ICS QUIZ 2

April 20, 2022

Problem 1.

Multiple choice questions. ***NOTE: There is at least one correct choice for each question.***

To receive credit, you must write your answer for each question in the following table:

1	2	3	4	5
a,d	b,c,f	a,c	a,b	c,d

6	7	8	9	10
b,c	a	a,d	e	a,b

1. Which of the following sentences about *optimization blockers* are true?
   (a) Memory aliasing describes a case that two pointers may designate the same memory location.
   (b) Function inlining reduces the overhead of function calls and the size of the generated binary object.
   (c) Compilers replace function calls with their bodies whenever encounter an *inline* function.
   (d) Higher level of optimization may make it harder to profile and debug.

2. Given the table 1, please calculate the throughput bound and latency bound on CPE for integer and floating point operations. Suppose there are two load units on the machine.

	Integer			Floating point		
Operation	Latency	Issue	Capacity	Latency	Issue	Capacity
Addition	1	1	8	3	1	2
Multiplication	3	1	2	5	5	1

Table 1: Latency, issue time, and capacity characteristics

   (a) Throughput bound on CPE for integer multiplication is 2.
   (b) Latency bound on CPE for floating point addition is 3.
   (c) Latency bound on CPE for floating point multiplication is 5.
   (d) Throughput bound on CPE for integer addition is 8.
   (e) Throughput bound on CPE for floating point multiplication is 0.2.
   (f) Latency bound on CPE for integer addition is 1.

3. We want to optimize the following code using $k \times k$ loop unrolling. The code is running on a machine whose characteristics is described in table 1. Select correct statements.

```c
// A, B, C are three double arrays
for (size_t i = 0; i < nelts; i++) {
    A[i] = B[i] + C[i];
}
```

   (a) Loop unrolling reduces the overhead of loop indexing and conditional branching and improve program performance.

   (b) The higher the k is, the higher chance of parallelism can be exploited and thus the faster the program runs.

   (c) The code will achieve the minimum CPE if $k \geq 6$.

   (d) If there are unlimited registers, loop unrolling will always improve the performance.

4. Which of the following sentences about *retirement unit* are true?

   (a) Register files are part of the retirement unit.

   (b) The instruction is decoded and placed into a first-in, first-out queue.

   (c) The instruction can be retired with any updates to the program registers being made if the operations for it have completed.

   (d) Register renaming is part of the retirement unit and control the communication of operands among the execution units.

5. Which of the following sentences about *store buffer* are true?

   (a) Store buffer is part of data cache.

   (b) Store buffer contains the addresses and data of the store operations that have been recently completed.

   (c) It is the store buffer that make the store operations can be issued every cycle.

   (d) A load operation will first check the store buffer for matching addresses.

6. The assembly code below is running on the Intel Core i7 Haswell reference machine from csapp textbook. Which statements are correct?

```
.L1:
mulsd   (%rdx),%xmm0,%xmm0
add     $0x8,%rdx
cmp     %rax,%rdx
jne     .L1
```

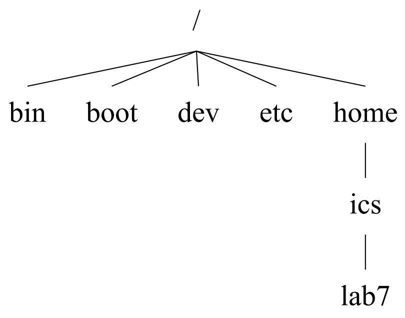
(a) There is a dependency between `add $0x8,%rdx` and `cmp %rax,%rdx`, so `cmp %rax,%rdx` will wait for the previous instruction to finish.

(b) `mulsd (%rdx),%xmm0,%xmm0` will be translated into one load operation and one multiplication operation.

(c) The updates to registers will be visible as long as the instructions are retired.

(d) The code will be executed out of order and sometimes we can observe that `add $0x8,%rdx` runs before `mulsd (%rdx),%xmm0,%xmm0`.

7. The directory hierarchy is shown below. Suppose we are now at `/home/ics` and type the command `cd ./lab7/../../../../../../home`. Which statements are correct?



