The Format of the Guild Wars 2 Archive File

Jon Dahm

May 2, 2014

Contents

Notes

Libraries

To my knowledge, there are two major C++ libraries for working with the Archive file. Github user Ahom has created a library for working with File Records and extracting images that you can find here. Github user Rhoot has created a library that will extract information from a large number of files within the Archive. You can find his work here. Most of the information in this document has come from these projects.

Endianness and Numbers

All numbers I list in this document are decimal (base 10) unless specified otherwise. Hexadecimal numbers are followed by a subscript x ($\texttt{1A}_x$). Sometimes a single byte will be listed as a character rather than a number. In these cases the value of that byte is the ASCII code of the character listed.

When I list values, sometimes I will list them as full numbers (like $\texttt{40CB}_x$) and sometimes I will list them as individual bytes (like [$\texttt{CB}_x$,$\texttt{40}_x$]). When I list the individual bytes, they are listed in the order they appear in the Archive. When I list them as full numbers, that is their actual value.

The Archive is arranged in little-endian format. This means that if you see a 16-bit value [$\texttt{CB}_x$,$\texttt{40}_x$], its actual value is $\texttt{40CB}_x$.

Disclaimer

I do not condone use of this document to modify the archive for any reason. Modifying the archive is a direct violation of the Terms of Service you agreed to follow when you bought the game.

Chapter 1

File Records

This chapter will introduce you to the main portions of the Archive, from which you can find every file represented within. After reading this chapter, you should be able to produce a list of all files within the archive. Additionally, if a file within the archive is referenced by its ID, you should be able to retrieve it.

1.1 The Archive Header

The Archive begins with a 40-byte header which describes some of the properties of the Archive and points to the Main File Table. The format of this header can be found in Table 1.1.

1.2 The Main File Table

The Main File Table (MFT) is a list of all of the files in the Archive. Its structure begins with a 24-byte-long header, whose format is given in Table 1.2. The header is followed by a number of 24-byte entries that make up the table. Each entry refers to a single file and some associated metadata. The entries are not listed in any particular order. See Table 1.3 for details.

The first fifteen entries in the MFT are reserved for special files in the Archive. They are documented below:

1	Archive Header
2	File ID Table (See section 1.3)
3	MFT (self reference)
4–15	Blank Entries

Table 1.1: the Archive header

Byte	Size	Value	Description
0	1	Version	Version of the Archive. Seems to always be `97`$_x$
1	3	Identifier	Identifies this file as the Archive file, as opposed to a MS Word file. Always [`45`$_x$,`4E`$_x$,`1A`$_x$].
4	4	Header Size	Size of this header. Always 40.
8	4	(unknown)	Always `CABA0001`$_x$.
12	4	Chunk Size	Size of each chunk in the file. Always 512.
16	4	(unknown)[1]	Always `8ED0A720`$_x$.
20	4	(unknown)	Always `00040002`$_x$.
24	8	MFT Offset	The offset from the beginning of the Archive to the Main File Table.
32	4	MFT Size	Size of the Main File Table in bytes.
36	4	(unknown)	Always 0.

Table 1.2: the MFT header

Byte	Size	Value	Description
0	4	Identifier	Identifies the start of the MFT. Always [`‘M’`,`‘f’`,`‘t’`,`1A`$_x$].
4	8	(unknown)	
12	4	Length	Number of entries in the table plus one.
16	8	(unknown)	Always 0.

Table 1.3: an MFT entry

Byte	Size	Value	Description
0	8	Offset	Offset from the beginning of the Archive to the start of the file.
8	4	Archived Size	Size in bytes of the file within the archive.
12	2	Compression	Type of compression the file is under. See below.
14	2	Flags	Other flags. See below.
16	4	(unknown)	Always 0.
20	4	(unknown)	Always `4867 4BC7`$_x$.

Valid values for Compression:

0	Uncompressed
8	Huffman Compression

Valid values for Flags:

1	In Use
2	(unknown)

1.3 The File ID Table

The File ID Table gives each file in the MFT an ID. Each entry in the table has the format listed in Table 1.4. The entries are not listed in any particular order.

For the most part, each entry has only one ID. However, many have more than one ID each. As of the time of this writing, approximately a third of the files in the Archive have two IDs, and none have more. *More research must be done into why some entries have multiple IDs.*

Additionally, some entries may contain nil values for either field. I haven't found a significant number of these, but they exist. I have only found entries where both fields are nil, and none where only one was nil. My recommendation is to discard any entries with nil fields.

Table 1.4: a File ID Table entry

Byte	Size	Value	Description
0	4	File ID	
4	4	MFT Entry Index	Indices start at 1

Chapter 2

Files and Compression

This chapter will introduce you to how to identify files and decompress files that have been compressed. Additionally, I'll discuss the compression used on many of the texture files in the Archive. After reading this chapter, you should be able to, given the address of the start of a file, provide its raw data, whether the file was compressed or not.

