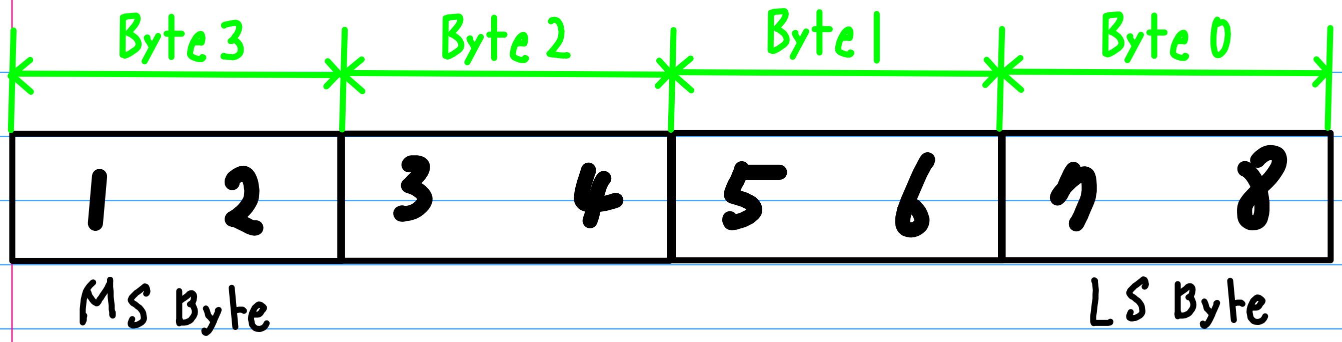
C Programming Day19.B

2017.11.21

Union, Bit Field, Macros

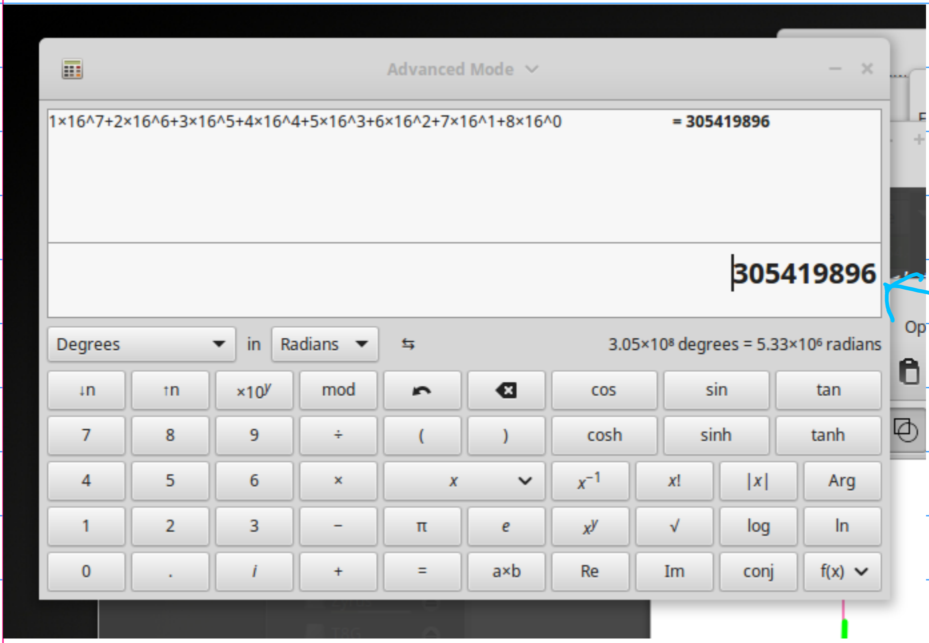


0X12345678

$= 1 \times 16^7 + 2 \times 16^6 + 3 \times 16^5 + 4 \times 16^4 + 5 \times 16^3 + 6 \times 16^2 + 7 \times 16^1 + 8 \times 16^0$

