CEC 470 **HW 1** **Due: Friday 16th September (By 11:59 pm)**

Total Points: / 30 Name: Jeremiah Webb

- *Late Day Policy:*
  You have **2 late days policy** with cost of 20% of the assignment grade. Past that the submissions are not accepted
- Turn in an **electronic copy** in PDF format on Canvas
- Show all the calculations and steps

1. **Stored Program Computer.** What is a stored program computer? (1 points)
   A stored program computer stores specific program instructions in a binary number format in a memory device, most likely RAM on the motherboard or registers & cache on the CPU. The idea of a stored program computer was initially introduced by John Von Neumann.

2. **Processor Performance.** What are the two ways that can be used by the processor designers to increase the performance of the microprocessor? (2 points)
   Physically put memory and processor cache close to the processor. Ensure that memory access is optimized in programs, unnecessary access to memory can cause the processor to slow down. Integrating processor pipelining where the processor can effectively implement instruction level parallelism. A designer can also increase the clock cycle rate or decrease the clock cycle time to increase performance of the microprocessor.

3. **Processor Performance.** What are the obstacles that become more significant with the increase in clock and logic density of microprocessor? *Please elaborate each point inorder to get the full credit*. (3 points)

A major obstacle would be power consumption and heat. The power density increases with logic density and the increasing clock speed can increase the amount of heat generation. This can be especially difficult to implement on mobile phones and laptops, for example. Another potential obstacle could be the limitation of memory, if memory does not have speed or storage, the microprocessor may throttle its computations and waste clocks as a result. In addition, the RC delay, the speed at which the physical electrons flow between resistors, and the capacitance of the wires connecting them can cause slowdowns for the CPU. The more wires that interconnect, the increased resistance. The closer the wires are together, the increased capacitance.

4. **Moore's Law.** Explain Moore's law? (2 points)

   Gordon Moore predicted that the number of transistors that can be integrated into a single chip would double every two years.

5. **Computer Architecture.** What is Instruction Set Architecture (ISA)? (1 point)

The instruction set architecture is the abstract model that defines how a CPU can be controlled by software. The ISA acts as an interface between the hardware of the computer and the software. It defines the supported data types in programs, the registers that users can program with, how the hardware manages main memory and how the CPU does I/O operations.

6. **Clock time.** If clock frequency is 900MHz, calculate the *cycle time* (1 point).

$$\frac{1}{frequency \cdot 10^6} = Clock\ Time$$

$\left(\frac{1}{900 \cdot 10^6}\right)$ = 1.111 nanoseconds

7. **Clock Speed.** A program runs in 15 seconds on Computer A, which has a clock frequency of 600 MHz. We are trying to help a computer designer build a new machine B, that will run this program in 6 seconds. One thing the designer can do is increase the clock frequency which will affect the CPU design causing machine B to require 1.5 times as many clock cycles as machine A for the same program. What clock rate should we tell the computer designer to target?
   (4 points)

   Computer A:
   Program Time: 15 seconds
   Clock Frequency: 600 MHz

   Computer B:
   Program Time: 6 seconds
   1.5 times as many clock cycles than A

$$\frac{15\ seconds}{6\ seconds} = \frac{x}{600\ MHz\ *\ 1.5}$$

$$x = 2250\ MHz$$

The clock rate should be at least 2250 MHz.

8. **Performance and CPI.** Assume a typical program has the following instruction type breakdown:
    a. 30% loads
    b. 10% stores
    c. 50% adds
    d. 8% multiplies
    e. 2% divides

    Assume the current-generation processor has the following instruction latencies:

    f. loads: 4 cycles
    g. stores: 4 cycles
    h. adds: 2 cycles
    i. multiplies: 16 cycles
    j. divides: 50 cycles

    If for the next-generation design you could pick one type of instruction to make twice as fast (half the latency), which instruction type would you pick? Why? (4 points)

    Initial instruction latencies:

    30 * 4 = 120 cycles Loads

    with 2 clock cycles

    30 * 2 = 60 cycles Loads (Upgraded)

    10 * 4 = 40 cycles Stores

    50 * 2 = 100 cycles Adds

    8 * 2 = 16 cycles Multiplies

    2 * 50 = 100 cycles Divides

    The program has a 30% “loads” instruction type and has a latency of 4 clock cycles, the best decision would be to make the loads instruction twice as fast, and reduce it to 2 clock cycles. Using weighted math above we can see that reducing the loads instruction type to 2 cycles, instead of 120 cycles being used, only 60 cycles would be used. This is proportionally greater than any other reduction for other instructions.