(a) The *current working directory* of shell is `/home` now.

(b) The command will fail because the path `./lab/../../../../../home` does not exist.

(c) We are now at `/`, i.e. root directory.

(d) The command will fail and throw a warning that the path exceeds `/`.

8. When will short counts occur?

(a) When `read` encounters EOF(end of file).

(b) When `write` encounters error during writing to disks.

(c) When `read` from a closed socket.

(d) When `write` on a Linux pipe.

9. Consider the following piece of code. Assume the file `ics.txt` exists.

```
// content in ics.txt: hello,world
int fd1 = open(".", O_RDONLY);
int fd2 = open("ics.txt", O_RDWR | O_APPEND);
int fd3 = open("ics.txt", O_RDWR | O_TRUNC);

int nread1 = read(fd1, buf, BUF_SIZE);
int nread2 = read(fd2, buf, BUF_SIZE);
int nread3 = read(fd3, buf, BUF_SIZE);
```

Select all the correct statements.

(a) `fd1` is -1, since it is opening a directory.

(b) `nread2` is 0, because it is opened with mode `O_APPEND`.

(c) `fd1` equals to `fd2`, because they are opening the same file.

(d) `nread3` is 0, because `read(fd2, buf, BUF_SIZE)` has set the position at the end.

(e) `nread2` is 11 and the function call successfully read all the content into the buf.

10. Choose all the correct statements after running the code below. Assume `a.txt` does not exist.

```
int fd1 = dup(STDOUT_FILENO);
int fd2 = open("a.txt", O_RDWR | O_CREAT);
dup2(fd2, STDOUT_FILENO);
printf("hello,world");
```

(a) The content of `a.txt` is `hello,world`.

(b) `fd2` and `STDOUT_FILENO` points to the same file table.

(c) `fd1` and `STDOUT_FILENO` points to the same file table.

(d) The program will output `hello,world` to screen.

Problem 2.

Code Optimization. For each optimization, please briefly describe what you have optimized.

1.
```
int pred(int x) {
    if (x == 0)
        return 0;
    else
        return x - 1;
}
int func(int y) {
    return pred(y) + pred(0) + pred(y+1);
}
```

We can optimize the code using function inlining. After inline the code, we can further remove dead code. The optimized code is shown below.

```
int func(int y) {
    int tmp = 0;
    if (y != 0) tmp = y - 1;
    if (y != -1) tmp += y;
    return tmp;
}
```

2. Hint: matrix a and vector b are not overlapped. There are at least two techniques can be leveraged.

```
/* Sum rows of n X n matrix a and store in vector b. */
void sum_rows(double *a, double *b, long n) {
    long i, j;
    for (i = 0; i < n; i++) {
        b[i] = 0;
        for (j = 0; j < n; j++)
            b[i] += a[i*n + j];
    }
}
```

We do not need to consider memory aliasing since a and b are not overlapped. In each inner loop, we add results to $b[i]$, which is actually not necessary, thus we can replace it with a temporary variable. Besides, $i*n$ is repeatedly computing in the inner loop which is also unnecessary. Furthermore, we can also optimize the inner loop with loop unrolling, which will provide more opportunities for compilers exploiting instruction level parallelism. The optimized code is shown below.

```
void sum_rows(double *a, double *b, long n) {
    long i, j;
    long r;
    double data;
    for (i = 0; i < n; i++) {
        data = 0;
        r = i * n;
        for (j = 0; j < n - 1; j += 2)
            data += a[r + j] + a[r + j + 1];
        for (; j < n; j++)
            data += a[r + j];
    }
}
```

3. Hint: Consider removing duplicate calculations and figure out what the program is actually doing. You do not need to use loop unrolling.

```
/* Two stages of some calculation */
void compute(double *a, double *b, long n) {
    for (long i = 0; i < n; i++)
        for (long j = 0, j < n; j++)
            a[j*n + i] = atan2(i, j);

    for (long i = 0; i < n; i++)
        for (long j = 0, j < n; j++)
            b[i*n + j] = a[i*n + j]
                        + (i >= 1 && j >= 1)
                         ? a[(i-1)*n + (j-1)]
                         : 0;
}
```

