# Solutions - Homework 1

(Due date: September 25th @ 5:30 pm)
Presentation and clarity are very important! Show your procedure!

## Problem 1 (15 pts)

- Complete the following table. We are representing positive integer numbers.

| Decimal | BCD (bits) | Binary | Hexadecimal |
|---|---|---|---|
| 766 | 0111 0110 0110 | 1011111110 | 2FE |
| 395 | 0011 1001 0101 | 110001011 | 18B |
| 172 | 0001 0111 0010 | 10101100 | AC |
| 249 | 0010 0100 1001 | 11111001 | F9 |

- Complete the following table. Use the fewest number of bits in each case:

| | REPRESENTATION | | |
|---|---|---|---|
| **Decimal** | **Sign-and-magnitude** | **1's complement** | **2's complement** |
| -247 | 111110111 | 100001000 | 100001001 |
| -51 | 1110011 | 1001100 | 1001101 |
| -111 | 11101111 | 10010000 | 10010001 |
| -180 | 110110100 | 101001011 | 101001100 |
| 101 | 01100101 | 01100101 | 01100101 |
| 55 | 0110111 | 0110111 | 0110111 |

## Problem 2 (20 pts)

- Perform the following additions and subtractions of 8-bit unsigned integers. Indicate every carry (or borrow) from $c_0$ to $c_8$ (or $b_0$ to $b_8$). For the addition, determine whether there is an overflow. For the subtraction, determine whether we need to keep borrowing from a higher byte.

**Example**:

- 54 + 210

```
                 c8=1 c7=1 c6=1 c5=1 c4=0 c3=1 c2=1 c1=0 c0=0
 54 = 0x36 =       0    0    1    1    0    1    1    0   +
210 = 0xD2 =       1    1    0    1    0    0    1    0
-------------------------------------------------------------
Overflow! ---->    1    0    0    0    0    1    0    0    0
```

- 77 - 194

```
                 Borrow out! ---->  b8=1 b7=0 b6=0 b5=0 b4=0 b3=0 b2=1 b1=0 b0=0
 77 = 0x4D =                        0    1    0    0    1    1    0    1   -
194 = 0xC2 =                        1    1    0    0    0    0    1    0
-------------------------------------------------------------------------------
                                    0    0    0    0    1    0    1    1
```

---

- 129 + 103
- 198 + 67

```
                 No Overflow  c8=0 c7=0 c6=0 c5=0 c4=0 c3=1 c2=1 c1=1 c0=0
129 = 0x81 =                   1    0    0    0    0    0    0    1   +
103 = 0x67 =                   0    1    1    0    0    1    1    1
-------------------------------------------------------------------------
232 = 0xE8 =                   1    1    1    0    1    0    0    0
```

```
                 Overflow! ---->  c8=1 c7=1 c6=0 c5=0 c4=0 c3=1 c2=1 c1=0 c0=0
198 = 0xC6 =                      1    1    0    0    0    1    1    0   +
 67 = 0x43 =                      0    1    0    0    0    0    1    1
-----------------------------------------------------------------------------
       0x09 =                     0    0    0    0    1    0    0    1
```

- 43 - 98
- 149 - 87

```
                 Borrow out! ---->  b8=1 b7=1 b6=0 b5=0 b4=0 b3=0 b2=0 b1=0 b0=0
 43 = 0x2B =                        0    0    1    0    1    0    1    1   -
 98 = 0x62 =                        0    1    1    0    0    0    1    0
-------------------------------------------------------------------------------
       0xC9 =                       1    1    0    0    1    0    0    1
```

```
                 No Borrow Out  b8=0 b7=1 b6=1 b5=1 b4=1 b3=1 b2=1 b1=0 b0=0
149 = 0x95 =                      1    0    0    1    0    1    0    1   -
 87 = 0x57 =                      0    1    0    1    0    1    1    1
-----------------------------------------------------------------------------
 62 = 0x3E =                      0    0    1    1    1    1    1    0
```

- Perform the following operations by representing the numbers in 2's complement representation using 8 bits. Indicate every carry (from $c_0$ to $c_8$). Determine whether the operation results in an overflow.
  - `101 + 23`
  - `98 - 62`
  - `-90 - 82`