2.1 File Types

Every file starts with an 8-byte header identifying the type of file and how large it is. The first 4 bytes of the header are the file's type identifier, typically represented by four character codes (4CC). The second 4 bytes tell you how long the uncompressed file is, if the file is compressed.

In the latest version of the Archive at the time of this writing, 99% of the files were compressed. All of these files are represented in the general file header by one 4CC. To find the actual 4CC defining the file type, you have to decompress the file, which we will go over in the next section.

The following table describes all 4CCs that appear in the general file header, listed in decreasing order of frequency:

[08_x,00_x,01_x,80_x]	Compressed File
[‘A’,‘T’,‘E’,‘X’]	General Use Texture
[‘A’,‘T’,‘E’,‘U’]	UI Texture
[‘K’,‘B’,‘2’,‘f’]	(unknown)
[‘K’,‘B’,‘2’,‘g’]	(unknown)
[$7C_x$,$1A_x$,‘I’,‘z’]	(unknown)
[97_x,‘A’,‘N’,$1A_x$]	(unknown)

2.2 File Compression

TODO: Add illustrations.

Compression is a difficult subject to describe tersely. The compression used in the Archive is very similar to that produced by the DEFLATE algorithm. If you are familiar with the DEFLATE algorithm, you may notice them. To keep things (relatively) short, however, I won't describe every difference between the two.

Data is compressed using Huffman codes and back-copying. The former is a method of taking a set of data and compressing it as small as possible, and the latter is a method of further compressing the data by replacing reoccuring data with a refrence to the last time it occured. I won't go into the details of how all this works, so if you aren't familiar with either of these, read this fantastic article on zlib which does a wonderful job explaining the concepts. Be sure to understand these concepts well before continuing in this section, or you will be lost. If this is well beyond you, and you don't care particularly about implementing a decompression algorithm yourself, just use Ahom's decompression algorithm and skip the rest of this chapter.

To begin, it is incredibly important to note the order in which bits are read. Strangely enough, bytes aren't read from beginning to end — instead, they are split into little-endian 32-bit values, and read from highest bit to lowest. For illustration, see Figure BLAH. When I refer to ordering of elements in this section, I assume that bits are being read in this order.

Next, every 64KiB, 4 bytes are skipped. As of the writing of this document, I am unaware the purpose of this. I would guess that those 4 bytes are a check on the previous data in order to help detect corruption.

The compressed data starts with a single byte that represents an adjustment to any back-copy sizes encountered in the data. This should be saved for later use. The rest of the data is split into blocks.

Each block begins with two Huffman Trees describing the Huffman codes for the literal/copy-length alphabet and the copy-offset alphabet. These are followed by 4 bits which represent the number of codes from the first alphabet to expect in this section. The rest of the block is the Huffman codes representing the information compressed in this section.

In the next subsection, I'll describe how you generate Huffman codes from the Huffman Trees presented in each block.

2.2.1 Huffman Trees

Each tree can represent a variable number of values. The first 16 bits are an unsigned value representing how many values this tree is giving Huffman codes to. This is followed by a number of entries describing sometimes several codes at once. These entries are compressed using predefined codes found in Appendix A.

Each entry represents at least one value and its code. The first entry refers to the highest values the tree represents, with each successive entry referring to a lower value. The highest three bits state how many more values this entry applies to. The lowest 5 bits state how long the Huffman codes are for these values. If the length is 0, those values aren't actually represented in the tree, and you can skip over them.

As it turns out, in order to generate a valid Huffman code for a value, all you need to know is how long the Huffman code for it is. The following algorithm derives the Huffman codes for all values whose lengths you know are non-zero.

Sort all of your value+code-length pairs first in ascending order of length, then in ascending order of value. Assign the first value a code of all 1's. For each successive value that uses the same length code, decrement the code by one. When you reach a value that uses more code bits, multiply the last code by 2 and then subtract one. Continue this process until you have assigned each value a Huffman code.

The Tree representing the literal/copy-length alphabet cannot have more than 285 values in it. The Tree representing the copy-offset alphabet cannot have more than 34 values.

2.2.2 Translating Huffman Codes to Data

In each block, after the Huffman Trees, there are 4 bits describing how many codes from the literal/copy-length alphabet there in the block. The number is determined by adding one to the value of the 4 bits and then multiplying by $\texttt{1000}_x$. If the end of the file has been reached, then this number may be greater than the actual number of codes, so you'll have to watch to make sure you don't overshoot the end of the stream.

There are two modes to translating the codes to data — literal, where each code matches one byte, and copy, where extra data follows the code describing how many bytes to copy from where in the output stream generated so far. If the value of the code translated is less than $\texttt{100}_x$, then the output is a byte with that value. If the value is greater than $\texttt{100}_x$, then you have to copy previous output back into the stream.

Following a copy code are additional bits that add to the length represented by the code itself. Table 2.1 provides the base lengths for each value and how many additional bits you must read and add to the base length.