most significant

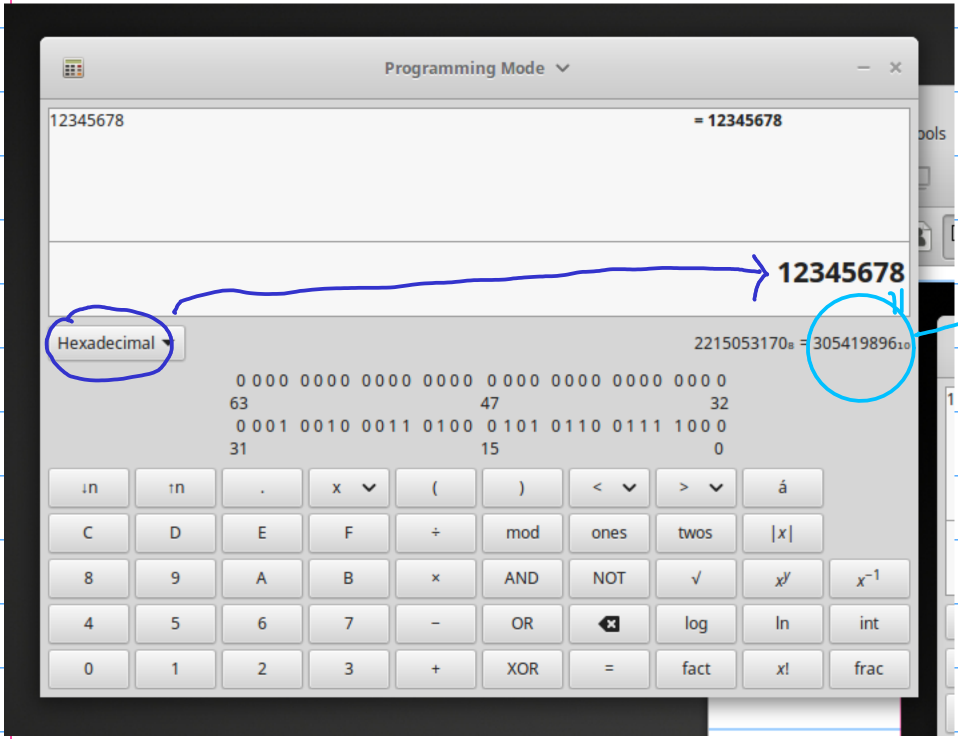
least significant



Advanced Mode

```
1×16^7+2×16^6+3×16^5+4×16^4+5×16^3+6×16^2+7×16^1+8×16^0        = 305419896

305419896
```

Degrees in Radians — 3.05×10⁸ degrees = 5.33×10⁶ radians



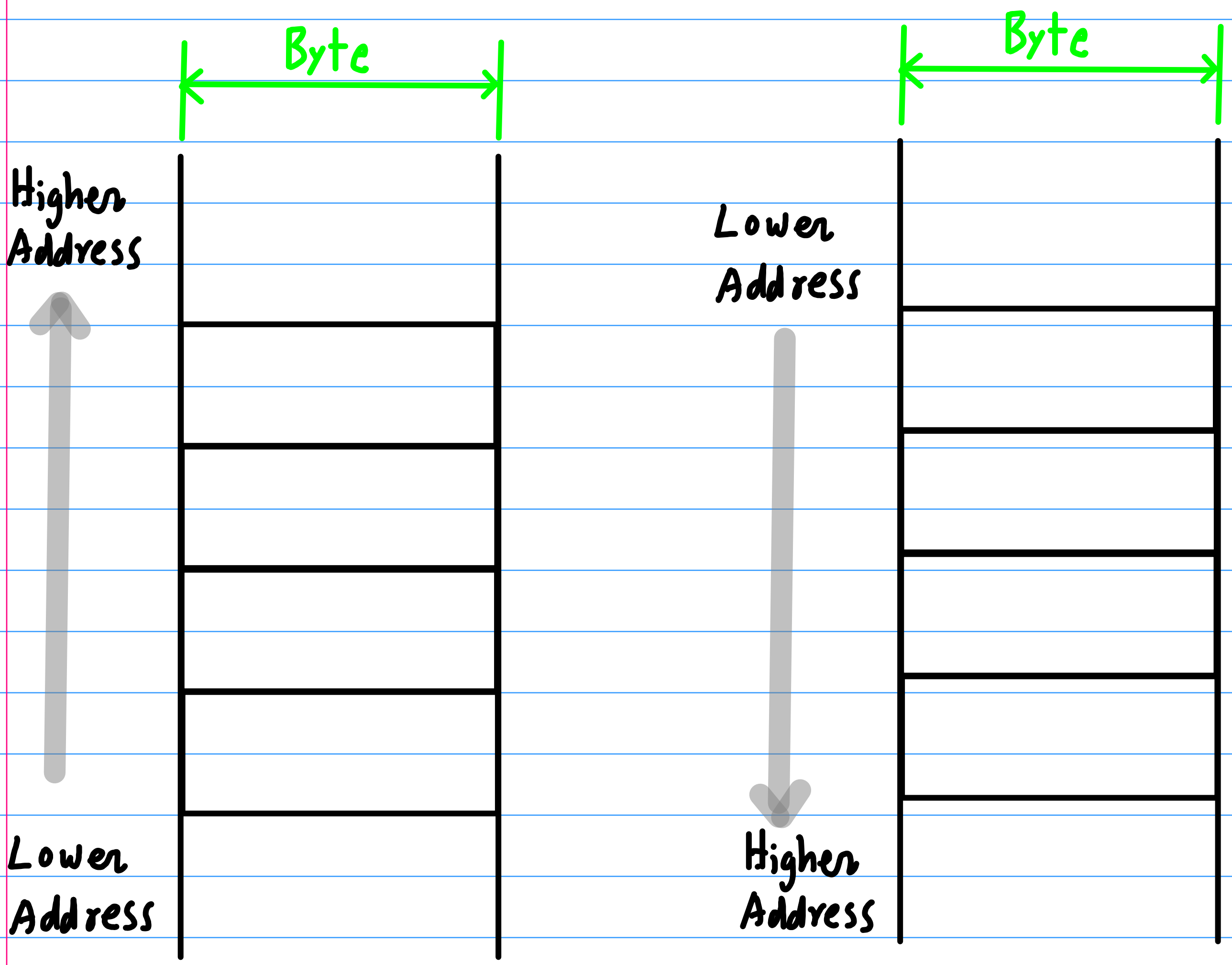
Programming Mode

```
12345678                                                      = 12345678

12345678
```

