**COMP4300 Spring 2021**

**Homework 1**

**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.

|  |  |  |
| --- | --- | --- |
| Address in hex | mnemonic | binary |
| 0x100 | CLA | 111 010 000 000 |
| 0x101 | TAD 0x200 | 001 000 000 000 |
| 0x102 | TAD 0x200 | 001 000 000 000 |
| 0x103 | TAD 0x200 | 001 000 000 000 |
| 0x104 | TAD 0x200 | 001 000 000 000 |
| 0x105 | DCA 0x201 | 011 000 000 001 |

Notes: 0x200 = 0b1000000000

0x201 = 0b 1000000001

Reasoning:

I’m adding what is in address 0x200 4 times to my memory and then storing it in 0x201.

Then the memory gets cleared. *I’m afraid my indirect addressing is wrong, but I hope that at least my logic makes sense.*

1. 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.

1. LW R1, 0x40(R2)

|  |  |  |  |
| --- | --- | --- | --- |
| LW | r2 | r1 | offset |
| 100011 | 00010 | 00001 | 0000000001000000 |

Encoded instructions: Binary: 10001100010000010000000001000000

Hex: 0x8C410040

1. LW R3, 0x1000(R0)

|  |  |  |  |
| --- | --- | --- | --- |
| LW | R0 | R3 | offset |
| 100011 | 00000 | 00011 | 0001000000000000 |

Encoded instructions: Binary: 10001100000000110001000000000000

Hex: 0x8C031000

1. ADDU R1, R2, R3

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| SPECIAL | R2 | R3 | R1 | 0 | ADDU |
| 000000 | 00010 | 00011 | 00001 | 00000 | 100001 |

Encoded Insruction in:

Binary: 00000000010000110000100000100001

Hex: 0x00430821

1. J Start

|  |  |
| --- | --- |
| J | target |
| 000010 | 00000000000001000000000000 |

Encoded Insruction in:

Binary: 00001000000000000001000000000000

Hex: 0x08001000

Our program in Hex:

0x8C410040

0x8C031000

0x00430821

0x08001000

Putting the program in memory (starting at 0x0001000):

|  |  |
| --- | --- |
| **Address** | **Data in hex** |
| **0x00001000** | 8C |
| **0x00001001** | 41 |
| **0x00001002** | 00 |
| **0x00001003** | 40 |
| **0x00001004** | 8C |
| **0x00001005** | 03 |
| **0x00001006** | 10 |
| **0x00001007** | 00 |
| **0x00001008** | 00 |
| **0x00001009** | 43 |
| **0x0000100A** | 08 |
| **0x0000100B** | 21 |
| **0x0000100C** | 08 |
| **0x0000100D** | 00 |
| **0x0000100E** | 10 |
| **0x0000100F** | 00 |

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?

Speed up execution of system = 1.75

ALU takes up 1/6 of execution time after the speedup.

We keep in mind that the rest of the system (5/6 of exec time) stayed the same.

Equation of new system:

1.75 = (5/6) + (1/6)\*X

X = 5.5  
This means that the ALU Operations have shrunk \*5.5 during optimization

So the size of ALU before was 1/6 \* 5.5 = 11/12

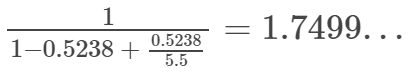
The rest of the execution time is 10/12

To find the percentage that that ALU took up before we divide 11/(10+11)

**=> Before optimization, ALU took up 52.38%**

**=> Its speedup factor is 5.5**

*Check if numbers add up with Amdahl’s Law:*



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

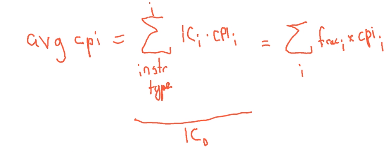
ALU operations, 2 cycles, make up 25% of dynamic (run-time) instruction count

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?



Following this logic:

Avg cpi = 2\*0.25 + 10 \* 0.3 + 3\*0.2 + 1\*0.25

**= 4.35**

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?

Same formula, different numbers:

Avg cpi = 2\*0.25 + 5 \* 0.3 + 3\*0.2 + 1\*0.25

**= 2.85**

Formula: *“Execution time = Total instruction count \* PCI \* cycle time”*

\*Suppose our old cycle time is T, therefor new cycle time is 1.2T

Old execution time = 100%\*4.35\*T = 4.35T

New Execution Time = 100%\*2.85\*1.2T = 3.42T (1.2T because of 20% increase)

Formula: *“Speedup = old exec time / new exec time”*

Speedup = 4.35T/3.42T  
Speedup = **1.27**

1. 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?

Avg pci = 2\*0.25 + 1\*0.3+ 3\*0.2 + 1\*0.25 = 1.65

Since cycle time is not lengthened,

Speedup = 4.35/1.65 = **2.63**