Table 2.1: Copy Length Table

Code	Base	Additional Bits		Code	Base	Additional Bits
100_x	1	0		110_x	33	3
101_x	2	0		111_x	41	3
102_x	3	0		112_x	49	3
103_x	4	0		113_x	57	3
104_x	5	0		114_x	65	4
105_x	6	0		115_x	81	4
106_x	7	0		116_x	97	4
107_x	8	0		117_x	113	4
108_x	9	1		118_x	129	5
109_x	11	1		119_x	161	5
$10A_x$	13	1		$11A_x$	193	5
$10B_x$	15	1		$11B_x$	225	5
$10C_x$	17	2		$11C_x$	256	0
$10D_x$	21	2				
$10E_x$	25	2				
$10F_x$	29	2				

After that is a code from the copy-offset alphabet. This also has additional bits following it to add to it. Table 2.2 details the base offsets and the number of additional bits for each value.

To calculate the total length of the copy, add the base length, the value of the additional length bits, and the copy size adjustment value from the beginning of the file. To calculate the total offset of the copy, add the base offset and the additional offset bits. It may be helpful to note that the sliding window on this algorithm appears to be 128KiB.

Table 2.2: Copy Offset Table

Code	**Base**	**Additional Bits**		**Code**	**Base**	**Additional Bits**
$\texttt{0}_x$	$\texttt{1}_x$	0		$\texttt{12}_x$	$\texttt{201}_x$	8
$\texttt{1}_x$	$\texttt{2}_x$	0		$\texttt{13}_x$	$\texttt{301}_x$	8
$\texttt{2}_x$	$\texttt{3}_x$	0		$\texttt{14}_x$	$\texttt{401}_x$	9
$\texttt{3}_x$	$\texttt{4}_x$	0		$\texttt{15}_x$	$\texttt{601}_x$	9
$\texttt{4}_x$	$\texttt{5}_x$	1		$\texttt{16}_x$	$\texttt{801}_x$	10
$\texttt{5}_x$	$\texttt{7}_x$	1		$\texttt{17}_x$	$\texttt{C01}_x$	10
$\texttt{6}_x$	$\texttt{9}_x$	2		$\texttt{18}_x$	$\texttt{1001}_x$	11
$\texttt{7}_x$	$\texttt{D}_x$	2		$\texttt{19}_x$	$\texttt{1801}_x$	11
$\texttt{8}_x$	$\texttt{11}_x$	3		$\texttt{1A}_x$	$\texttt{2001}_x$	12
$\texttt{9}_x$	$\texttt{19}_x$	3		$\texttt{1B}_x$	$\texttt{3001}_x$	12
$\texttt{A}_x$	$\texttt{21}_x$	4		$\texttt{1C}_x$	$\texttt{4001}_x$	13
$\texttt{B}_x$	$\texttt{31}_x$	4		$\texttt{1D}_x$	$\texttt{6001}_x$	13
$\texttt{C}_x$	$\texttt{41}_x$	5		$\texttt{1E}_x$	$\texttt{8001}_x$	14
$\texttt{D}_x$	$\texttt{61}_x$	5		$\texttt{1F}_x$	$\texttt{C001}_x$	14
$\texttt{E}_x$	$\texttt{81}_x$	6		$\texttt{20}_x$	$\texttt{10001}_x$	15
$\texttt{F}_x$	$\texttt{C1}_x$	6		$\texttt{21}_x$	$\texttt{18001}_x$	15
$\texttt{10}_x$	$\texttt{101}_x$	7				
$\texttt{11}_x$	$\texttt{181}_x$	7				

Chapter 3

Uncompressed Files

TODO: Fill out this chapter.

Chapter 4

Compressed Files

TODO: Fill out this chapter.

Chapter 5

Pack Files

This chapter will introduce you to one file type – The Pack File. This file type is used to store a large portion of the game data, and has many subtypes for data such as animations, models, maps, textures, and audio. You'll learn how to navigate through the pack file to grab individual portions of the data it holds, as well as what data each subtype contains.

5.1 Headers

Each Pack File begins with a 12-byte header identifying the type of data contained. This header includes the 4-byte Character Code identifier labeling the file as a Pack File. The format of this header can be found in Table 5.1.

Table 5.1: the Pack File header

Byte	**Size**	**Value**	**Description**
0	2	Identifier	Always ['P','F']
2	4	Unknown	Always 1_x
6	2	Header Size	Always 12
8	4	Type	4 Character Codes defining the data held in this Pack File.

Each Pack File is split into blocks of data, called Chunks. The first chunk follows immediately after the Pack File header. Each one contains a pointer to the next one.

Chunks are identified by 4 character codes, which determines the format of the chunk and the data stored in it. While different Pack Files may share chunks with the same identifier, these chunks may not be the same. For example, the Content Manifest and Map Metadata Pack Files both have a chunk labeled 'Main,' but their formats are different.

Chunks begin with a 16-byte header describing where the next chunk is in the file and the type of data stored in this chunk. The format can be found in Table 5.2.

Table 5.2: the Chunk header

Byte	Size	Value	Description
0	4	Identifier	Type of Chunk
4	4	Next Chunk	Number of bytes after the end of this value the next chunk appears.
8	2	Unknown	
10	2	Header Size	Always 12
12	4	Unknown Pointer	A pointer to an unknown data structure.

