COMP4300 Computer Architecture
Fall 2022
Homework 1

1. a. (15 points) Suppose that a given optimization results in an OVERALL speedup of 1.5 over the original design. If the optimization speeds up loads and stores, which collectively accounted for 80% of the execution time BEFORE the optimization, by what factor were loads and stores sped up by the optimization?

$Speedup_{over} = 1/((1 – frac) + (frac/speedup_{enhanced}))$

$1.5 = 1/((1 – 0.8) + (0.8/speedup_{enhanced}))$

$1.5(0.2 + (0.8/ speedup_{enhanced})) = 1$

$0.8/ speedup_{enhanced} = 7/15$

$speedup_{enhanced} = 0.8/(7/15)$

$speedup_{enhanced} = 1.71428571429$

b. (5 points) If, in problem one, the described optimization could make loads and stores take no time at all (not realistic, just for the sake of argument), what would the overall speedup to the execution time be?

$Speedup_{over} = 1 / (1 – 0.8) = 5$

2. (15 points) Suppose a given server computer has a distributed file system on 4 disk drives. The distribution is for speed, not redundancy, so if any one of the four disk drives fails, the computer fails as a whole – it cannot serve any requests. Now suppose that that the MTTF for each of the individual disk drives is 1 year. What is the MTTF of the disk drive subsystem consisting of the drives working together. There is no other hardware in the subsystem except the disk drives.

MTTF = hours of operation / assets in use

MTTF = 365 / 4 = 91.25 Days

MTTF = 8760 / 4 = 2190 hours

3. For a particular computer, the CPI for certain types of instructions is as follows:

- ALU operations, 1 cycle, make up 20% of dynamic (run-time) instruction count
- Load/store operations, 5 cycles, 40% of dynamic instruction count
- Control flow, 3 cycles, 20% of dynamic instruction count
- All other, 2 cycles, 20% of dynamic instruction count

a. What is the average CPI? (10 points)

1(.2) + 5(.4) + 3(.2) + 2(.2) = 3.2 CPI

b. Suppose there is an optimization in which the CPI of load/store is reduced to 2, but cycle time is lengthened by 10%. What is the speedup due to this optimization? (15 points)
speedup = old average / new average

new CPI = 2 + (.1 *2) = 2.2

speedup = 3.2 / (1(.2) + 2.2(.4) + 3(.2) + 2(.2)) = 3.2 / 2.08 = 1.538

5. (15 points) For a PDP-8, generate assembly code to multiply the number in hex address 0x200 by 4, and store the result in address 0x201. The program should start in address 0x100. You can assume the number in 0x200 is positive and less than 0x100. You will need to consult the Internet for the PDP-8 mnemonics and instruction formats. Note in particular that the PDP-8 has no multiply instruction. Be sure to give the address of each instruction.

Hex address	mnemonic	binary(attempted, not needed)
0x100	**CLA**	**111 010 000 000**
0x101	**TAD I 0x105**	**001 110 000 101**
0x102	**RTL**	**111 000 000 110**
0x103	**DCA I 0x106**	**011 110 000 110**
0x104	**HLT**	**111 100 000 010**
0x105	**(Data stored here) 0x200**	**001 000 000 000**
0x106	**(Data stored here) 0x201**	**001 000 000 000**

6. a. (5 points) One of the last and most complicated 8-bit processors of the early microcomputer era, the Zilog Z-80, included an instruction whose mnemonic was LDIR. The direct ancestor of the Z-80, the 8080, did not implement this instruction. What did the instruction do? This is not in your textbook or lecture notes – consult the Internet.

Transfers a byte of data from the memory location addressed by the contents of the HL register pair to the memory location addressed by the DE register pair. Both register pairs are increment and BC register pair is decremented. If BC hits 0, the instruction is terminated. If BC doesn't equal 0, the program counter is decremented by two and the instruction is repeated. Interrupts are recognized and two refresh cycles are executed after each data transfer. When

the BC is set to 0 prior to instruction execution, the instruction loops through 64 KB. (LDIR = Load increment and repeat)

Description from Z80 user manual

b. (20 points) Write a short assembly program (text, not binary machine language; do not hand- or machine- assemble it) using LDIR to move 0x100 bytes starting at address 0xa000 to the range of 0x100 addresses starting 0xb000. Consult the Internet for Z-80 mnemonics, address modes, and registers. Use the RET instruction as the last instruction of your program. Be sure to give the address of each instruction (The EDTASM editor/assembler program used the ORIG assembler directive to specify the starting address of assembly code)

Hex address	**mnemonic**
0x100	**LD HL, 0xa000**
0x101	**LD DE, 0xb000**
0x102	**LD BC, 0x100**
0x103	**LDIR**
0x104	**RET**

Start by loading data for transfer starting at 0xa000 and loading the start address for the target location. Then loaded the length of the data string. Then did the transfer instruction and finally finished with RET command.

Modeled after below example from Z80 user manual.

When a 737-byte data string in memory location DATA must be moved to location BUFFER, the operation is programmed as follows:

```
LD      HL, DATA        ;START ADDRESS OF DATA STRING
LD      DE, BUFFER      ;START ADDRESS OF TARGET BUFFER
LD      BC, 737         ;LENGTH OF DATA STRING
LDIR                    ;MOVE STRING-TRANSFER MEMORY POINTED
                        ;TO BY HL INTO MEMORY LOCATION POINTED
                        ;TO BY DE INCREMENT HL AND DE,
                        ;DECREMENT BC PROCESS UNTIL BC = 0
```

EC: (20 points) For extra credit, discuss the design decision to include the LDIR instruction's inclusion in the Z-80 instruction set with regard to "bridging the semantic gap" (Google it) and with regard to RISC design principles.

According to my search, the semantic gap is the difference between high-level programming sets and the simple computing instructions that micro processors work with. The Z-80 probably included the LDIR instruction to help programmers bridge that gap in relation to some RISC design principles. Two examples being load and store, and simple instructions. I found these principles at the website linked below. I think the inclusion of LDIR helps bridge the semantic gap through being a simple instruction, also fulfilling the RISC principle of simple instructions. It is a useful and simple way to transfer data that seems fairly easy to understand. I also think it was a good design decision to include LDIR in order to help the RISC principle of load and store. The LDIR instruction seems to be a useful and efficient way to load and store data.

https://www.gartner.com/en/information-technology/glossary/risc-reduced-instruction-set-computer#:~:text=RISC%20has%20five%20design%20principles%3A&text=Single%2Dcycle%20execution%20%E2%80%94%20In%20most,has%20some%20fixed%20lower%20limit