Pointer

```
case PixFmt.A8_R8_G8_B8:
    switch (dst.PixFormat)
    {
        case PixFmt.R8_G8_B8:
            copyLine = (pSrc, iSrc, pDst, iDst, nPixels) =>
            {
                pSrc += iSrc;
                pDst += iDst;
                byte* pEndSrc = pSrc + nPixels * BytesPerPixel;

                while (pSrc < pEndSrc)
                {
                    pSrc++;              // skip alpha
                    *pDst++ = *pSrc++;  // copy R
                    *pDst++ = *pSrc++;  // copy G
                    *pDst++ = *pSrc++;  // copy B
                }
            };

            break;
        case PixFmt.L8:

            break;
    }
    break;
```

Streaming SIMD Extension

```
case PixFmt.A8_R8_G8_B8:
switch (dst.PixFormat)
{
    case PixFmt.R8_G8_B8:

        if (Ssse3.IsSupported)
        {
            byte[] arrShufMask = {1, 2, 3, 5,
                                  6, 7, 9, 10,
                                  11, 13, 14, 15,
                                  0xFF, 0xFF, 0xFF, 0xFF};
            fixed (byte *pShufMask = arrShufMask)
            {
                var vShufMask = Ssse3.LoadVector128(pShufMask);

                copyLine = (pSrc, iSrc, pDst, iDst, nPixels) =>
                {
                    pSrc += iSrc;
                    pDst += iDst;
                    int nBytes = nPixels * BytesPerPixel;
                    int nBytesPacked = nBytes - nBytes % 16;
                    byte* pEndSrc       = pSrc + nBytes;
                    byte* pEndSrcPacked = pSrc + nBytesPacked;

                    while (pSrc < pEndSrcPacked)
                    {
                        Vector128<byte> vSrc = Ssse3.LoadVector128(pSrc);
                        Vector128<byte> vDst = Ssse3.Shuffle(vSrc, vShufMask);
                        Ssse3.Store(pDst, vDst);
                        pSrc+=16;
                        pDst+=12;
                    }

                    while (pSrc < pEndSrc)
                    {
                        pSrc++;              // skip alpha
                        *pDst++ = *pSrc++;   // copy R
                        *pDst++ = *pSrc++;   // copy G
                        *pDst++ = *pSrc++;   // copy B
                    }
                };
            }
        }
        // else // fallback to pointer access
}
break;
```