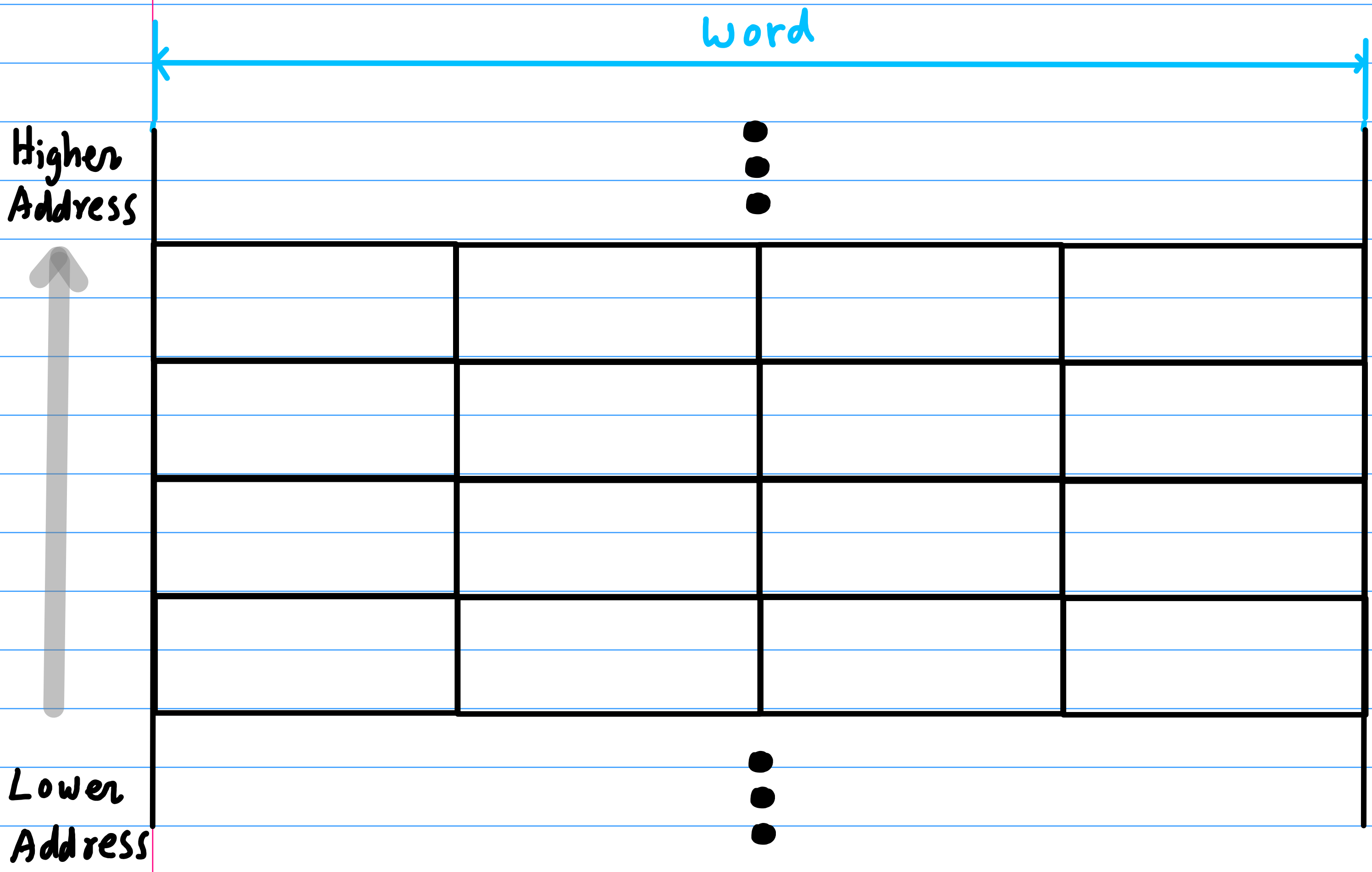
Hexadecimal — 2215053170₈ = 305419896₁₀

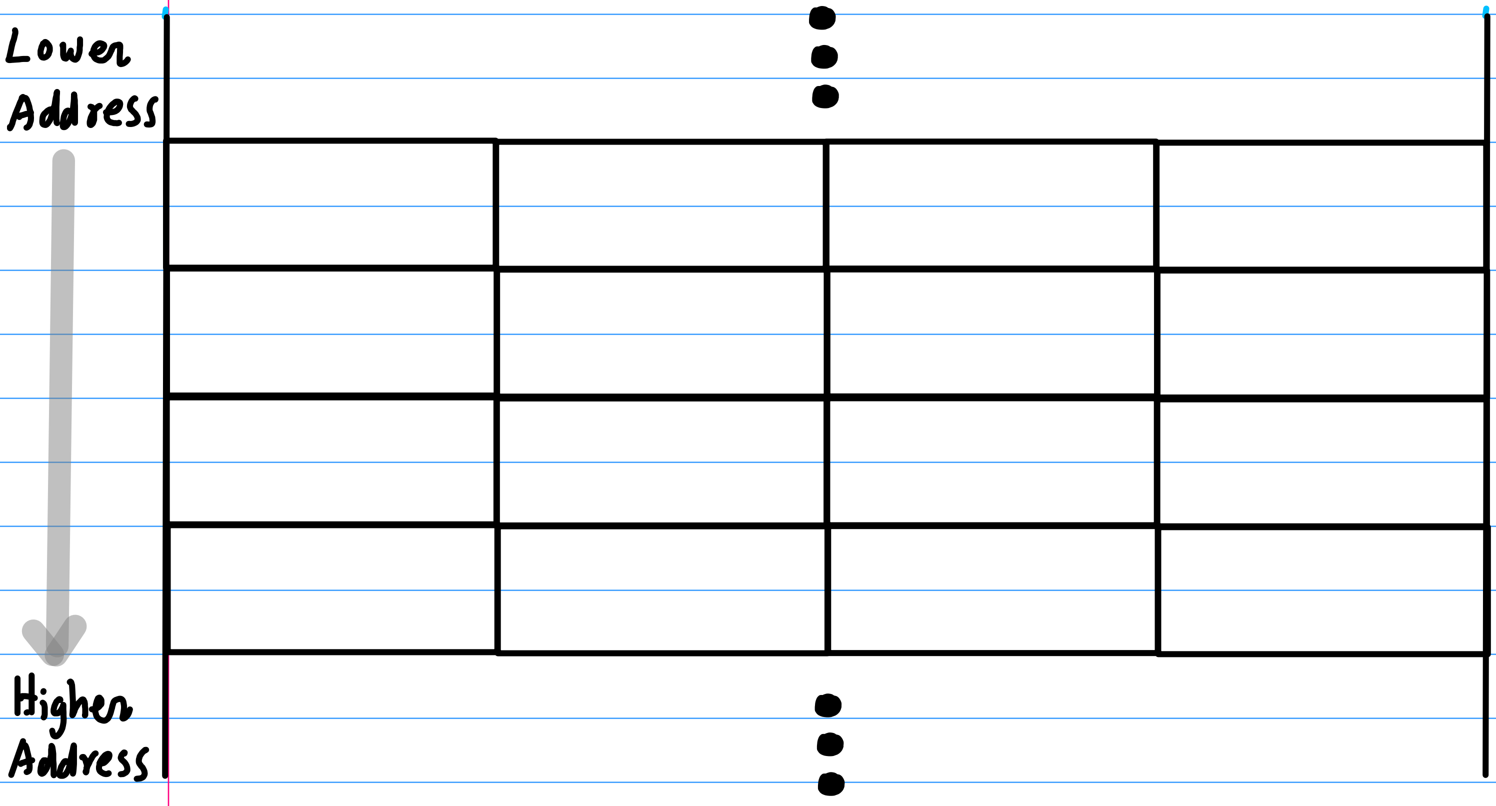
```
0 000  0000  0000  0000  0 000  0000  0000  000 0
63                       47                      32
0 001  0010  0011  0100  0 101  0110  0111  100 0
31                       15                       0
```