9. **Amdahl's Law.** A program has 10% divide instructions. All non-divide instructions take one cycle. All divide instructions take 20 cycles. (6 points)
    a. What is the CPI of this program on this processor?
    b. What percent of time is spent just doing divides?
    c. What would the speedup be if we sped up divide by 2x?
    d. What would the speedup be if we sped up divide by 5x?
    e. What would the speedup be if we sped up divide by 20x?
    f. What would the speedup be if divide instructions were infinitely fast (zero cycles)?

a. $\frac{(20+90 * 1)}{100} = 1.1\,CPI$

b. If we assume 100 instruction program, it means that 90 instructions are non divide and 10 instructions are divide. The 90 non divide insutrctions take 90 cycles whereas the 10 divide instruction takes 20 cycles. Therefore the percent time is

$$\left(\frac{20\,amount\ of\ divide\ cycles}{110\,total\ cycles}\right) * 100 = 18.18\,\%$$

c. If we speed up divide by 2 it means it would only take 10 cycles to complete the divide instructions. So the new cycle would be 100 cycles.

$\frac{(110\ previous\ cycle)}{100\ new\ cycle} = 110\,\%\ faster$ or 1.1 times faster

d. If we speed up divide by 5, it then means it would only take 4 cycles to complete the divide instructions. So the new amount of cycles would be 94.

$\frac{(110\ previous\ cycle)}{94\ new\ cycle} = 117\,\%\ faster$ or 1.17 times faster

e. If we speed up divide by 20, it means that it would only take 1 cycle to complete the divide instructions. So the new amount of cycles would be 91.

$\frac{(110\ previous\ cycle)}{91\ new\ cycle} = 121\,\%\ faster$ or 1.21 times faster.

f. If all divide instructions take zero cycles, the new amount of cycles would be 90.

$\frac{(110\ previous\ cycle)}{90\ new\ cycle} = 122\,\%\ faster$ or 1.22 times faster.

10. **Performance and ISA.** Chip A executes the ARM ISA and has a 2.5Ghz clock frequency. Chip B executes the x86 and has a 3Ghz clock frequency. Assume that on average, programs execute 1.5 times as many ARM instructions than x86 instructions. (6 points)
    a. For Program P1, Chip A has a CPI of 2 and Chip B has a CPI of 3. Which chip is faster for P1? What is the speedup for Program P1?
    b. For Program P2, Chip A has a CPI of 1 and Chip B has a CPI of 2. Which chip is faster for P2? What is the speedup for Program P2?
    c. Assuming that Programs P1 and P2 are equally important workloads for the target market of this chip, which chip is "faster"? Calculate the average speedup.

a. CPIa = 2 CPIb = 3 Assuming 100 x86 Instructions => 150 ARM

A seconds = $\left(\frac{(2 * 150 * 1)}{2.5}\right) * 10^{-9} = 120 * 10^{-9} seconds$

B seconds = $\left(\frac{(2 * 100 * 1)}{3}\right) * 10^{-9} = 100 * 10^{-9} seconds$

B is faster in this case.

Speed up = 120/100 = 1.2 times

b. CPIa = 1 CPIb = 2

A seconds = $\left(\frac{(1 * 150 * 1)}{2.5}\right) * 10^{-9} = 60 * 10^{-9} seconds$

B seconds = $\left(\frac{(2 * 100 * 1)}{3}\right) * 10^{-9} = 66.67 * 10^{-9} seconds$

A is faster in this case.

