3.1.11 Section Review, Questions 1, 2, 3, 4

1. Dec: -35d

Hex: 0DDh

Oct: 335o

Bin: 11011101b

1. No
2. No
3. (10 + 4) \* -(6-4) / 2 mod 3 = -2

3.2.4 Section Review, Questions 4, 5

1. The EAX register
2. INVOKE ExitProcess,0

3.3.3 Section Review, Questions 1, 2, 3, 4

1. Object file, Listing file (optional)
2. True
3. True
4. The loader

3.4.13 Section Review, Questions 1, 2, 3, 4, 5

1. value1 SWORD ?
2. value1 BYTE ?
3. value1 SBYTE ?
4. value1 QWORD ?
5. SDWORD

3.5.5 Section Review Question 3

3. ArraySize = ($ - myArray)

3.9.1 Short Answer, Questions 4, 5, 6, 25

1. ‘Assembler’ refers to the program which creates the executable, not the actual language itself.
2. The difference between big endian and little endian is how they order multiple bytes of information. Little endian refers to the placement of the least significant byte at the first memory address allocated for data. The last byte is the largest value, being placed subsequent from all previous lower values, hence the term “low to high” being applicable. Big endian is the reverse, and data is stored from high to low. The first byte is the biggest.
3. Because code is easier to read and maintain when using symbols. If a constant is present throughout the code and needs to be redefined at some point, it is much easier to change a symbol and have that reflected through all instances of the code, rather than searching and changing all instances of the constant.
4. The Equal-Sign directive

3.9.2 Algorithm Workbench, Questions 4, 7, 13

1. What this tells me about the assembler’s type checking is that it is largely non-existent. While a DWORD (32-bt unsigned) variable ranges between 0 and 4294967295, I can declare a negative value, which causes the variable to ‘wrap around’ to the highest possible values.
2. myArray DWORD 120 DUP (?)

13. myString BYTE 500 DUP ("TEST")