Byte Address



Word Address 32 bit = 4 Bytes = 1 word

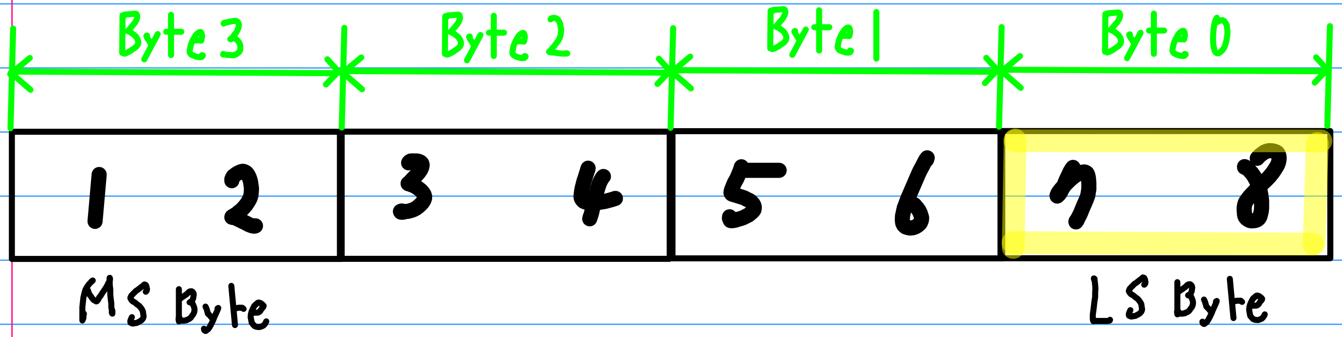