In Table 5.3 you will find all the known Pack File types, as well as the chunks each one contains. The following sections will go into each chunk and its format.

5.2 Animation Sequence Pack File

TODO: Finish this section

5.3 Audio Pack File

TODO: Finish this section

5.4 Bank Pack File

TODO: Finish this section

5.5 Bank Index Pack File

TODO: Finish this section

Table 5.3: Pack File Types sorted by FourCC

FourCC	Value	Name	Included Chunks
[‘A’,‘B’,‘I’,‘X’]	58494241$_x$	Bank Index	[‘B’,‘I’,‘D’,‘X’]
[‘A’,‘B’,‘N’,‘K’]	4b4e4241$_x$	Bank	[‘B’,‘K’,‘C’,‘K’]
[‘A’,‘M’,‘A’,‘T’]	54414d41$_x$	Material	[‘G’,‘R’,‘M’,‘T’], [‘D’,‘X’,‘9’,‘S’]
[‘A’,‘M’,‘S’,‘P’]	50534d41$_x$	Script	[‘A’,‘M’,‘S’,‘P’]
[‘a’,‘n’,‘i’,‘c’]	63696E61$_x$	Animation Sequence	[‘s’,‘e’,‘q’,‘n’]
[‘A’,‘S’,‘N’,‘D’]	444e5341$_x$	Audio	[‘A’,‘S’,‘N’,‘D’]
[‘C’,‘I’,‘N’,‘P’]	504e4943$_x$	Scene	[‘C’,‘S’,‘C’,‘N’]
[‘c’,‘m’,‘a’,‘C’]	43616d63$_x$	Collide Model Manifest	[‘m’,‘a’,‘i’,‘n’]
[‘c’,‘m’,‘p’,‘c’]	63706d63$_x$	Composite	[‘c’,‘o’,‘m’,‘p’]
[‘c’,‘n’,‘t’,‘c’]	63746e63$_x$	Content Manifest	[‘M’,‘a’,‘i’,‘n’]
[‘e’,‘m’,‘o’,‘c’]	636f6d65$_x$	Emote Animation	[‘a’,‘n’,‘i’,‘m’]
[‘e’,‘u’,‘l’,‘a’]	616c7565$_x$	EULA	[‘e’,‘u’,‘l’,‘a’]
[‘h’,‘v’,‘k’,‘C’]	436b7668$_x$	Havok	[‘h’,‘a’,‘v’,‘k’]
[‘m’,‘a’,‘p’,‘c’]	6370616d$_x$	Map	[‘a’,‘u’,‘d’,‘i’], [‘m’,‘s’,‘n’,00$_x$], [‘p’,‘a’,‘r’,‘m’], [‘s’,‘h’,‘o’,‘r’], [‘s’,‘u’,‘r’,‘f’], [‘t’,‘r’,‘n’,‘i’], [‘a’,‘r’,‘e’,‘a’], [‘h’,‘a’,‘v’,‘k’], [‘c’,‘u’,‘b’,‘e’], [‘d’,‘c’,‘a’,‘l’], [‘e’,‘n’,‘v’,00$_x$], [‘l’,‘g’,‘h’,‘t’], [‘p’,‘r’,‘p’,‘2’], [‘r’,‘i’,‘v’,‘e’], [‘s’,‘h’,‘e’,‘x’], [‘t’,‘r’,‘n’,00$_x$], [‘z’,‘o’,‘n’,‘2’]
[‘m’,‘M’,‘e’,‘t’]	74654d6d$_x$	Map Metadata	[‘M’,‘a’,‘i’,‘n’]
[‘M’,‘O’,‘D’,‘L’]	4c444f4d$_x$	Model	[‘A’,‘N’,‘I’,‘M’], [‘M’,‘O’,‘D’,‘L’], [‘G’,‘E’,‘O’,‘M’], [‘P’,‘R’,‘P’,‘S’], [‘R’,‘O’,‘O’,‘T’], [‘S’,‘K’,‘E’,‘L’]
[‘m’,‘p’,‘s’,‘d’]	6473706d$_x$	Map Shadow	[‘s’,‘h’,‘a’,‘d’]
[‘P’,‘I’,‘M’,‘G’]	474d4950$_x$	Paged Image	[‘P’,‘G’,‘T’,‘B’]
[‘p’,‘r’,‘l’,‘t’]	746c7270$_x$	Portal Manifest	[‘m’,‘f’,‘s’,‘t’]
[‘t’,‘x’,‘t’,‘m’]	6d747874$_x$	Text Manifest	[‘t’,‘x’,‘t’,‘m’]
[‘t’,‘x’,‘t’,‘V’]	56747874$_x$	Text Variant	[‘v’,‘a’,‘r’,‘i’]
[‘t’,‘x’,‘t’,‘v’]	76747874$_x$	Text Voice	[‘t’,‘x’,‘t’,‘v’]