There are many duplicate computations like $j*n, i*n$ which is unnecessary. Besides, the two separate outer loop can be merged into one outer loop to reduce loop indexing and condition checking. The optimized code is shown below.

```
void compute(double *a, double *b, long n) {
    long idx;
    for (long i = 0; i < n; i++) {
        for (long j = 0, j < n; j++) {
            idx = i*n + j;
            a[idx] = atan2(j, i);
            b[idx] = a[idx]
                        + (i >= 1 && j >= 1)
                         ? a[idx - n - 1]
                         : 0;
        }
    }
}
```

Problem 3.

CPE Estimate. Each code is running on the reference machine whose characteristics is described as table 2.

	Integer			Floating point		
Operation	Latency	Issue	Capacity	Latency	Issue	Capacity
Addition	1	1	4	3	1	1
Multiplication	3	1	2	5	1	1
Load / Store	3	1	1	3	1	1

Table 2: Latency, issue time, and capacity characteristics

1. Please draw the data flow of the following code and estimate its CPE. **NOTE: The graph should includes every register that appears in the code.** You may refer to Figure 5.19(a) in csapp textbook.

```
double sum(double *arr, long len) {
    long i;
    double res0, res1, res2;

    for (i = 0; i < len - 2; i += 3) {
        res0 = res0 + arr[i];
        res1 = res1 + arr[i + 1];
        res2 = res2 + arr[i + 2];
    }

    for (; i < len; i++) {
        res0 = res0 + arr[i];
    }

    return res0 + res1 + res2;
}

// len-2 in %rdx, i in %rax
.L2:
    addsd   (%rdi,%rax,8), %xmm1
    addsd   8(%rdi,%rax,8), %xmm2
    addsd   16(%rdi,%rax,8), %xmm0
    addq    $3, %rax
    cmpq    %rax, %rdx
    jg .L4
```

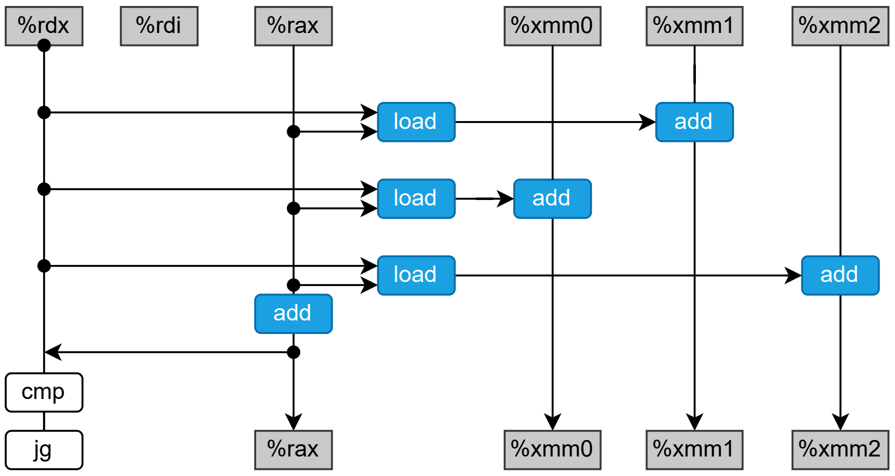


Figure 1: Throughput bound CPE

The CPE should be close to the throughput bound of floating point addition, i.e. 1. Because the code is actually a 3×3 loop unrolling and $3 \geq Capacity \times Latency = 3$.

