2024 Fall Midterm

Course: ECE454: Computer Systems Software
Examiners: Jon Eyolfson, Jianwen Zhu
Date: October 24, 2024
Duration: 1 hours 15 minutes (75 minutes)

Exam Type: A

A “closed book” examination.
No aids are permitted other than the information printed on the examination paper.

Calculator Type: 3

Non-programmable calculators from a list of approved calculators as issued by the Faculty Registrar.

Instructions:

Do not write answers on the back of pages as they will not be graded. Use the blank sheet at the end of the exam for extra space, and clearly indicate in the provided answer space if your response continues.

If a question seems unclear or ambiguous, state your assumptions and answer accordingly. In case of an error, identify it, provide a corrected version, and respond as if the question has been fixed.

Provide brief and specific answers. Clear and concise responses will receive higher marks compared to vague and wordy ones. Note that marks will be deducted for incorrect statements in your answers.

Below is a partial listing of GCC's optimize options.

-01 turns on the following optimization flags:

```
-fcompare-elim
-fcprop-registers
-fdce
-fdefer-pop
-fdelayed-branch
-fdse
-fguess-branch-probability
-fif-conversion
-fif-conversion2
-finline-functions-called-once
-fipa-modref
-fipa-reference
-fipa-reference-addressable
-fmerge-constants
-fmove-loop-invariants
-fmove-loop-stores
-ftree-bit-ccp
-ftree-ccp
-ftree-ch
-ftree-coalesce-vars
-ftree-copy-prop
-ftree-dce
-ftree-dse
-funit-at-a-time
```

-02 turns on all previous optimization flags, and additionally:

```
-fcode-hoisting
-fcse-follow-jumps  -fcse-skip-blocks
-ffinite-loops
-fgcse  -fgcse-lm
-finline-functions
-finline-small-functions
-fipa-bit-cp  -fipa-cp  -fipa-icf
-fipa-ra  -fipa-sra  -fipa-vrp
-flra-remat
-foptimize-strlen
-fpartial-inlining
-fpeephole2
-fstore-merging
-fstrict-aliasing
-fthread-jumps
-ftree-builtin-call-dce
-ftree-loop-vectorize
-ftree-pre
-ftree-slp-vectorize
-ftree-tail-merge
```

Short Answer (15 marks total)

Q1 (1 mark). **True / (False)** (Circle the correct answer)
Latency(L1 cache) < Latency(register) < Latency(Main Memory).

Q2 (1 mark). **(True) / False** (Circle the correct answer)
Latency(Main Memory) is 2 order of magnitude larger than Latency(L1 cache).

Q3 (1 mark). **True / (False)** (Circle the correct answer)
Last Level Cache ususally refers to level 2 cache in a server CPU.

Q4 (1 mark). **(True) / False** (Circle the correct answer)
If you can only speed up 50% of your code, the best overall speedup you can achieve is 2x.

Q5 (1 mark). **True / (False)** (Circle the correct answer)
A program running on a processor with a lower Cycle-Per-Instruction (CPI) always runs faster than on a processor with higher CPI.

Q6 (1 mark). **True / (False)** (Circle the correct answer)
With Out-Of-Order superscalar processors, the Instruction-Per-Cycle (IPC) of a program can always be larger than 1.

Q7 (1 mark). **True / (False)** (Circle the correct answer)
Compiler optimizations can effectively reduce compulsary cache misses.

Q8 (1 mark). **True / (False)** (Circle the correct answer)
Programmers can adjust cache associativity to reduce conflict cache misses.

Q9 (1 mark). **True / (False)** (Circle the correct answer)
A 4-way set associative cache has a 4 sets per line.

Q10 (1 mark). **True / (False)** (Circle the correct answer)
gcc compiler with -O2 option typically invokes inter-procedural link-time optimizations.

Q11 (1 mark). **(True) / False** (Circle the correct answer)
Performance profilers typically use instrumentation-based method, or sampling-based method, or both.

Q12 (1 mark). **True / (False)** (Circle the correct answer)
A program with frequent TLB misses but not page faults has a working set larger than main memory.

Q13 (1 mark). **(True) / False** (Circle the correct answer)
A physical address can correspond to multiple different virtual addresses in different processes.