---

All of the addition/subtraction operations in 2's complement arithmetic, regardless of whether the operands are positive or negative, can be expressed as an addition:

`98 - 62 = 98 + (-62)` `-90-82 = -90 + (-82)`

The first step is to represent the positive numbers using the 2's complement representation with 8 bits. This is a straightforward process: we take the unsigned number representation and attach a zero to the left.
The second step is to represent the negative numbers in 2's complement representation with 8 bits. To this end, we first obtain the 2's complement representation of the positive numbers. Then, we apply the 2's complement operation to get the 2's complement representation of the negative numbers:

```
62 = 0x3E = 00111110 → -62 = 0xC2 = 11000010
90 = 0x5A = 01011010 → -90 = 0xA6 = 10100110
82 = 0x52 = 01010010 → -82 = 0xAE = 10101110
```

Now, we are ready for the addition operations. This is a very simple step. The only difference is that the overflow is specified by $c_8 \oplus c_7$ (for 8 bits):

$c_8 \oplus c_7 = 0$ No Overflow

| | $c_8$=0 | $c_7$=0 | $c_6$=0 | $c_5$=0 | $c_4$=0 | $c_3$=1 | $c_2$=1 | $c_1$=1 | $c_0$=0 | |
|---|---|---|---|---|---|---|---|---|---|---|
| 101 = 0x65 = | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | | + |
| 23 = 0x17 = | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | | |
| 232 = 0x7C = | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | | |

$c_8 \oplus c_7 = 1$ Overflow!

| | $c_8$=1 | $c_7$=0 | $c_6$=1 | $c_5$=0 | $c_4$=1 | $c_3$=1 | $c_2$=1 | $c_1$=0 | $c_0$=0 | |
|---|---|---|---|---|---|---|---|---|---|---|
| -90 = 0xA6 = | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | | + |
| -82 = 0xAE = | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | | |
| 0x54 = | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | | |

$c_8 \oplus c_7 = 0$ No Overflow

| | $c_8$=1 | $c_7$=1 | $c_6$=0 | $c_5$=0 | $c_4$=0 | $c_3$=0 | $c_2$=1 | $c_1$=0 | $c_0$=0 | |
|---|---|---|---|---|---|---|---|---|---|---|
| 98 = 0x62 = | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | | + |
| -62 = 0xC2 = | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | | |
| 36 = 0x24 = | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | | |

- BCD Addition: We want to add these two decimal numbers using normal binary addition:
  - `6289 + 3098 = 9387`

- Convert the summation operands to their BCD representation.
- Add the binary numbers as if they were unsigned numbers.
- Specify which binary number we need to add to the previous summation so that we get the answer we are looking for (`9387`).

---

The idea here is to implement BCD addition using standard binary summation. So, we first add the BCD numbers using normal summation. Then, we "adjust" in order to get the proper BCD result. The "adjust" consists on adding a 0x6 to every nibble position where the summation resulted in a number larger than 9.

In the `"6289 + 3098 = 9387"` case, we found that the sum of two nibbles in the two least significant nibbles of the summation is larger than 9. Thus, we need to add the number `0x0066` to the whole summation.

| | $c_4$=0 | $c_3$=0 | $c_2$=1 | $c_1$=1 | $c_0$=0 | |
|---|---|---|---|---|---|---|
| 0x 6 2 8 9 | | | | | | + |
| 0x 3 0 9 8 | | | | | | |
| 0x 9 3 2 1 | | | | | | + |
| 0x 0 0 6 6 | | | | | | |
| 0x 9 3 8 7 | | | | | | |