2. Please draw the data flow of the following code and estimate its CPE. **NOTE: You do not need to draw every registers, but you have to explicitly point out the critical path and briefly explain it.**

```
typedef struct {
    double  price;
    long    cnt;
} item_t;

typedef struct {
    item_t  *item_arr;
    long    item_cnt;
} cart_t;

void do_order(cart_t *cart, double *total_price) {
    long i;
    long len = cart->item_cnt;
    item_t *data = cart->item_arr;
    item_t *end = data + len;

    while (data != end) {
        *total_price = *total_price + data->price * data->cnt;
        data++;
    }
}
// total_price in %xmm1, data in %rax, end in %rdx
.L2:
    pxor    %xmm0, %xmm0
    cvtsi2sdq   8(%rax), %xmm0
    mulsd   (%rax), %xmm0
    addsd   (%rsi), %xmm0
    movsd   %xmm0, (%rsi)
    addq    $16, %rax
    cmpq    %rdx, %rax
    jne .L2
```

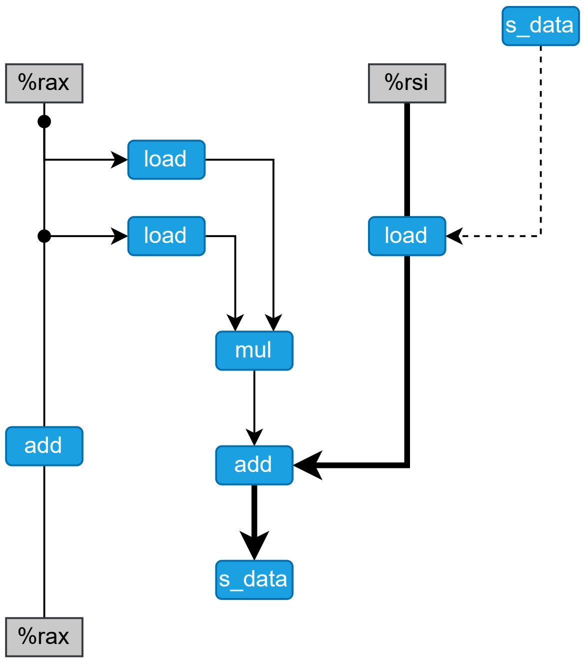


Figure 2: Critical Path

There is a dependency between each load and last store, thus the critical path is load-add-store. The estimate CPE should be the accumulated latency of load, add and store, i.e. 3 + 3 + 3 = 9.

Problem 4.

Fun with file descriptors. Read the program and answer the questions. Note that every program is independent with each other. You do not need to consider the correlations between the programs.

We have a file **ics.txt**, whose initial content contains **7** characters, **ICS-+1S**.

1. What is the output of the program?

```c
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>

#define FILE_NAME "ics.txt"
#define COUNT 7

int main() {
    char c = 'a';
    ssize_t n;
    int fd[COUNT];

    for (int i = 0; i < COUNT; i++) {
        fd[i] = open(FILE_NAME, O_RDONLY, 0);
    }

    for (int i = 0; i < COUNT; i++) {
        n= read(fd[i], &c, sizeof(c));
    }

    printf("n = %ld, c = %c\n", n ,c);

    for (int i = 0; i < COUNT; i++) {
        close(fd[i]);
    }
    return 0;
}
```

n = 1, c = I

2. What is the output of the program?

```c
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>
#include <sys/wait.h>

#define FILE_NAME "ics.txt"

int main() {
    char c = 'a';
    ssize_t n;

    int fd1 = open(FILE_NAME, O_RDONLY, 0);

    for (int i = 0; i < 4; i++) {
        read(fd1, &c, sizeof(c));
    }

    for (int i = 0; i < 2; i++) {
        pid_t pid;

        pid = fork();
        if (pid) {
            wait(NULL);
            continue;
        } else {
            read(fd1, &c, sizeof(c));
            close(fd1);
            return 0;
        }
    }

    n = read(fd1, &c, sizeof(c));
    printf("n = %ld, c = %c\n", n ,c);
    close(fd1);
    return 0;
}
```

n = 1, c = S

3. What is the output of the program?

```c
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>
#include <sys/wait.h>

#define FILE_NAME "ics.txt"

int main() {
    char c = 'a';
    ssize_t n;

    int fd1 = open(FILE_NAME, O_RDONLY, 0);

    for (int i = 0; i < 4; i++) {
        read(fd1, &c, sizeof(c));
    }

    for (int i = 0; i < 2; i++) {
        pid_t pid;

        pid = fork();
        if (pid) {
            wait(NULL);
            continue;
        } else {
            int fd2 = open(FILE_NAME, O_RDONLY, 0);
            dup2(fd1, fd2);
            read(fd1, &c, sizeof(c));
            close(fd1);
            close(fd2);
            return 0;
        }
    }

    n = read(fd1, &c, sizeof(c));
    printf("n = %ld, c = %c\n", n ,c);
    close(fd1);
    return 0;
}
```