Q14 (1 mark). **True / (False)** (Circle the correct answer)
Memory references are dictated by an application's algorithmic behavior and therefore a programmer can do little to enhance the application's temporal or local locality.

Q15 (1 mark). **True / (False)** (Circle the correct answer)
Consider an SRAM-based cache for a DRAM-based main memory. Neglect the possibility of other caches or levels of the memory hierarchy below main memory. If a cache is improved, increasing the typical hit rate from 98% to 99%, the typical average memory access time will be improved by 1%.

Compiler Optimization (8 marks total)

```
#define A (9)
#define B (2)

static int foo(int x, int y) {
  if (x + A > y * B) {
    return x;
  } else {
    return y;
  }
}

int main(void) {
  int x, y, z;
  x = 4;
  y = 7;
  z = foo(y, x);
  return z;
}
```

Q16 (8 marks). List 3 types of optimizations that the compiler can perform on the code shown above. Briefly describe how these optimizations would be applied. Also, show the final optimized `main` function.

1. Constant Propagation & Folding
2. Copy Propagation
3. Inlining
4. Dead Code Elimination

```
int main(void) {
  return 7;
}
```

Program Optimization (6 marks total)

Consider the following code that computes the minimum of the pairwise sum of two vectors:

```
#include <limits.h>

#define MIN(x,y)   ((x) < (y) ? (x) : (y))
#define INFINITY  INT64_MAX

typedef struct { int64_t* data; int len; } vec_t;
int vec_len( vec_t* v ) { return v->len; }
int64_t vec_elem( vec_t*v, int i ) { return v->data[i]; }
void vec_init( vec_t* v, int len ) { v->len = len; v->data = calloc( sizeof(int64_t)*len ); }
void vec_fini( vec_t* v ) { free(v->data); }

void minsum( vec_t* a, vec_t* b, int64_t* result)
{
        *result = INFINITY;
        for (int i = 0; i < vec_len(a); i++)
                *result = MIN(*result, vec_elem(a,i)+vec_elem(b,i));
}
```

Q17 (6 marks). Use the techniques learned in class to improve the implementation above. Clearly name the techniques utilized (You do not need to do loop unrolling). Show your final optimized `minsum` function.

Name 2 of:

1. Inlining
2. Loop-Invariant Code Motion
3. Local Variable (Reduction Recognition)

```
void minsum(vec_t* a, vec_t* b, int64_t* result)
{
   int64_t temp = INFINITY;
   int len = a->len;
   for (int i = 0; i < len; i++) {
       temp = MIN(temp, a->data[i], b->data[i]);
   }
   *result = temp;
}
```

Cache Hierarchy (11 marks total)

The memory hierarchy of a 32-bit machine has 4 GB byte-addressable main memory and one-level data cache with no prefetching. The data cache is two-way set-associative, has a block size of 32 bytes and can store 2KB of data. The cache uses LRU as its replacement policy. Assume that the cache lines are initially invalid.

Q18 (3 marks). Calculate the number of bits needed for the cache tag, index, and offset.

tag: 22 bits, index: 5 bits, offset: 5 bits

Consider the following code, which calls the `minsum` function developed on the previous page.

```
vec_t a, b;

int main(void) {
  int64_t result;
  vec_init(&a, 256);
  vec_init(&b, 256);
  minsum(&a,  &b, &result);
  return result;
}
```

You can assume the following:

1. `sizeof(int64_t) = 8`
2. Array begins at memory location 0x0
3. The only memory accesses are to the array entries. All other variables are stored in registers.

Q19 (4 marks). What is the cache miss rate when this code is run? Recall that miss rate is defined as #misses / #accesses.

25%

Consider the following code.

```
vec_t a;

int main(void) {
  int64_t result;
  vec_init(&a, 256);
  minsum(&a,  &a, &result);
  return result;
}
```

Q20 (2 marks). What is the cache miss rate when this code is run?

12.5%

Q21 (2 marks). What would be the cache miss rate if the code shown in **Q19** is run on a machine with a direct-mapped cache, with the same block size (32 bytes) and storage (2KB of data)?