5.6 Collide Model Manifest Pack File

TODO: Finish this section

5.7 Composite Pack File

TODO: Finish this section

5.8 Content Manifest Pack File

TODO: Finish this section

5.9 Emote Animation Pack File

TODO: Finish this section

5.10 EULA Pack File

TODO: Finish this section

5.11 Havok Pack File

TODO: Finish this section

5.12 Map Pack File

TODO: Finish this section

5.12.1 Unknown Chunk

TODO: Finish this section

5.12.2 Unknown Chunk

TODO: Finish this section

5.12.3 Unknown Chunk

TODO: Finish this section

5.12.4 Unknown Chunk

TODO: Finish this section

5.12.5 Unknown Chunk

TODO: Finish this section

5.12.6 Unknown Chunk

TODO: Finish this section

5.12.7 Unknown Chunk

TODO: Finish this section

5.12.8 Unknown Chunk

TODO: Finish this section

5.12.9 Unknown Chunk

TODO: Finish this section

5.12.10 Unknown Chunk

TODO: Finish this section

5.12.11 Unknown Chunk

TODO: Finish this section

5.12.12 Unknown Chunk

TODO: Finish this section

5.12.13 Unknown Chunk

TODO: Finish this section

5.12.14 Unknown Chunk

TODO: Finish this section

5.12.15 Unknown Chunk

TODO: Finish this section

5.12.16 Unknown Chunk

TODO: Finish this section

5.12.17 Unknown Chunk

TODO: Finish this section

5.13 Map Metadata Pack File

TODO: Finish this section

5.14 Map Shadow Pack File

TODO: Finish this section

5.15 Material Pack File

TODO: Finish this section

5.15.1 Unknown Chunk

TODO: Finish this section

5.15.2 Unknown Chunk

TODO: Finish this section

5.16 Model Pack File

TODO: Finish this section

5.16.1 Unknown Chunk

TODO: Finish this section

5.16.2 Unknown Chunk

TODO: Finish this section

5.16.3 Unknown Chunk

TODO: Finish this section

5.16.4 Unknown Chunk

TODO: Finish this section

5.16.5 Unknown Chunk

TODO: Finish this section

5.16.6 Unknown Chunk

TODO: Finish this section

5.17 Paged Image Pack File

TODO: Finish this section

5.18 Portal Manifest Pack File

TODO: Finish this section

5.19 Scene Pack File

TODO: Finish this section

5.20 Script Pack File

TODO: Finish this section

5.21 Text Manifest Pack File

TODO: Finish this section

5.22 Text Variant Pack File

TODO: Finish this section

5.23 Text Voice Pack File

TODO: Finish this section

Appendix A

Static Huffman Trees

The static tree used when defining trees for decompressing files:

Value	Huffman Code	Number of Bits
$\texttt{08}_x$	$\texttt{111}_b$	3
$\texttt{09}_x$	$\texttt{110}_b$	3
$\texttt{0A}_x$	$\texttt{101}_b$	3
$\texttt{00}_x$	$\texttt{1001}_b$	4
$\texttt{07}_x$	$\texttt{1000}_b$	4
$\texttt{0B}_x$	$\texttt{0111}_b$	4
$\texttt{0C}_x$	$\texttt{0110}_b$	4
$\texttt{06}_x$	$\texttt{01011}_b$	5
$\texttt{29}_x$	$\texttt{01010}_b$	5
$\texttt{2A}_x$	$\texttt{01001}_b$	5
$\texttt{E0}_x$	$\texttt{01000}_b$	5
$\texttt{04}_x$	$\texttt{001111}_b$	6
$\texttt{05}_x$	$\texttt{001110}_b$	6
$\texttt{20}_x$	$\texttt{001101}_b$	6
$\texttt{28}_x$	$\texttt{001100}_b$	6
$\texttt{2B}_x$	$\texttt{001011}_b$	6
$\texttt{2C}_x$	$\texttt{001010}_b$	6
$\texttt{40}_x$	$\texttt{001001}_b$	6
$\texttt{4A}_x$	$\texttt{001000}_b$	6
$\texttt{03}_x$	$\texttt{0001111}_b$	7
$\texttt{0D}_x$	$\texttt{0001110}_b$	7
$\texttt{25}_x$	$\texttt{0001101}_b$	7
$\texttt{26}_x$	$\texttt{0001100}_b$	7
$\texttt{27}_x$	$\texttt{0001011}_b$	7

