test_mmm_block.c -lrt
tux-1$ ./test_mmm_block
```

Plot the results. What is the maximum efficient block size? What can you deduce about the cache size?

B.1.8 Modify the file `test_mmm_inter.c` to obtain results for the same matrix size 2048×2048. How does the performance of the blocked code compare to the different loop orders?

B.1.9 Interchange the loops in the blocked matrix multiply code. What is the best performance you can obtain by combining the two techniques?

B.1.10 Submission Submit the following:

- The modified files `test_mmm_inter.c` and `test_mmm_blocked.c`.
- The plots generated and your answers to the questions in B.1.2 to B.1.9.

Part C: Open-ended Survey Report

C.1 Survey & Report

In this *individual report*, your task is to perform a brief survey of commercial processors and use them to describe how processor architectures have changed over the span of 60 years. Specifically, select ONE typical processor from the 1960s, 1990s, and 2020s, i.e., you should pick THREE processors in total. Study and compare the processors, then report on the following:

- What processors have you picked? Be specific and make sure you report the exact model number of the processor.
- Why did you choose this processor as a typical/representative processor from the 1960s, 1990s and 2020s?
- For each processor, give a brief description of its technical design, including but not limited to the following:
  - Year of release
  - Manufacturing technology (25 µm, 4 nm, etc.)
  - Clock frequency. Is it fixed or variable?
  - Overall architecture (single processor? vector processor? multi-core? etc.)
  - Cache design (capacity, organization, split/unified instruction/data, number of level, etc.)
  - Number of operations per cycle (single scalar? superscalar?)
  - Other information
- Comparing the 3 processors, how have technologies evolved? Are they similar, or are they very different?
- Are there any features from the processors that have gone out of favor? Any features that remain?

Be creative on what processor you want to pick. You are free to pick any processor you like. Your report will be more interesting if you can select processors with some relationship among them such that you can see some trend. Your report should contain enough details and represnt your own analysis and observations. Keep it to be within 1000 words.