100%

Virtual Memory (12 marks total)

Q22 (1 mark). Which of these features in a system best justify the use of at two-level page table structure, as opposed to a one level page table structure? (Circle the correct answer)
(a) Small page sizes **[Correct]**
(b) Frequent memory accesses
(c) High degree of spatial locality in programs
(d) Sparse memory usage patterns

Q23 (1 mark). Which section of an ELF file contains the compiled functions from a program?
(Circle the correct answer)
(a) .data
(b) .rodata
(c) .text **[Correct]**
(d) .bss

Q24 (10 marks). Assume a system that has:

1. A two way set associative TLB
2. A TLB with 8 total entries
3. 2^8 byte page size
4. 2^{16} bytes of virtual memory

Assume TLB has the following content (Recall that a TLB is like a cache: with index equivalent to set number, frame number equivalent to cache block content):

Index	Tag	Frame Number (PPN)	Valid
0	0010 0111 0010 1001	1100 0110 0111 0011	1 1
1	0001 0001 0000 1010	1111 1111 1110 1100	0 1
2	0010 1001 0011 1001	1100 1101 1010 1011	1 1
3	0011 0010 0010 0011	1111 1011 0100 0110	0 0

Fill in the table below. Strike out anything that you don't have enough information to fill in.

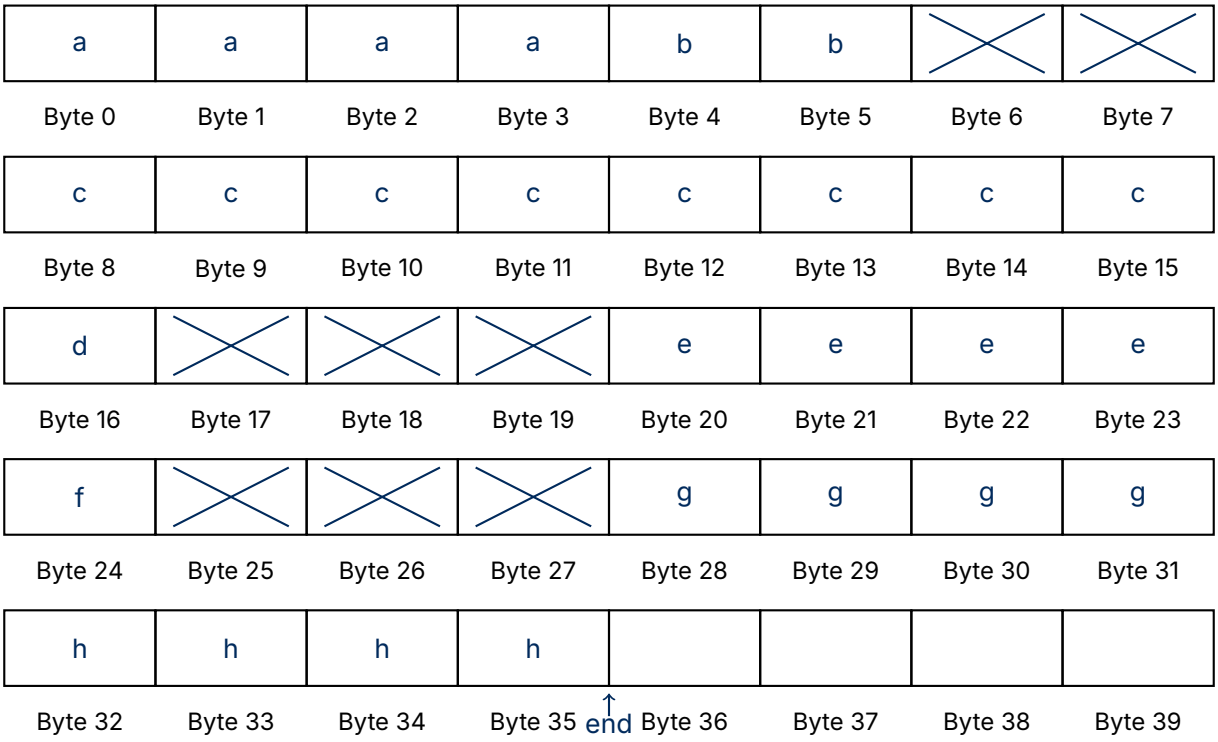
Virtual Address	Physical Address
1000 1111 0000 1111	tag not found
frame not found	0100 0110 1001 0000
1010 0100 0000 0000	0111 0011 0000 0000
0010 1001 0011 0011	1110 1100 0011 0011
0010 1000 0011 1001	tag not found