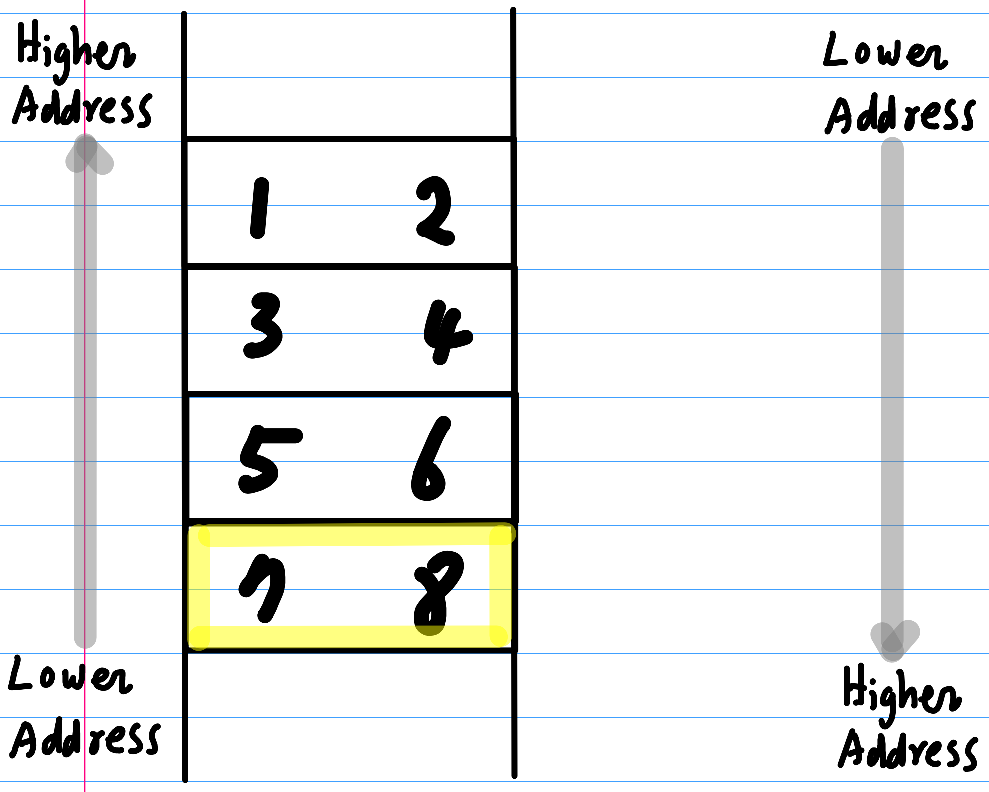


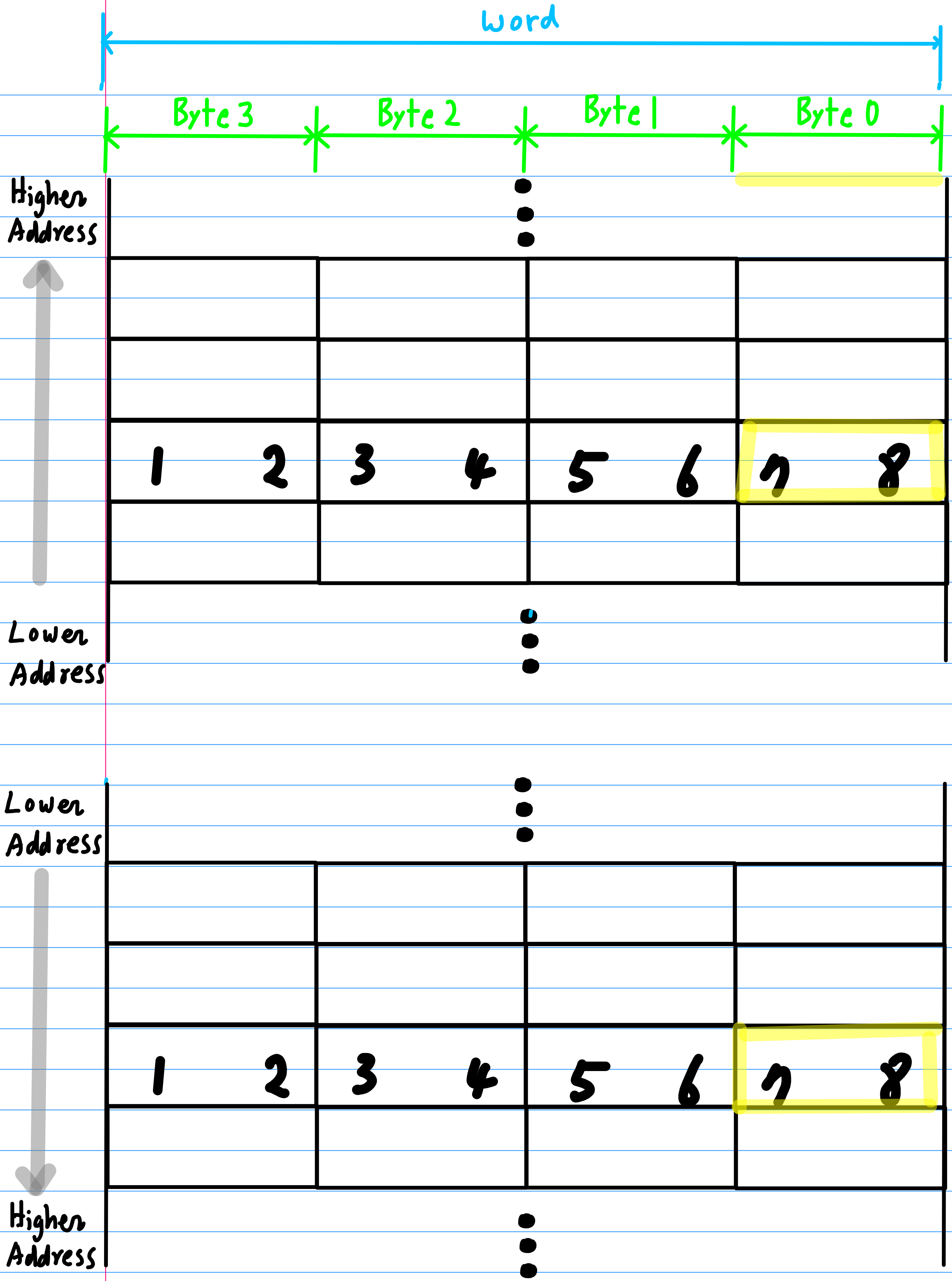
Little Endian.