48_x	0001010_b	7
49_x	0001001_b	7
24_x	00010001_b	8
47_x	00010000_b	8
$4B_x$	00001111_b	8
$4C_x$	00001110_b	8
69_x	00001101_b	8
$6A_x$	00001100_b	8
23_x	000010111_b	9
46_x	000010110_b	9
60_x	000010101_b	9
63_x	000010100_b	9
67_x	000010011_b	9
68_x	000010010_b	9
88_x	000010001_b	9
89_x	000010000_b	9
$A0_x$	000001111_b	9
$E8_x$	000001110_b	9
01_x	0000011011_b	10
02_x	0000011010_b	10
$2D_x$	0000011001_b	10
43_x	0000011000_b	10
44_x	0000010111_b	10
45_x	0000010110_b	10
65_x	0000010101_b	10
66_x	0000010100_b	10
80_x	0000010011_b	10
87_x	0000010010_b	10
$8A_x$	0000010001_b	10
$A8_x$	0000010000_b	10
$A9_x$	0000001111_b	10
$C0_x$	0000001110_b	10
$C9_x$	0000001101_b	10
$E9_x$	0000001100_b	10
$0E_x$	00000010111_b	11
$4D_x$	00000010110_b	11
64_x	00000010101_b	11
$6B_x$	00000010100_b	11
$6C_x$	00000010011_b	11

84$_x$	00000010010$_b$	11
85$_x$	00000010001$_b$	11
8B$_x$	00000010000$_b$	11
A4$_x$	00000001111$_b$	11
A5$_x$	00000001110$_b$	11
AA$_x$	00000001101$_b$	11
C8$_x$	00000001100$_b$	11
E5$_x$	00000001011$_b$	11
83$_x$	000000010101$_b$	12
86$_x$	000000010100$_b$	12
A6$_x$	000000010011$_b$	12
A7$_x$	000000010010$_b$	12
C7$_x$	000000010001$_b$	12
CA$_x$	000000010000$_b$	12
E7$_x$	000000001111$_b$	12
22$_x$	0000000011101$_b$	13
2E$_x$	0000000011100$_b$	13
8C$_x$	0000000011011$_b$	13
C4$_x$	0000000011010$_b$	13
E4$_x$	0000000011001$_b$	13
E6$_x$	0000000011000$_b$	13
4E$_x$	00000000101111$_b$	14
6D$_x$	00000000101110$_b$	14
C6$_x$	00000000101101$_b$	14
EC$_x$	00000000101100$_b$	14
0F$_x$	000000001010111$_b$	15
10$_x$	000000001010110$_b$	15
11$_x$	000000001010101$_b$	15
8D$_x$	000000001010100$_b$	15
AB$_x$	000000001010011$_b$	15
AC$_x$	000000001010010$_b$	15
CC$_x$	000000001010001$_b$	15
EA$_x$	000000001010000$_b$	15
12$_x$	0000000010011111$_b$	16
13$_x$	0000000010011110$_b$	16
14$_x$	0000000010011101$_b$	16
15$_x$	0000000010011100$_b$	16
16$_x$	0000000010011011$_b$	16
17$_x$	0000000010011010$_b$	16

$\texttt{18}_x$	$\texttt{0000000010011001}_b$	16
$\texttt{19}_x$	$\texttt{0000000010011000}_b$	16
$\texttt{1A}_x$	$\texttt{0000000010010111}_b$	16
$\texttt{1B}_x$	$\texttt{0000000010010110}_b$	16
$\texttt{1C}_x$	$\texttt{0000000010010101}_b$	16
$\texttt{1D}_x$	$\texttt{0000000010010100}_b$	16
$\texttt{1E}_x$	$\texttt{0000000010010011}_b$	16
$\texttt{1F}_x$	$\texttt{0000000010010010}_b$	16
$\texttt{21}_x$	$\texttt{0000000010010001}_b$	16
$\texttt{2F}_x$	$\texttt{0000000010010000}_b$	16
$\texttt{30}_x$	$\texttt{0000000010001111}_b$	16
$\texttt{31}_x$	$\texttt{0000000010001110}_b$	16
$\texttt{32}_x$	$\texttt{0000000010001101}_b$	16
$\texttt{33}_x$	$\texttt{0000000010001100}_b$	16
$\texttt{34}_x$	$\texttt{0000000010001011}_b$	16
$\texttt{35}_x$	$\texttt{0000000010001010}_b$	16
$\texttt{36}_x$	$\texttt{0000000010001001}_b$	16
$\texttt{37}_x$	$\texttt{0000000010001000}_b$	16
$\texttt{38}_x$	$\texttt{0000000010000111}_b$	16
$\texttt{39}_x$	$\texttt{0000000010000110}_b$	16
$\texttt{3A}_x$	$\texttt{0000000010000101}_b$	16
$\texttt{3B}_x$	$\texttt{0000000010000100}_b$	16
$\texttt{3C}_x$	$\texttt{0000000010000011}_b$	16
$\texttt{3D}_x$	$\texttt{0000000010000010}_b$	16
$\texttt{3E}_x$	$\texttt{0000000010000001}_b$	16
$\texttt{3F}_x$	$\texttt{0000000010000000}_b$	16
$\texttt{41}_x$	$\texttt{0000000001111111}_b$	16
$\texttt{42}_x$	$\texttt{0000000001111110}_b$	16
$\texttt{4F}_x$	$\texttt{0000000001111101}_b$	16
$\texttt{50}_x$	$\texttt{0000000001111100}_b$	16
$\texttt{51}_x$	$\texttt{0000000001111011}_b$	16
$\texttt{52}_x$	$\texttt{0000000001111010}_b$	16
$\texttt{53}_x$	$\texttt{0000000001111001}_b$	16
$\texttt{54}_x$	$\texttt{0000000001111000}_b$	16
$\texttt{55}_x$	$\texttt{0000000001110111}_b$	16
$\texttt{56}_x$	$\texttt{0000000001110110}_b$	16
$\texttt{57}_x$	$\texttt{0000000001110101}_b$	16
$\texttt{58}_x$	$\texttt{0000000001110100}_b$	16
$\texttt{59}_x$	$\texttt{0000000001110011}_b$	16
$\texttt{5A}_x$	$\texttt{0000000001110010}_b$	16

