CSCI - 516 Exam 1

You are required to show the steps for calculations and instruction conversion.
1. (30%) Consider three different implementations of the same instruction set architecture. The instructions can be divided into four classes(details listed in the table). The clock rate for P1, P2, and P3 is 3.0 GHz, 2.5 GHz, and 2.1 GHz, respectively. The instruction Count of these three program are the same ($1.0 * 10^6$).

	Integer	Floating Point	Load/store	Branch
CPI	1	2	3	1
IC for program 1 running on P1	30%	40%	20%	10%
IC for program 2 running on P2	10%	60%	20%	10%
IC for program 3 running on P3	10%	60%	25%	5%

1). What is the definition of CPI?
CPI: Clock cycles per instructions. Average number of clock cycles per instruction for a program or program fragment.

2). What is the average CPI of each program?

$$CPI = \sum_{i=1}^{n} CPI_i * \frac{Instruction\ Count_i}{Instruction\ Count} \tag{1}$$

$CPI_{program\ 1} = 1 * 0.3 + 2 * 0.4 + 3 * 0.2 + 1 * 0.1 = 1.8$
$CPI_{program\ 2} = 1 * 0.1 + 2 * 0.6 + 3 * 0.2 + 1 * 0.1 = 2.0$
$CPI_{program\ 3} = 1 * 0.1 + 2 * 0.6 + 3 * 0.25 + 1 * 0.05 = 2.1$
3). What are the execution times of Program 1, 2, and 3?

$$Execution\ Time = \frac{Instruction\ Count * CPI}{Clock\ Rate} \tag{2}$$

$Execution\ Time_{program\ 1} = \frac{1.0*10^6*1.8}{3*10^9} = 0.6 * 10^{-3}s$
$Execution\ Time_{program\ 2} = \frac{1.0*10^6*2}{2.5*10^9} = 0.8 * 10^{-3}s$
$Execution\ Time_{program\ 3} = \frac{1.0*10^6*2.1}{2.1*10^9} = 1 * 10^{-3}s$
4). What are the MIPS ratings on each processor?

$$MIPS\ rating\ of\ a\ processor = (clock\ speed)/CPI \tag{3}$$

$MPIS\ rating\ of\ P1 = 3.0GHz\ /\ 1.8 = 1666.7\ MPIS$
$MPIS\ rating\ of\ P2 = 2.5GHz\ /\ 2 = 1250\ MPIS$
$MPIS\ rating\ of\ P3 = 2.1GHz\ /\ 2.1 = 1000\ MPIS$

2. (20%) Continue the scenario in the first question, and answer the following questions:
1). What is the definition of Amdhal's law? What is the equation for Amdhal's law?
Amdahl's Law is a rule stating that the performance enhancement possible with a given improvement is limited by the amount that the improved feature is used.

$$Exe\,Time\;After\;Improvement = \frac{Exe\;Time\;Affected\;by\;Improvement}{Amount of Improvement} + Exe\;Time\;unaffected \tag{4}$$

2). We try to optimize the execution of the programs on different processors. Answer if the speed-up can be achieved by optimizing the operations listed below.

	Optimized operations	Original ExeTime for Optimized Operations	Required Speed-up	Possible (Yes/No)
Program 1	Integer operations	$0.1 * 10^{-3}s$	2	No
Program 2	Floating point operations	$0.48 * 10^{-3}s$	2	Yes
Program 3	Load/Store operations	$0.35 * 10^{-3}s$	2	No

Program 1:

$$\frac{0.6 * 10^{-3}s}{2} = \frac{0.1 * 10^{-3}s}{n} + 0.5 * 10^{-3}s \tag{5}$$

where n is the improvement factor for Integer operations. Because 'n' is a negative number, it is not possible to achieve 2 times speed-up, by only improving integer operations.

Program 2:

$$\frac{0.8 * 10^{-3}s}{2} = \frac{0.48 * 10^{-3}s}{n} + 0.32 * 10^{-3}s \tag{6}$$

where n is the improvement factor for Floating Point operations. Here n = 6, which means it is possible to achieve 2 times' speed-up.

Program 3:

$$\frac{1 * 10^{-3}s}{2} = \frac{0.35 * 10^{-3}s}{n} + 0.75 * 10^{-3}s \tag{7}$$

where n is the improvement factor for Load/Store operations. Because n is a negative number, which means it is not possible to achieve 2 times' speed-up, by only improving Load/Store operations.

3. (20%) Convert the following assembly instructions to C code and writing the results on the memory in the figure below. Assume each location is 8 bytes wide and address start from 0xEF00 (Hexadecimal). X6 stores the base address of array A, and

X9	X10	X11	X12	X13	X14	X15	X16
a	b	c	d	e	f	g	h

You should be aware of which registers store Value, and which register store address. Fill out the number in the following figure.

```
MOV X9, #2               // a = 2;

MOV X10, #4              // b = 4;

STUR X9, [X6, #0]        // A[0] = 2;

LSL X11, X10, #3         // c = b * 8   ->   c = 4 * 8;

ADD X12, X6, X11         // d = &A[b]   ->   d = &A[4];

STUR X10, [X12, #0]      // A[4] = 4

ADD X13, X9, X10         // e = a + b;   ->   e = 2 + 4;

LSR X13, X13, #1         // e = 6 / 2

LSR X14, X11, #1         // f = c / 2

ADD X14, X6, X14         // f = &A[2]

STUR X13, [X14, #0]      // A[2] = 3

ORR X15, X9, X10         // g = a | b

STUR X15, [X6, #8]       // A[1] = g

ANDI X16, X9, X10        // h = a & b

STUR X16, [X6, #24]      // A[3] = h
```

Memory Cell	Address
2	0xEF00 (A[0])
6	0xEF08 (A[1])
3	0xEF10 (A[2])
0	0xEF18 (A[3])
4	0xEF20 (A[4])