CS 572: Computer Architecture **(Due: April 14th, 2021)**

Submission Homework 1

Instructor: Prof. Justin Goins *Name:* Rashmi Jadhav, *Student ID:* 934-069-574

1. The algorithm for determining whether two given numbers form an amicable pair was implemented as follows:

```
/* Find square root in O(log N) using binary search space division */
int getSquareRoot(int n) {
    int left = 2, right = n / 2, mid;
    long square;

    if (n < 2) {
        return n;
    }
    while (left <= right) {
        mid = left + (right - left) / 2;
        square = (long) mid * mid;
        if (square > n) {                   // Move to left half
            right = mid - 1;
        } else if (square < n) {            // Move to right half
            left = mid + 1;
        } else {                            // Found square root
            return mid;
        }
    }
    return right;
}

/* Get all proper divisors for the number except itself */
long getDivisorsSum(int number) {
    long sum = 1;                           // Consider 1 to be the proper d
    int i = 2;
    while (i <= getSquareRoot(number)) {
        if ((number % i) == 0) {            // Add divisor i
            sum += i;
            if (number / i != i) {
                sum += number / i;          // Add divisor number / i
            }
        }
        i++;
    }
    return sum;
}

/*
   Returns 1 if the two integers form an amicable pair, 0 otherwise.
   This code needs to function correctly regardless of the ordering.
   For example, check_amicable(220, 284) should return 1, as should
   check_amicable(284, 220).
   This function must work correctly for all integers up to 2 billion.
   Be sure that the function prototype remains intact.
 */
int check_amicable(int a, int b) {
    long sumA, sumB;
    sumA = getDivisorsSum(a);
    sumB = getDivisorsSum(b);

    return ((int)sumA == b && (int)sumB == a) ? 1 : 0;
}
```

2. The code was setup and run on the **flip.engr.oregonstate.edu** server:

```
flip3 ~/CS572/hw1 1030$ gcc -std=c11 ./amicable_pairs_v1.c
flip3 ~/CS572/hw1 1031$ a.out; echo $?
1
flip3 ~/CS572/hw1 1032$
```

3. The algorithm was checked for correctness by taking some sample input cases with different **num_a** and **num_b**:

```
flip3 ~/CS572/hw1 1026$ gcc -std=c11 ./amicable_pairs_v1.c
flip3 ~/CS572/hw1 1027$ a.out; echo $?
num_a: 1        num_b: 2        are amicable?: 0
num_a: 220      num_b: 284      are amicable?: 1
num_a: 5564     num_b: 5020     are amicable?: 1
num_a: 17296    num_b: 18416    are amicable?: 1
num_a: 1987985835       num_b: 1974754485       are amicable?: 1
num_a: 1982313333       num_b: 1892277387       are amicable?: 1
num_a: 1892277387       num_b: 1982313333       are amicable?: 1
```

4. **num_a** and **num_b** were set to **1982313333** and **1892277387** respectively and the non-optimized 64-bit x86 code was generated using the commands:

```
flip3 ~/CS572/hw1 1040$ gcc -std=c11 ./amicable_pairs_v1.c
flip3 ~/CS572/hw1 1041$ gcc -m64 -fno-asynchronous-unwind-tables -std=c11 -O0
-S -o ./x86_64bit.asm ./amicable_pairs_v1.c
flip3 ~/CS572/hw1 1042$ gcc -m64 -fno-asynchronous-unwind-tables -std=c11 -O0
-o ./x86_64bit.exe ./amicable_pairs_v1.c
flip3 ~/CS572/hw1 1043$ ls
amicable_pairs_v1.c  amicable_pairs_v2.c  a.out  x86_64bit.asm  x86_64bit.exe
flip3 ~/CS572/hw1 1044$
```