$\texttt{5B}_x$	$\texttt{0000000001110001}_b$	16	
$\texttt{5C}_x$	$\texttt{0000000001110000}_b$	16	
$\texttt{5D}_x$	$\texttt{0000000001101111}_b$	16	
$\texttt{5E}_x$	$\texttt{0000000001101110}_b$	16	
$\texttt{5F}_x$	$\texttt{0000000001101101}_b$	16	
$\texttt{61}_x$	$\texttt{0000000001101100}_b$	16	
$\texttt{62}_x$	$\texttt{0000000001101011}_b$	16	
$\texttt{6E}_x$	$\texttt{0000000001101010}_b$	16	
$\texttt{6F}_x$	$\texttt{0000000001101001}_b$	16	
$\texttt{70}_x$	$\texttt{0000000001101000}_b$	16	
$\texttt{71}_x$	$\texttt{0000000001100111}_b$	16	
$\texttt{72}_x$	$\texttt{0000000001100110}_b$	16	
$\texttt{73}_x$	$\texttt{0000000001100101}_b$	16	
$\texttt{74}_x$	$\texttt{0000000001100100}_b$	16	
$\texttt{75}_x$	$\texttt{0000000001100011}_b$	16	
$\texttt{76}_x$	$\texttt{0000000001100010}_b$	16	
$\texttt{77}_x$	$\texttt{0000000001100001}_b$	16	
$\texttt{78}_x$	$\texttt{0000000001100000}_b$	16	
$\texttt{79}_x$	$\texttt{0000000001011111}_b$	16	
$\texttt{7A}_x$	$\texttt{0000000001011110}_b$	16	
$\texttt{7B}_x$	$\texttt{0000000001011101}_b$	16	
$\texttt{7C}_x$	$\texttt{0000000001011100}_b$	16	
$\texttt{7D}_x$	$\texttt{0000000001011011}_b$	16	
$\texttt{7E}_x$	$\texttt{0000000001011010}_b$	16	
$\texttt{7F}_x$	$\texttt{0000000001011001}_b$	16	
$\texttt{81}_x$	$\texttt{0000000001011000}_b$	16	
$\texttt{82}_x$	$\texttt{0000000001010111}_b$	16	
$\texttt{8E}_x$	$\texttt{0000000001010110}_b$	16	
$\texttt{8F}_x$	$\texttt{0000000001010101}_b$	16	
$\texttt{90}_x$	$\texttt{0000000001010100}_b$	16	
$\texttt{91}_x$	$\texttt{0000000001010011}_b$	16	
$\texttt{92}_x$	$\texttt{0000000001010010}_b$	16	
$\texttt{93}_x$	$\texttt{0000000001010001}_b$	16	
$\texttt{94}_x$	$\texttt{0000000001010000}_b$	16	
$\texttt{95}_x$	$\texttt{0000000001001111}_b$	16	
$\texttt{96}_x$	$\texttt{0000000001001110}_b$	16	
$\texttt{97}_x$	$\texttt{0000000001001101}_b$	16	
$\texttt{98}_x$	$\texttt{0000000001001100}_b$	16	
$\texttt{99}_x$	$\texttt{0000000001001011}_b$	16	
$\texttt{9A}_x$	$\texttt{0000000001001010}_b$	16	