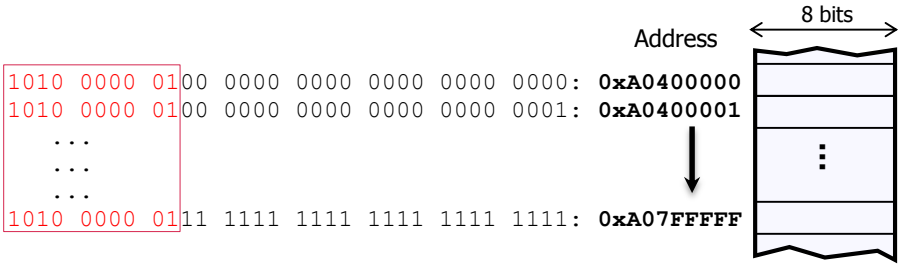
| | $c_{16}$=0 | $c_{15}$=1 | $c_{14}$=1 | $c_{13}$=0 | $c_{12}$=0 | $c_{11}$=1 | $c_{10}$=1 | $c_9$=0 | $c_8$=0 | $c_7$=1 | $c_6$=1 | $c_5$=0 | $c_4$=0 | $c_3$=0 | $c_2$=1 | $c_1$=1 | $c_0$=0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 1 1 0 | 0 | 0 | 1 | 0 | | 1 | 0 | 0 | | 1 | 0 | 0 | | 1 | 0 | 0 | 1 | + |
| 0 0 1 1 | 0 | 0 | 0 | 0 | | 1 | 0 | 0 | | 1 | 0 | 0 | | 1 | 0 | 0 | 0 | |
| 1 0 0 1 | 0 | 0 | 1 | 1 | | 0 | 0 | 1 | | 0 | 0 | 1 | | 0 | 0 | 0 | 1 | + |
| 0 0 0 0 | 0 | 0 | 0 | 0 | | 0 | 1 | 1 | | 0 | 1 | 1 | | 0 | 1 | 1 | 0 | |
| 1 0 0 1 | 0 | 0 | 1 | 1 | | 1 | 0 | 0 | | 0 | 1 | 1 | | 1 | 1 | 1 | 1 | |

# PROBLEM 3 (15 PTS)

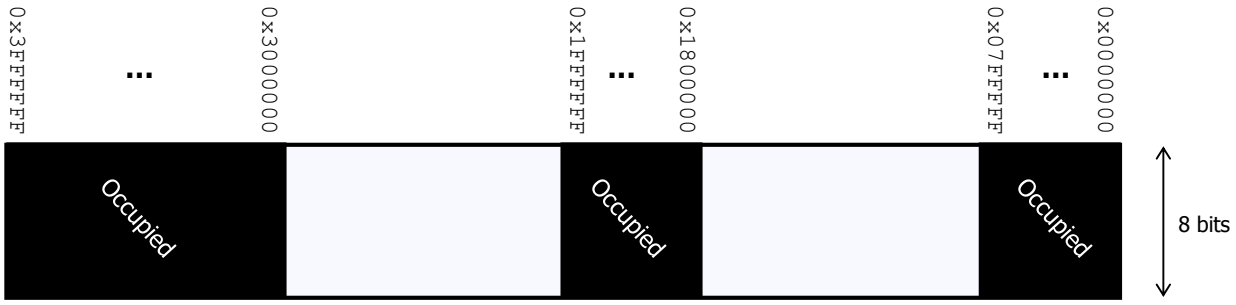
- A microprocessor has a 32-bit address line. The size of the memory contents of each address is 8 bits.
  - What is the address range (lowest to highest, in hexadecimal) of the memory space for this microprocessor? What is the size (in bytes, KB, or MB) of the memory space?
  - A memory device is connected to the microprocessor. Based on the size of the memory, the microprocessor has assigned the addresses `0xA0400000` to `0xA07FFFFF` to this memory device. What is the size (in bytes, KB, or MB) of this memory device? What is the minimum number of bits required to represent the addresses on this memory device?

---

- Address Range: `0x00000000` to `0xFFFFFFFF`.
  With 32 bits, we can address $2^{32}$ bytes, thus, we have $2^2 2^{30}$ = 4 GB of address space
- As per the figure below, we only need 22 bits for the address in the given range. Thus, the size of the memory device is $2^{22}$ = 4MB.