5. The **check_amicable** function in **x86_64bit.asm** has about **24** instructions excluding labels and directives:

```
check_amicable:
        pushq   %rbp
        movq    %rsp, %rbp
        subq    $24, %rsp
        movl    %edi, -20(%rbp)
        movl    %esi, -24(%rbp)
        movl    -20(%rbp), %eax
        movl    %eax, %edi
        call    getDivisorsSum
        movq    %rax, -8(%rbp)
        movl    -24(%rbp), %eax
        movl    %eax, %edi
        call    getDivisorsSum
        movq    %rax, -16(%rbp)
        movq    -8(%rbp), %rax
        cmpl    -24(%rbp), %eax
        jne     .L14
        movq    -16(%rbp), %rax
        cmpl    -20(%rbp), %eax
        jne     .L14
        movl    $1, %eax
        jmp     .L15
.L14:
        movl    $0, %eax
.L15:
        leave
        ret
        .size   check_amicable, .-check_amicable
        .globl  main
        .type   main, @function
```

6. The amount of time spent by the CPU in **user mode** whilst running the generated **x86_64bit.exe** was **0.023s**:

```
flip3 ~/CS572/hw1 1048$ time x86_64bit.exe

real    0m0.026s
user    0m0.023s
sys     0m0.001s
```

7. The commands were run to generate assembly code occupying less space:

```
flip3 ~/CS572/hw1 1048$ gcc -m64 -fno-asynchronous-unwind-tables -std=c11 -Os -S -o ./x
86_64bit_s.asm ./amicable_pairs_v1.c
flip3 ~/CS572/hw1 1049$ gcc -m64 -fno-asynchronous-unwind-tables -std=c11 -Os -o ./x86_
64bit_s.exe ./amicable_pairs_v1.c
flip3 ~/CS572/hw1 1050$ ls
amicable_pairs_v1.c  a.out          x86_64bit.exe    x86_64bit_s.exe
amicable_pairs_v2.c  x86_64bit.asm  x86_64bit_s.asm
flip3 ~/CS572/hw1 1051$
```

8. The **check_amicable** function in **x86_64bit_s.asm** now has about **20** instructions excluding labels and directives. We notice that the instructions count reduced from 24 to 20 and hence the optimized code is 1.2x more efficient by 20% in terms of instruction count:

```
check_amicable:
        pushq   %r12
        pushq   %rbp
        movl    %edi, %ebp
        pushq   %rbx
        movl    %esi, %ebx
        call    getDivisorsSum
        movl    %ebx, %edi
        movq    %rax, %r12
        call    getDivisorsSum
        xorl    %edx, %edx
        cmpl    %r12d, %ebx
        jne     .L18
        xorl    %edx, %edx
        cmpl    %eax, %ebp
        sete    %dl
.L18:
        popq    %rbx
        popq    %rbp
        movl    %edx, %eax
        popq    %r12
        ret
        .size   check_amicable, .-check_amicable
        .section        .text.startup,"ax",@progbits
        .globl  main
        .type   main, @function
```

9. The amount of time spent by the CPU in **user mode** whilst running the newly generated **x86_64bit_s.exe** was **0.028s**. This is slower than the previously generated code's execution. Surprisingly, as instructions reduced by 20%, time taken was increased by 20%:

```
flip3 ~/CS572/hw1 1053$ time x86_64bit_s.exe

real    0m0.029s
user    0m0.028s
sys     0m0.001s
flip3 ~/CS572/hw1 1053$
```

10. The commands to generate 32-bit and 64-bit versions that are optimized to minimize execution time were run:

```
[flip3 ~/CS572/hw1 1056$ gcc -m32 -fno-asynchronous-unwind-tables -std=c11 -O3 -o ./x86_
32bit_3.exe ./amicable_pairs_v1.c
[flip3 ~/CS572/hw1 1057$ gcc -m32 -fno-asynchronous-unwind-tables -std=c11 -O3 -S -o ./x
86_32bit_3.asm ./amicable_pairs_v1.c
[flip3 ~/CS572/hw1 1058$ gcc -m64 -fno-asynchronous-unwind-tables -std=c11 -O3 -o ./x86_
64bit_3.exe ./amicable_pairs_v1.c
[flip3 ~/CS572/hw1 1059$ ls
amicable_pairs_v1.c  a.out             x86_32bit_3.exe  x86_64bit.asm  x86_64bit_s.asm
amicable_pairs_v2.c  x86_32bit_3.asm   x86_64bit_3.exe  x86_64bit.exe  x86_64bit_s.exe
flip3 ~/CS572/hw1 1060$
```

