Solutions to Tutorial 1

1.1 Number representation

(1) Solutions:

binary number	unsigned	2's-complement
(a) $0111\ 1111_2$	127	127
(b) $1111\ \ 1111_2$	255	-1
(c) $0000\ 0000_2$	0	0
(d) $1000\ 0000_2$	128	-128
(e) $1111\ 1110_2$	254	-2

(2) Solutions: unsigned magnitude range **0 to 255**
2's complement range **-128 to 127**

(3) See lecture notes on the range of different C data types.

(4) Suggested solution:
(a) **`signed char`** .
(b) **`unsigned short int`**.
(c) **`unsigned long long int`**.
(d) **`_Bool`** from the stdbool.h header.

1.2 Hexadecimal number representation

(1) (a) and (d)
(2) (c) and (e)
(3) (a) **`-1`** (b) **`15`**
(4) Yes
(5) We can do this using sign extension.

1.3 Data representation in memory

(1) **`0x0002` (start address).** The format is **Little Endian.**

(2) "***Login:**". Any subset of this is also correct.

(3) (a) `r.i = ` **`0x67`** (1-byte value)
(b) **`r.j = 0x696E3A00`** (4-byte, MSByte at lowest address since big endian)
(c) **`r.a[0] = 0x61`** (1-byte, first element of 3 element array)
(d) **`r.a[2] = 0x63`** (1-byte, last element of 3 element array)

Solutions to Tutorial 1

1.4 Data and Address Busses

(1) 16 Mbyte

(2) Two bytes

(3)
```
struct rec {
  long int j;  %starts at even address by
  unsigned char i;  %swapping j and i
  char a[3];
};
struct rec r;
```

1.5 ARM Programmer's Model and Instruction Execution

(1) **32 bits.**

(2) 32-bit range in hexadecimal notation:

Range	unsigned	2's-complement
Largest	0xFFFFFFFF	0x7FFFFFFF
Smallest	0x00000000	0x80000000

(3) Yes, this is the set of User Mode registers. There are 16 registers (R0 – R15) and the CPSR. Several of the Rn registers have special dedicated functions.

(4) Description of each instruction:

```
MOVS R0,#0x00000001  ; I1 – move the 32-bit value of 1 to register R0
MOVS R1,#0xFFFFFFFF  ; I2 – move the negative value of -1 to R1
MOVS R2,#0x7FFFFFFF  ; I3 – move the value 0x7FFFFFFF into R2
ADDS R3,R0,R1        ; I4 – add R0 and R1 and put result in R3
ADDS R4,R0,R2        ; I5 – add R0 and R2 and put result in R4
```

(5) N flag set because the value 0xFFFFFFFF moved into R3 is negative.

(6) N flag clear because the value 0x7FFFFFFF moved into R4 is not negative.

(7) The `ADDS R3,R0,R1` instruction:
- a) `R3=0x0000000`.
- b) Correct if numbers are 2's complement signed numbers only.
- c) The Z and C flags are set.
- d) Result is zero and unsigned overflow has occurred.

(8) The `ADDS R4,R0,R2` instruction:
- e) `R4=0x8000000`.
- f) Correct if numbers are unsigned numbers only.
- g) The N and V flags are set.
- h) Result is negative and signed overflow has occurred.

(9) Try adding the value of `0x80000000` to itself. See what flags are set. Can you think of the combination of numbers?