$\texttt{9B}_x$	$\texttt{0000000001001001}_b$	16
$\texttt{9C}_x$	$\texttt{0000000001001000}_b$	16
$\texttt{9D}_x$	$\texttt{0000000001000111}_b$	16
$\texttt{9E}_x$	$\texttt{0000000001000110}_b$	16
$\texttt{9F}_x$	$\texttt{0000000001000101}_b$	16
$\texttt{A1}_x$	$\texttt{0000000001000100}_b$	16
$\texttt{A2}_x$	$\texttt{0000000001000011}_b$	16
$\texttt{A3}_x$	$\texttt{0000000001000010}_b$	16
$\texttt{AD}_x$	$\texttt{0000000001000001}_b$	16
$\texttt{AE}_x$	$\texttt{0000000001000000}_b$	16
$\texttt{AF}_x$	$\texttt{0000000000111111}_b$	16
$\texttt{B0}_x$	$\texttt{0000000000111110}_b$	16
$\texttt{B1}_x$	$\texttt{0000000000111101}_b$	16
$\texttt{B2}_x$	$\texttt{0000000000111100}_b$	16
$\texttt{B3}_x$	$\texttt{0000000000111011}_b$	16
$\texttt{B4}_x$	$\texttt{0000000000111010}_b$	16
$\texttt{B5}_x$	$\texttt{0000000000111001}_b$	16
$\texttt{B6}_x$	$\texttt{0000000000111000}_b$	16
$\texttt{B7}_x$	$\texttt{0000000000110111}_b$	16
$\texttt{B8}_x$	$\texttt{0000000000110110}_b$	16
$\texttt{B9}_x$	$\texttt{0000000000110101}_b$	16
$\texttt{BA}_x$	$\texttt{0000000000110100}_b$	16
$\texttt{BB}_x$	$\texttt{0000000000110011}_b$	16
$\texttt{BC}_x$	$\texttt{0000000000110010}_b$	16
$\texttt{BD}_x$	$\texttt{0000000000110001}_b$	16
$\texttt{BE}_x$	$\texttt{0000000000110000}_b$	16
$\texttt{BF}_x$	$\texttt{0000000000101111}_b$	16
$\texttt{C1}_x$	$\texttt{0000000000101110}_b$	16
$\texttt{C2}_x$	$\texttt{0000000000101101}_b$	16
$\texttt{C3}_x$	$\texttt{0000000000101100}_b$	16
$\texttt{C5}_x$	$\texttt{0000000000101011}_b$	16
$\texttt{CB}_x$	$\texttt{0000000000101010}_b$	16
$\texttt{CD}_x$	$\texttt{0000000000101001}_b$	16
$\texttt{CE}_x$	$\texttt{0000000000101000}_b$	16
$\texttt{CF}_x$	$\texttt{0000000000100111}_b$	16
$\texttt{D0}_x$	$\texttt{0000000000100110}_b$	16
$\texttt{D1}_x$	$\texttt{0000000000100101}_b$	16
$\texttt{D2}_x$	$\texttt{0000000000100100}_b$	16
$\texttt{D3}_x$	$\texttt{0000000000100011}_b$	16
$\texttt{D4}_x$	$\texttt{0000000000100010}_b$	16

$\mathtt{D5}_x$	$\mathtt{0000000000100001}_b$	16
$\mathtt{D6}_x$	$\mathtt{0000000000100000}_b$	16
$\mathtt{D7}_x$	$\mathtt{0000000000011111}_b$	16
$\mathtt{D8}_x$	$\mathtt{0000000000011110}_b$	16
$\mathtt{D9}_x$	$\mathtt{0000000000011101}_b$	16
$\mathtt{DA}_x$	$\mathtt{0000000000011100}_b$	16
$\mathtt{DB}_x$	$\mathtt{0000000000011011}_b$	16
$\mathtt{DC}_x$	$\mathtt{0000000000011010}_b$	16
$\mathtt{DD}_x$	$\mathtt{0000000000011001}_b$	16
$\mathtt{DE}_x$	$\mathtt{0000000000011000}_b$	16
$\mathtt{DF}_x$	$\mathtt{0000000000010111}_b$	16
$\mathtt{E1}_x$	$\mathtt{0000000000010110}_b$	16
$\mathtt{E2}_x$	$\mathtt{0000000000010101}_b$	16
$\mathtt{E3}_x$	$\mathtt{0000000000010100}_b$	16
$\mathtt{EB}_x$	$\mathtt{0000000000010011}_b$	16
$\mathtt{ED}_x$	$\mathtt{0000000000010010}_b$	16
$\mathtt{EE}_x$	$\mathtt{0000000000010001}_b$	16
$\mathtt{EF}_x$	$\mathtt{0000000000010000}_b$	16
$\mathtt{F0}_x$	$\mathtt{0000000000001111}_b$	16
$\mathtt{F1}_x$	$\mathtt{0000000000001110}_b$	16
$\mathtt{F2}_x$	$\mathtt{0000000000001101}_b$	16
$\mathtt{F3}_x$	$\mathtt{0000000000001100}_b$	16
$\mathtt{F4}_x$	$\mathtt{0000000000001011}_b$	16
$\mathtt{F5}_x$	$\mathtt{0000000000001010}_b$	16
$\mathtt{F6}_x$	$\mathtt{0000000000001001}_b$	16
$\mathtt{F7}_x$	$\mathtt{0000000000001000}_b$	16
$\mathtt{F8}_x$	$\mathtt{0000000000000111}_b$	16
$\mathtt{F9}_x$	$\mathtt{0000000000000110}_b$	16
$\mathtt{FA}_x$	$\mathtt{0000000000000101}_b$	16
$\mathtt{FB}_x$	$\mathtt{0000000000000100}_b$	16
$\mathtt{FC}_x$	$\mathtt{0000000000000011}_b$	16
$\mathtt{FD}_x$	$\mathtt{0000000000000010}_b$	16
$\mathtt{FE}_x$	$\mathtt{0000000000000001}_b$	16
$\mathtt{FF}_x$	$\mathtt{0000000000000000}_b$	16