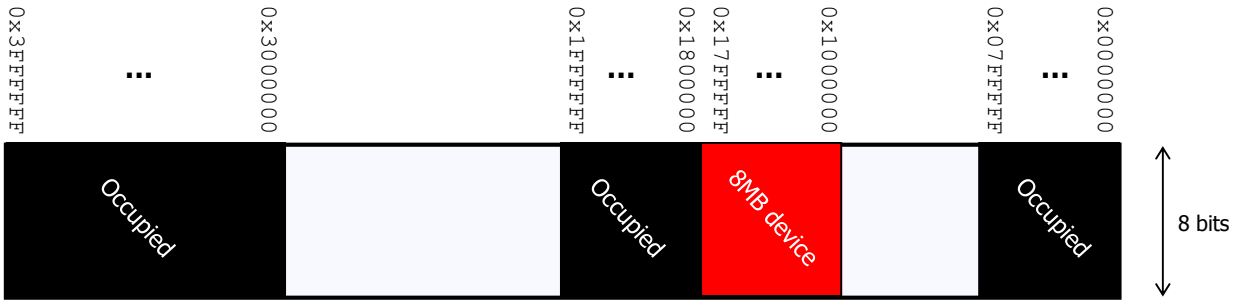



- The figure below depicts the complete memory space of a microprocessor:
  - What is the size (in bytes) of the memory space? What is the address bus size of the microprocessor?
  - If we have a memory chip of 8MB, how many bits do we require to address 8MB of memory (each memory address occupies 1 byte)?
  - If we want to connect the 8MB memory chip to the microprocessor, provide an address range (within the given memory space) that allows the 8MB of memory to be properly address. You are only allowed to use the non-occupied portions of the memory space as shown in the figure below.
  -




---

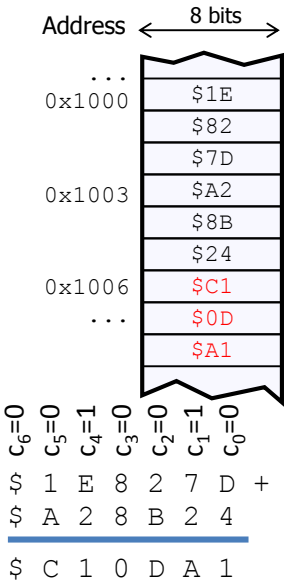
- Address Range: `0x0000000` to `0x3FFFFFF`. To represent all these addresses, we require 26 bits. So, the address bus size of the microprocessor is 26 bits. The size of the memory space is then $2^{26}$ = 64MB.
- 8MB memory device: 8MB = $2^3 2^{20} = 2^{23}$ bytes. Thus, we require 23 bits to address the memory device.
- If we had a memory space of 8MB, then with 23 bits, the address range would be: `0x000000` to `0x7FFFFF`. So, we need to place this range within the memory space from `$0000000` to `$3FFFFFF`. There are 4 options in the figure, and we picked the address range from: `0x1000000` to `0x17FFFFF`.



# PROBLEM 4 (15 PTS)

- Multi-precision addition: Write a set of instructions that properly perform the following unsigned addition: `$1E827D+$A28B24`.

  Your code first needs to store these constants in memory addresses `$1000` and `$1003` respectively (from higher byte to lower byte, as shown in the figure). The result (3 bytes) must be stored starting from memory address `$1006`. Complete the respective memory contents (in hexadecimal) in the figure.




