COMP4300 Spring 2021
Homework 1 Solution Set
Due 11:59pm, Feb 12, 2021

Each problem worth 20 points

1. 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 PDP-8 programming card for mnemonics and instruction formats. Note in particular that the PDP-8 has no multiply instruction. Be sure to give the address of each instruction.

Solution:

0x100 CLA	111 010 000 000
0x101 TAD I 0x105 (ok to write 5, the offset)	001 110 000 101
0x102 RTL	111 000 000 110
0x103 DCA I 0x106 (ok to write 6)	011 110 000 110
0x104 HLT	111 100 000 010
0x105 (data stored here) 0x200	001 000 000 000
0x106 (data stored here) 0x201	001 000 000 001

As mentioned in an announcement it was also okay if you used the MUY instruction available in the EAE (extended arithmetic element) option on later PDP 's. Unlike all the regular PDP-8 instructions, this one takes 2 12-bit words, with the address of the operand to multiply by the accumulator in the 2nd word.

0x100 CLA	111 010 000 000
0x101 TAD I 0x106	001 110 000 110
0x102 MUY	101 100 000 101
0x103 (address of the 4 here) 0x107	000 100 000 111
0x104 DCA I 0x106	011 110 000 110
0x105 HLT	111 100 000 010
0x106 (data stored here) 0x200	

0x107 (data stored here) 0x004

2. From the following assembly language code for a 32-bit MIPS processor, generate the binary machine language for this code fragment. Consult the Internet, for example:

   http://max.cs.kzoo.edu/cs230/Resources/MIPS/MachineXL/InstructionFormats.html

   for the bit patterns for opcodes and register numbers (5 points/instruction).

```
Start:    LW R1,40(R2)
          LW R3,1000(R0)
          ADDU R1, R2, R3
          J Start:
```

(where Start is at 32-bit address 0x00001000)

** ORDER IS BIG-ENDIAN **

Tell what bits go in each memory address. Remember that an address holds 8 bits.

Solution: Code for MIPS, assuming hex values (16-bit) for 40 and 1000

addr opcode rs rt offset

0x1000 100011 00010 00001 0000000001000000

0x1004 100011 00000 00011 0001000000000000

opcode rs rt rd shamt func

0x1008 000000 00010 00011 00001 00000 100001

opcode bits 27-2 of target address

0x100c 000010 00000000000000010000000000

Dividing this up into bytes in big-endian (most significant byte first) gives

Addr	Data
0x100	10001100
0x1001	00000011
0x1002	00000000
0x1003	01000000
0x1004	10001100
0x1005	00000011
0x1006	00010000
0x1007	00000000
0x1008	00000000
0x1009	01000011
0x100a	00001000
0x100b	00100001
0x100c	00001000
0x100d	00000000
0x100e	00001000
0x100f	00000000

Binary for first two instructions, assuming decimal values (16-bit) for 40 and 1000

addr opcode rs rt offset

0x1000 100011 00010 00001 0000000000101000

0x1004 100011 00000 00011 0000001111101000

Dividing this up into bytes in big-endian (most significant byte first) gives

Addr	**Data**
0x100	10001100
0x1001	00000011
0x1002	00000000
0x1003	00101000
0x1004	10001100
0x1005	00000011
0x1006	00000011
0x1007	11101000

The other two instructions (addr 0x1008-0x100f) are the same.

3. Suppose a given optimization to the ALU speeds up execution for the system as a whole by a factor of 1.75. After optimization, ALU operations take up 1/6 of the total execution time. What fraction of execution time BEFORE OPTIMIZATION was taken up by ALU operations? What was the speedup factor to ALU operations due to the optimization?

Solution:

$$1.75 = \frac{1}{(1 - frac) + frac/speedup}$$

Since ALU operations take up 1/6 of the total time after optimization, that means that *(1-frac)*,

the unoptimized part, takes 5 times as long as *frac*/*speedup*. Thus

$$1.75 = \frac{1}{(1 - frac) + \frac{1}{5}(1 - frac)}$$

Solving for *frac*, we get

$$frac = 0.5238$$

This passes a sanity check since it is between 0 and 1 as it should be. Then we can plug *frac* back into the original equation and get

$$speedup = 5.7$$

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

   ALU operations, 2 cycles, make up 25% of dynamic (run-time) instruction count
   Load/store operations, 10 cycles, 30% of dynamic instruction count
   Control flow, 3 cycles, 20% of dynamic instruction count
   All other instructions, 1 cycle

   What is the average CPI?
   Suppose there is an optimization in which the CPI of load/store is reduced to 5, but cycle time is lengthened by 20%. What is the speedup due to this optimization?

Solution: Using the processor performance equation, we get

$$CPI = (0.25) \times 2 + (0.30) \times 10 + (0.20) \times 3 + (1 - 0.25 - 0.3 - 0.2) \times 1$$
$$= 0.5 + 3 + 0.6 + 0.25 = 4.35$$

Initially execution time *ET* is given by

$$ET = CT \times IC_d \times CPI = 4.35(CT)(IC_d)$$

If we make load/store take only 5 we get

$$CPI_{new} = (0.25) \times 2 + (0.30) \times 5 + (0.20) \times 3 + (1 - 0.25 - 0.3 - 0.2) \times 1$$
$$= 0.5 + 1.5 + 0.6 + 0.25 = 2.85$$

But the new CT is 1.2 of the old CT. Instruction cound is unchanged, so

$$ET_{new} = CT_{new} \times IC_d \times CPI_{new} = (1.2)(CT)(ID_d)(2.85) = 3.42(CT)(IC_d)$$

So the speedup, the ratio of the old and new execution times, is 4.35/3.42 = 1.27

5. Suppose for the problem in question 4, Load/store operations were made to take 1 cycle, without lengthening the cycle time. What would be the speedup due to that optimization?

$$CPI_{new} = (0.25) \times 2 + (0.30) \times 1 + (0.20) \times 3 + (1 - 0.25 - 0.3 - 0.2) \times 1$$
$$= 0.5 + 0.3 + 0.6 + 0.25 = 1.65$$

Since CT and IC are unchanged, the speedup is the ratio of the old and new CPI, or

4.35/1.65 = 2.64