n = 1, c = S

4. What is the output of the program?

```c
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>
#include <sys/wait.h>

#define FILE_NAME "ics.txt"

int main() {
    char c = 'a';
    ssize_t n;

    int fd1 = open(FILE_NAME, O_RDONLY, 0);

    for (int i = 0; i < 4; i++) {
        read(fd1, &c, sizeof(c));
    }

    for (int i = 0; i < 2; i++) {
        pid_t pid;

        pid = fork();
        if (pid) {
            wait(NULL);
            continue;
        } else {
            int fd2 = open(FILE_NAME, O_RDONLY, 0);
            dup2(fd2, fd1);
            read(fd1, &c, sizeof(c));
            close(fd1);
            close(fd2);
            return 0;
        }
    }

    n = read(fd1, &c, sizeof(c));
    printf("n = %ld, c = %c\n", n ,c);
    close(fd1);
    return 0;
}
```

n = 1, c = +

5. What is the output of the program? What is the final content of **ics.txt**?

```c
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>

#define FILE_NAME "ics.txt"

int main() {
    char c = 'a';
    ssize_t n;

    int fd1 = open(FILE_NAME, O_RDWR, 0);
    read(fd1, &c, sizeof(c));
    write(fd1, &c, sizeof(c));
    n = read(fd1, &c, sizeof(c));
    printf("n = %ld, c = %c\n", n ,c);
    close(fd1);
    return 0;
}
```

n = 1, c = S

ics.txt: IIS-+1S

6. What is the output of the program?

```c
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>

#define FILE_NAME "ics.txt"
#define COUNT 7

int main() {
    char c = 'a';
    ssize_t n;
    int fd[COUNT];

    fd[0] = open(FILE_NAME, O_RDONLY, 0);

    for (int i = 1; i < COUNT; i++) {
        fd[i] = fd[0] + i;
        dup2(fd[0], fd[i]);
    }

    for (int i = 0; i < COUNT; i++) {
        n= read(fd[i], &c, sizeof(c));
    }

    printf("n = %ld, c = %c\n", n ,c);

    for (int i = 0; i < COUNT; i++) {
        close(fd[i]);
    }
    return 0;
}
```

n = 1, c = S

7. What is the output of the program? What is the final content of **ics.txt**?

```c
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>

#define FILE_NAME "ics.txt"

int main() {
    char c = 'a';
    ssize_t n;

    int fd1 = open(FILE_NAME, O_RDWR | O_APPEND, 0);
    read(fd1, &c, sizeof(c));
    write(fd1, &c, sizeof(c));
    n = read(fd1, &c, sizeof(c));
    printf("n = %ld, c = %c\n", n ,c);
    close(fd1);
    return 0;
}
```

n = 0, c = I

ics.txt: ICS-+1SI

8. What is the output of the program?

```c
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>
#include <sys/wait.h>

#define FILE_NAME "ics.txt"

int main() {
    char c = 'a';
    ssize_t n;

    int fd1 = open(FILE_NAME, O_RDONLY, 0);
    int fd2 = open(FILE_NAME, O_RDONLY, 0);
    dup2(fd1, fd2);

    for (int i = 0; i < 4; i++) {
        read(fd1, &c, sizeof(c));
    }

    for (int i = 0; i < 2; i++) {
        pid_t pid;

        pid = fork();
        if (pid) {
            wait(NULL);
            continue;
        } else {
            read(fd1, &c, sizeof(c));
            close(fd1);
            close(fd2);
            return 0;
        }
    }

    n = read(fd2, &c, sizeof(c));
    printf("n = %ld, c = %c\n", n ,c);
    close(fd1);
    close(fd2);
    return 0;
}
```

n = 1, c = S