EE443 - Embedded Systems

# Exercise - 4

Programming Background

**1.** Write the necessary instructions to perform the following operations without changing any other bit settings in an 8-bit SFR.

**a)** Clear bit-0 of the SFR.
**b)** Set bit-3 of the SFR.
**c)** Toggle bit-7 of the SFR.
**d)** Clear bit-4 and toggle bit-0 of the SFR.
**e)** Set bit-0 and bit-7, and clear bit-2 of the SFR.

The microprocessor cannot access the individual bits in the SFR, but it can read/write the SFR contents and modify the accumulator using the following assembly instructions.

```
ReadSFR   A, <SFRn>;  Read from SFRn into accumulator
WriteSFR  A, <SFRn>;  Write accumulator contents into SFRn
And       A, #0xNN;   Bitwise immediate AND operation
Or        A, #0xNN;   Bitwise immediate OR operation
ExOr      A, #0xNN;   Bitwise immediate EXOR operation
```

**2.** Write the necessary **shift** and **addition** instructions that can replace the following multiplication and division operations.

Example: In order to calculate **Dout = 3 * Din:**

```
Dout = Din + (Din << 1);  // Dout = Din + 2*Din;
```

**a)** Dout = 6 * Din
**b)** Dout = 5 * Din
**c)** Dout = 10 * Din
**d)** Dout = 1.5 * Din
**e)** Dout = Din / 4
**f)** Dout = Din / 1.6    (Hint: 1/1.6 = 0.625 = 0.5 + 0.125)

**3.** What will be the contents of 16-bit Stack pointer, and the three 8-bit registers, A, B, C, after executing the following instructions?

```
PUSH    A;
PUSH    B;
PUSH    C;
POP     B;
POP     A;
```

| Initial Condition | Final value |
|---|---|
| SP = 0xFFFF | SP = |
| A = 0x35 | A = |
| B = 0x77 | B = |
| C = 0xA1 | C = |

**4.** Trace the following stack-related operations showing the hexadecimal values stored in program counter (PC), stack pointer (SP), and stack memory **after** execution of each instruction.

| Code Addr. | Instruction | Program Counter | Stack Pointer | ------------ Stack Memory ------------ | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | FFFF | FFFE | FFFD | FFFC | FFFB | FFFA | FFF9 |
| | MainProg: | ---- | ---- | -- | -- | -- | -- | -- | -- | -- |
| ---- | ---------- | 0A10 | FFFF | ?? | ?? | ?? | ?? | ?? | ?? | ?? |
| 0A10 | Load A, #0xAA; | | | | | | | | | |
| 0A12 | Call Sub1; | | | | | | | | | |
| 0A15 | ---------- | ---- | ---- | -- | -- | -- | -- | -- | -- | -- |
| ---- | ---------- | ---- | ---- | -- | -- | -- | -- | -- | -- | -- |
| ---- | ---------- | ---- | ---- | -- | -- | -- | -- | -- | -- | -- |
| | Sub1: | ---- | ---- | -- | -- | -- | -- | -- | -- | -- |
| 1B20 | Push A; | | | | | | | | | |
| 1B21 | Load A, #0xBB; | | | | | | | | | |
| 1B23 | Call Sub2; | | | | | | | | | |
| 1B26 | ---------- | ---- | ---- | -- | -- | -- | -- | -- | -- | -- |
| ---- | ---------- | ---- | ---- | -- | -- | -- | -- | -- | -- | -- |
| 1B40 | Pop A; | | | | | | | | | |
| 1B42 | Return; | | | | | | | | | |
| ---- | ---------- | ---- | ---- | -- | -- | -- | -- | -- | -- | -- |
| ---- | ---------- | ---- | ---- | -- | -- | -- | -- | -- | -- | -- |
| | Sub2: | ---- | ---- | -- | -- | -- | -- | -- | -- | -- |
| 2C30 | Push A; | | | | | | | | | |
| ---- | ---------- | ---- | ---- | -- | -- | -- | -- | -- | -- | -- |
| 2C50 | Pop A; | | | | | | | | | |
| 2C52 | Return; | | | | | | | | | |
| ---- | ---------- | ---- | ---- | -- | -- | -- | -- | -- | -- | -- |

**5.a)** What are the meanings of "***Little Endian***" and "***Big Endian***" byte orders for storing integer numbers in memory?

**b)** Show the order of bytes stored in memory according to the "Little Endian" and "Big Endian" byte ordering schemes for the following integer numbers:
**0x2255** (short integer or 2-byte integer)
**0x0A1B2C3D** (long integer or 4-byte integer)

**c)** Describe the problems that may come up when transmitting data between computers that use different byte ordering schemes.

**6.** Consider the following declarations:

```
long int     M;
typedef struct
{ long int     Count;
  char         Nsmp;
  short int    Period;
  short int    Amplitude;
} WaveRecord;
WaveRecord     WavePar;
WaveRecord     *WavePtr;
unsigned char  *BytePtr;
```

Identify if it is an address or data written to the left hand side in the following statements. Specify the number of bytes transferred as a result of each statement assuming 16-bit memory address.

| | **A or D** | **# bytes** |
|---|---|---|
| **a)** `M = WavePar.Count;` | | |
| **b)** `M = (long int)WavePar.Period;` | | |
| **c)** `M = (long int)WavePtr->Period;` | | |
| **d)** `M = (long int)(*BytePtr);` | | |
| **e)** `BytePtr = (unsigned char *)(&M);` | | |
| **f)** `BytePtr = (unsigned char *)WavePtr;` | | |
| **g)** `*BytePtr = *(unsigned char *)WavePtr;` | | |
| **h)** `*(WaveRecord *)BytePtr = WavePar;` | | |
| **i)** `*(WaveRecord *)BytePtr = *WavePtr;` | | |
| **j)** `*WavePtr = WavePar;` | | |
| **k)** `WavePtr = &WavePar;` | | |
| **l)** `WavePtr = (WaveRecord *)BytePtr;` | | |
| **m)** `WavePar = *(WaveRecord *)BytePtr;` | | |
| **n)** `*WavePtr = *(WaveRecord *)BytePtr;` | | |
| **o)** `*BytePtr = WavePar.Nsmp;` | | |
| **p)** `*BytePtr = WavePtr->Nsmp;` | | |
| **q)** `*BytePtr = (unsigned char)WavePtr->Period;` | | |
| **r)** `WavePar.Amplitude = (short int)M;` | | |
| **s)** `WavePtr->Amplitude = (short int)(*BytePtr);` | | |
| **t)** `*(long int *)BytePtr = (long int)WavePtr->Period;` | | |

**7.** Trace the code segment given below showing the target address and the result written to target with each statement. Refer to the following symbol table to find the necessary addresses:

| Symbol | Address |
|---|---|
| M | 0x1A00 |
| N | 0x1A04 |
| LongPtr | 0x1A08 |
| AnotherPtr | 0x1A0A |

```
// Declarations:
long int  M, N;          // long integer variables
long int  *LongPtr;      // pointer to long integer
long int  **AnotherPtr;  // pointer to pointer
```

| // Trace the following lines: | Target address | Result written |
|---|---|---|
| AnotherPtr = &LongPtr; | | |
| *AnotherPtr = &M; | | |
| M = 0; | | |
| *LongPtr ++; | | |
| LongPtr ++; | | |
| *LongPtr = M; | | |