Speed up = 66.66/60 = 1.11

c. The average instruction time of Chip B is less than the average time of Chip A, so Chip B will be faster, on average.

Average Instruction Time for A = $\frac{1.2ns + .6ns}{2} = .9ns$

Average Instruction Time for B = $\frac{1ns + .66ns}{2} = .83ns$

Average Speedup = $\frac{.9\,ns}{.83} = 1.08$

11. **Bonus Question. Compiler Activity** (Bonus: 4 points)
In this exercise, you will compile a couple small pieces of code and compare the assembly code generated. You can use an online tool named "gcc explorer" by Matt Godbolt, to look at the generated assembly code and compare the output with several different configurations. To use this tool, navigate to: http://gcc.godbolt.org.

    11.1 Select x86-64 gcc 12.2 for the compiler, clear out compiler options, and copy the following code into the code editor:

```
int testFunction(int* input, int length) {
  int sum = 0;
  for (int i = 0; i < length; ++i) {
    sum += input[i];
  }
  return sum;
}
```

Copy/paste the assembly output into your solution.

    11.2 **x86 vs. ARM Assembly:** Using the same code as the last problem, change the compiler option to ARM gcc 10.3.1 (2021.10 none). Compare the number of instructions with previous x86 compiler output in 11.1. Notice how x86 uses fewer instructions than ARM. What do you think is the reason, explain (Google to learn more about x86 and ARM).

x86 Assembly:

```
testFunction(int*, int):
        push    rbp
        mov     rbp, rsp
        mov     QWORD PTR [rbp-24], rdi
        mov     DWORD PTR [rbp-28], esi
        mov     DWORD PTR [rbp-4], 0
        mov     DWORD PTR [rbp-8], 0
        jmp     .L2
.L3:
        mov     eax, DWORD PTR [rbp-8]
        cdqe
        lea     rdx, [0+rax*4]
        mov     rax, QWORD PTR [rbp-24]
        add     rax, rdx
        mov     eax, DWORD PTR [rax]
        add     DWORD PTR [rbp-4], eax
```

```
        add     DWORD PTR [rbp-8], 1
.L2:
        mov     eax, DWORD PTR [rbp-8]
        cmp     eax, DWORD PTR [rbp-28]
        jl      .L3
        mov     eax, DWORD PTR [rbp-4]
        pop     rbp
        ret
```

ARM Assembly:

```
           testFunction(int*, int):
        str     fp, [sp, #-4]!
        add     fp, sp, #0
        sub     sp, sp, #20
        str     r0, [fp, #-16]
        str     r1, [fp, #-20]
        mov     r3, #0
        str     r3, [fp, #-8]
        mov     r3, #0
        str     r3, [fp, #-12]
.L3:
        ldr     r2, [fp, #-12]
        ldr     r3, [fp, #-20]
        cmp     r2, r3
        bge     .L2
        ldr     r3, [fp, #-12]
        lsl     r3, r3, #2
        ldr     r2, [fp, #-16]
        add     r3, r2, r3
        ldr     r3, [r3]
        ldr     r2, [fp, #-8]
        add     r3, r2, r3
        str     r3, [fp, #-8]
        ldr     r3, [fp, #-12]
        add     r3, r3, #1
        str     r3, [fp, #-12]
        b       .L3
.L2:
        ldr     r3, [fp, #-8]
        mov     r0, r3
        add     sp, fp, #0
```

```
ldr    fp, [sp], #4
bx     lr
```

Because ARM processors belong to the “Reduced Instruction Set Computing” (RISC) architecture, this specific instruction set architecture splits activities into simple instructions, where a single instruction is executed in one clock cycle. This is in contrast to x86 where those processors follow the “Complex Instruction Set Computing” (CISC) architecture. This architecture follows an instruction set where complex instructions are processed in a single step in multiple clock cycles, this allows the computer to focus more on process efficiency. Furthermore, ARM uses fewer registers to do its processing, while x86 uses more registers to achieve it. ARM, however, takes advantage of this less consumption of registers to achieve very low power consumption and generate less heat. Thus, the instructions will be different amongst the different architectures, even for the same function.