11. The user mode CPU time of **x86_32bit_3.exe** is **2 minutes and 54.779 seconds = 174.779 seconds** which is much slower than **x86_64bit_3.exe** version which takes just **0.010 seconds**.

```
[flip3 ~/CS572/hw1 1062$ time x86_32bit_3.exe

real    2m54.989s
user    2m54.779s
sys     0m0.190s
[flip3 ~/CS572/hw1 1063$ time x86_64bit_3.exe

real    0m0.013s
user    0m0.010s
sys     0m0.001s
flip3 ~/CS572/hw1 1064$
```

12. All of the commonly used 32-bit registers **eax, ebx, ecx, edx, esi, edi, ebp,** and **esp** are used in the generated **x86_32bit_3.asm**.

13. The MIPS assembly code **sample_mips.s** was generated:

```
flip3 ~/CS572/hw1 1066$ mips64-linux-gnu-gcc -mabi=32 -mips32 -static -std=c11 -Os -S
 -o ./sample_mips.s ./amicable_pairs_v1.c
flip3 ~/CS572/hw1 1067$ ls
amicable_pairs_v1.c  sample_mips.s    x86_64bit_3.exe  x86_64bit_s.asm
amicable_pairs_v2.c  x86_32bit_3.asm  x86_64bit.asm    x86_64bit_s.exe
a.out                x86_32bit_3.exe  x86_64bit.exe
```

14. The **x86_32bit_3.asm** utilized around **138** instructions for the **check_amicable** function whereas MIPS assembly utilized only **22** instructions. MIPS has much lesser instructions compared to 32 bit asm.

15. Out of the 32 32-bit MIPS registers, `$2 ($v0)` , `$3 ($v1)`, `$4 ($a0)`, `$5 ($a1)`, `$6 ($a2)`, `$16 ($s0)`, `$17 ($a1)`, `$18 ($a2)`, `$29 ($sp)`, `$31 ($ra)` were utilized in the **sample_mips.s**.

16. The faster algorithm at the expense of memory for determining whether two given numbers form an amicable pair was implemented as follows:

```
/* Since first array is sorted, apply binary search to spot a or b in first array */
int binarySearch(const int arr[], int size, int number) {
    int left = 0, right = size - 2, mid;
    while (left <= right) {
        mid = left + (right - left) / 2;                // To avoid int overflow
        if (arr[mid] == number) {
            return mid;
        }
        if (arr[mid] < number) {
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }
    return -1;
}

/*
    Returns 1 if the two integers form an amicable pair, 0 otherwise.
    This code needs to function correctly regardless of the ordering.
    For example, check_amicable(220, 284) should return 1, as should
    check_amicable(284, 220).
    This function must work correctly for all integers up to 2 billion.
    Be sure that the function prototype remains intact.
*/
int check_amicable(int a, int b) {
    int arraySize = sizeof(first) / sizeof(first[0]);
    int aIndex = binarySearch(first, arraySize, a);                 // Search a in fi
    if (aIndex == -1) {
        aIndex = binarySearch(first, arraySize, b);                 // If a not founc
    }
    if (aIndex != -1 && (first[aIndex] == a && second[aIndex] == b)     // Found both a a
        || (first[aIndex] == b && second[aIndex] == a)) {
        return 1;
    }
    return 0;                                                       // Any other case
}
```

17. An optimized 64-bit version was compiled out of this faster v2 algorithm and it took as low as **0.001 seconds** of **CPU user mode**. Compared to version 1 (0.01s), version 2 is **10x** faster.

```
[flip1 ~/CS572/hw1 1007$ time v2x86_64bit.exe

real    0m0.005s
user    0m0.001s
sys     0m0.002s
flip1 ~/CS572/hw1 1008$
```