Homework #1 **M1522.000800 System Programming**

Sample Solution

Question 1

Number Formats (Hexadecimal and Binary notations)

Perform the following number conversions and calculations.

example) `0x2015` in binary notation ('0x' represents hexadecimal number.)

=> 0b10000000010101 ('0b' represents binary number.)

a) 0b 1101 1110 1010 1101 1011 1110 1110 1111 in hexadecimal notation

=> 0xDEADBEEF

b) `0xFACE` in binary notation

=> 0b 1111 1010 1100 1110

c) Your Student-Number (encoded in decimal numbers) in hexadecimal notation

=> D.I.Y.

Question 2

Number Formats (Hexadecimal and Binary notations)

Fill in the missing entries in the following figure, giving the decimal, binary, and hexadecimal values of different byte patterns.

Decimal (0d)	Binary (0b)	Hexadecimal (0x)
0	0000 0000	00
2766	1010 1100 1110	ACE
191	1011 1111	BF
47806	1011 1010 1011 1110	BABE

Question 3
Number Formats (Two's Complement Binary Representations)

Fill in the missing entries in the following figure, giving the binary, and decimal values of different patterns. (Use Two's Complement as representing signed numbers, and all formats follow 8-bit.)

Decimal (0d)	Binary (0b)	Hexadecimal (0x)
- 34	1101 1110	DE
- 128	1000 0000	80
- 1	1111 1111	FF
- 84	1010 1100	AC

Question 4
Computer Architectures (Pipeline)

Explain why pipeline architectures are faster than sequential architectures in instruction-level parallelism.

=> CPU has many stages to process one instruction. However they are not executed at the same time. Sequential architectures allow only one instruction to monopolize all CPU components. On the other hands, Pipeline architectures allow each instruction to use each CPU components, i.e., pipeline architectures allow multiple instructions to share CPU components, and try to make it busy so that no CPU components are idle.

Question 5
Assembly Language Programming

Assume the following values are stored at the indicated memory addresses and registers:

Address	Value
0x1000	0x0000_0000
0x1004	0x0000_0000

Register	Value
%eax	0x0000_1000
%ecx	0x0000_0001
%edx	0x0000_0003

Fill in the following table showing the effects of the following instructions, both in terms of the register or memory location that will be updated and the resulting value. (All instructions follow AT&T syntax, and might affect the next instructions.)

Instruction	**Destination**	**Value**
`movl %eax, (%eax)`	MEM (0x1000)	0x0000_1000
`addl %ecx, %edx`	REG (%edx)	0x0000_0004
`movl %eax, (%eax, %edx, 1)`	MEM (0x1004)	0x0000_1000
`leal (%eax, %ecx, 4), %eax`	REG (%eax)	0x0000_1004
`addl %edx, (%eax)`	MEM (0x1004)	0x0000_1004

Question 6

Computer Architectures (Code Optimizations)

Consider the two C functions sum1() and sum2() below. Both compute the same result (the sum of all eight parameters), both use 7 addition operations to do so. Which one runs faster and why?

```
int sum1(int a,int b,int c,int d,
         int e,int f,int g,int h)
{
  int s = a;
  s += b;
  s += c;
  s += d;
  s += e;
  s += f;
  s += g;
  s += h;

  return s;
}
```

```
int sum2(int a,int b,int c,int d,
         int e,int f,int g,int h)
{
  int s1 = a+b;
  int s2 = c+d;
  int s3 = e+f;
  int s4 = g+h;

  s1 = s1 + s2;
  s3 = s3 + s4;
  s1 = s1 + s3;

  return s1;
}
```

=> sum2 is faster than sum1. Because of data dependencies, sum2 is less bounded and has more chances to process instructions simultaneously with pipelining. On the other hands, sum1 is serialized by a variable 's', and next instructions should be stalled by when previous one ends.