Dynamic Memory (8 marks total)

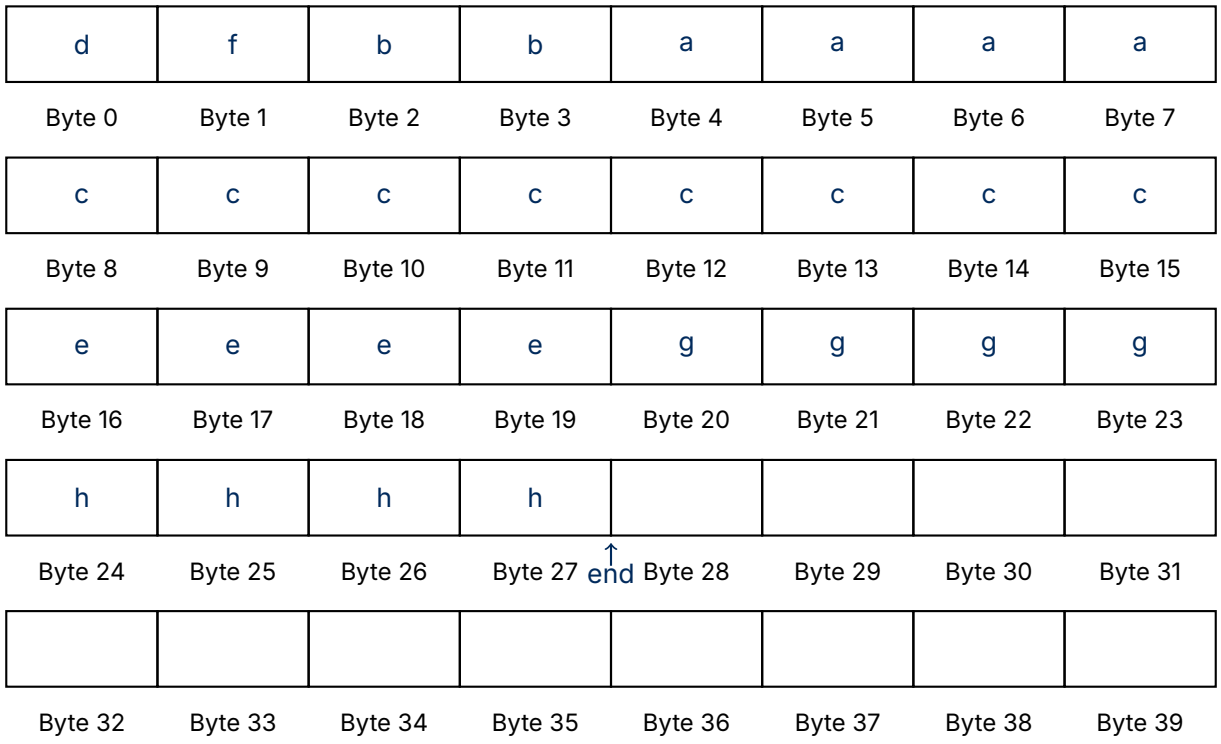
Consider the follow code fragment:

```
struct {
  char *a;
  short b;
  double c;
  char d;
  float e;
  char f;
  int g;
  void *h;
} foo;
```

Q25 (1 mark). Show how the struct above would appear on a 32-bit Linux machine (primitives of size k are k-byte aligned). Label the bytes that belong to the various fields with their names and clearly mark the end of the struct. Cross out any bytes that are allocated in the struct but are not used.



Q26 (1 mark). Rearrange the above fields in foo to conserve the most space in the memory below. Label the bytes that belong to the various fields with their names and clearly mark the end of the struct. Cross out any bytes that are allocated in the struct but are not used.



Q27 (6 marks). Compare the three classic memory management algorithms on their asymptotic complexity in the following Table, assuming the peak number of memory blocks (including allocated blocks and free blocks) is M, the peak number of free memory blocks is N. Use the Big O notation for asymptotic complexity.

Algorithm	Allocation Complexity	Free Complexity
Implicit List	O(M)	O(1)
Explicit List	O(N)	O(1)
Segregated List	O(1)	O(1)