**Assembly Code (there is more than one possibility):**

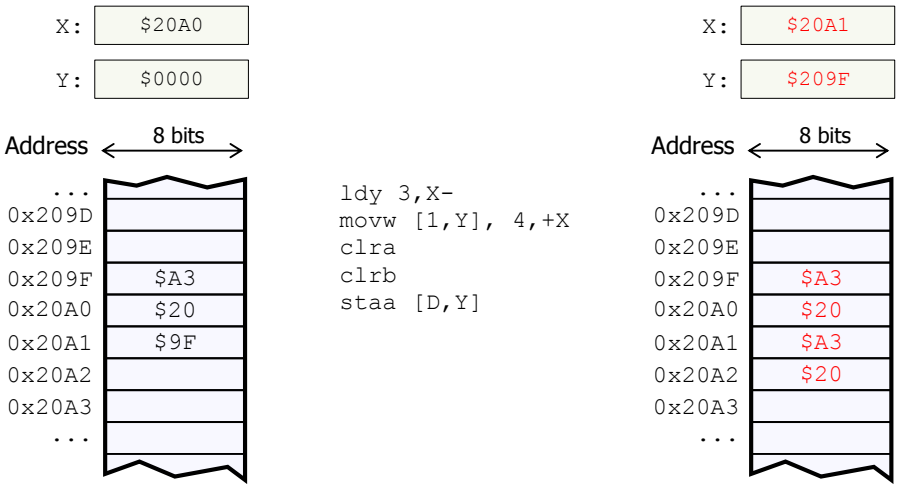
```
movw #$827D, $1001  ; m[$1001] ← $82, m[$1002] ← $7D
movb #$1E, $1000    ; m[$1000] ← $1E
movw #8B24, $1004   ; m[$1004] ← $8B, m[$1005] ← $24
movb #$A2, $1003    ; m[$1003] ← $A2

ldd $1001           ; D ← [$1001]:[$1002] = $827D
addd $1004          ; D ← [D] + [1004]:[1005] = $827D + $8B24, C ← 1
std $1007           ; m[$1007] ← DH = $0D , m[$1008] ← DL = $A1

ldaa $1000          ; A ← [$1000] = $1E
adca $1003          ; A ← [A] + [$1003] + C = $1E + $A2 + 1
staa $1006          ; m[$1006] ← [A] = $C1
```

# PROBLEM 5 (15 PTS)

- For the following set of instructions, provide the contents (in hexadecimal) of `X`, `Y` and relevant memory addresses after the last instruction has been executed. The figure on the left shows the values on `X`, `Y`, and memory before the first instruction is executed.




```
ldy 3,X-            ; Y ← [[X]]=[$20A0] = $209F, X ← $20A0-$3 = $209D
movw [1,Y],4,+X     ; Source Address = [$20A0]:[$20A1] = $209F
                    ;      X ← $209D + $4 = 20A1
                    ;      Source Data: [$209F:$20A0] = $A320
                    ; Destination Address: $20A1
                    ; Then: m[$20A1] ← $A3, m[$20A2] ← $20
clra                ; A ← $00
clrb                ; B ← $00
staa [D,Y]          ; Destination Address = [$0+$209F] = [$209F] = $A320
                    ; m[$A320] ← [A] = $00
```

# PROBLEM 6 (20 PTS)

Given the following set of instructions, complete the following:

- Register values (in hexadecimal format) as the instructions are executed.
- The state of the memory contents (in hexadecimal format) after the last instruction has been executed. Also, specify the memory address at which the contents of `D` are stored (last instruction).
- The addressing mode of each instruction. Be specific, if for example the addressing mode is indexed, indicate which one in particular. Note that the `movw` instruction uses two addressing modes.

| Addressing Mode | | A | B | X | Y |
|---|---|---|---|---|---|
| Inherent | `clra` | $00 | $00 | $000F | $0000 |
| Inherent | `clrb` | $00 | $00 | $000F | $0000 |
| Immediate | `ldx #$0F` | $00 | $00 | $000F | $0000 |
| Immediate | `ldy #$00` | $00 | $00 | $000F | $0000 |
| Immediate | `ldy #$1000` | $00 | $00 | $000F | $1000 |
| Immediate, Indexed - Constant Offset | `movw #$1F30,0,Y` | $00 | $00 | $000F | $1000 |
| Immediate | `ldab #149` | $00 | $95 | $000F | $1000 |
| Inherent | `sex b,d` | $FF | $95 | $000F | $1000 |
| Inherent | `nega` | $01 | $95 | $000F | $1000 |
| Indexed - Constant Offset | `addd 0,y` | $20 | $C5 | $000F | $1000 |
| Inherent | `iny` | $20 | $C5 | $000F | $1001 |
| Inherent | `exg x,y` | $20 | $C5 | $1001 | $000F |
| Indexed Indirect - 16-bit Offset | `std [-1,X]` | | | | |