1. Bit\_Set

Bei dieser Funktion soll der Benutzer einen Wert mit der maximalen Größe von 255 eingeben. Dazu wird auch noch nach einer bestimmten bit-Stelle gefragt. Das bit an dieser bestimmten Stelle wird dann auf ‚1‘ gesetzt, falls es davor nicht sowieso schon auf ‚1‘ war. Diese ‚1‘ wird dann in einem eigenen Byte an die gewünschte Stelle verschoben.

Die beiden Bytes werden dann verodert. So muss dann an jeder Stelle wo vorher schon eine ‚1‘ war, auch jetzt wieder eine ‚1‘ sein + an der Stelle wo das eine bit im anderen Byte geschoben wurde.

1. Bit\_Clear

Bei dieser Funktion soll der Benutzer einen Wert mit der maximalen Größe von 255 eingeben. Dazu wird auch noch nach einer bestimmten bit-Stelle gefragt. Das bit an dieser bestimmten Stelle wird dann auf ‚0‘ gesetzt, falls es davor nicht sowieso schon auf ‚0‘ war. Diese ‚1‘ wird dann in einem eigenen Byte an die gewünschte Stelle verschoben. Mit der Umkehrfunktion werden dann alle Nullen zu einer Eins und anders herum.

Wenn man jetzt das eingegebene Byte mit dem erzeugten Byte mit der UND-Funktion behandelt,

1. Bit\_Test

Bei dieser Funktion, habe ich einen Teil von Bit\_set verwendet. Der Eingangswert wird in einer 2. Variable gespeichert und danach wird normal die Funktion Bit\_Set durchgeführt. Wenn dies fertig ist, wird der Originale Wert mit dem in der 2. Variable verglichen. Wenn es dort eine Differenz gibt, dann ist an der vom Benutzer eingegebenen Stelle kein Bit gesetzt. Wenn es keine Differenz gibt, dann ist an der Stelle ein Bit gesetzt.

1. Bit\_Print

Bei dieser Funktion habe zunächst ein Array erstellt, wo die 8 Bits gespeichert werden. Dann habe ich die vom Benutzer eingegebene Zahl immer durch 2 geteilt. Wenn bei der Division ein Rest bleibt, wird eine 1 im Array gesetzt. Wenn kein Rest bleibt, dann wird eine 0 im Array geschrieben. Die Position an der die 0 oder 1 geschrieben ist, ist abhängig in dem wievielten Durchlauf die Rechnung ist.

Wenn man also eine 13 mit 2 dividiert, bleibt eine ganzzahlige Zahl (6) über und ein Rest von 0,5. Da es der erste Durchlauf ist, wird die 1 im ersten Feld vom Array geschrieben. Nach 8 Durchlaufen, wird das gesamte Array falschherum ausgeben, damit es in der Richtig Leserichtung steht.

1. Bit\_Count

Bei dieser Funktion habe ich den Eingangswert immer durch 2 dividiert. Wenn ein Rest von der Division entsteht, dann war ein Bit gesetzt, und eine Variable wird um den Wert 1 erhöht. Dies wird 8 mal wiederholt (Anzahl der Bits in einem Byte).

1. Int\_to\_4BYTE
2. Bit\_clearlowest
3. Bit\_reverse

Bei dieser Funktion wird dieselbe Funktion von Bit\_Print verwendet. Am Ende wird jedoch das Array andersherum ausgegeben.