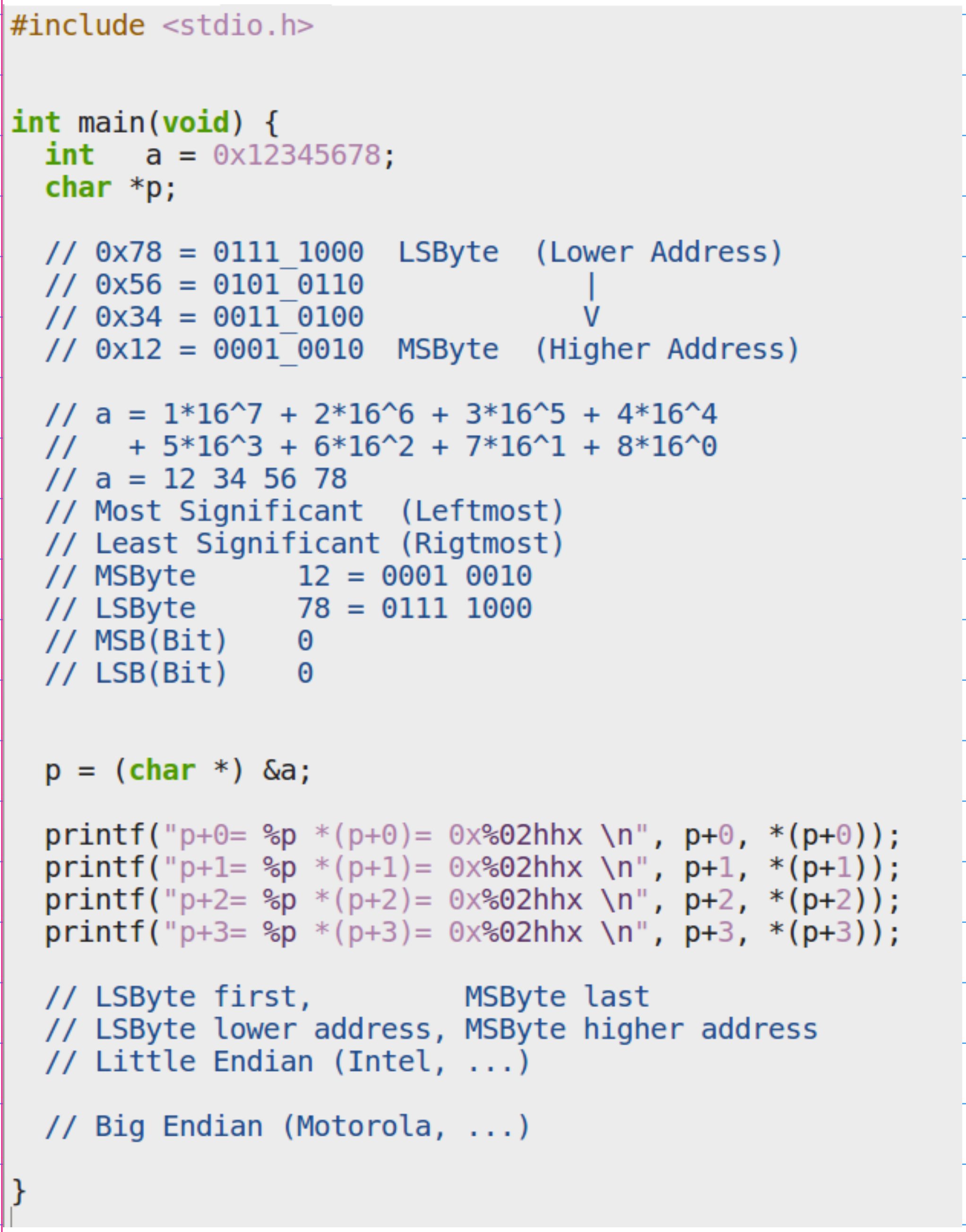
LS Byte First (lower address)

MS Byte Last (Higher Address)

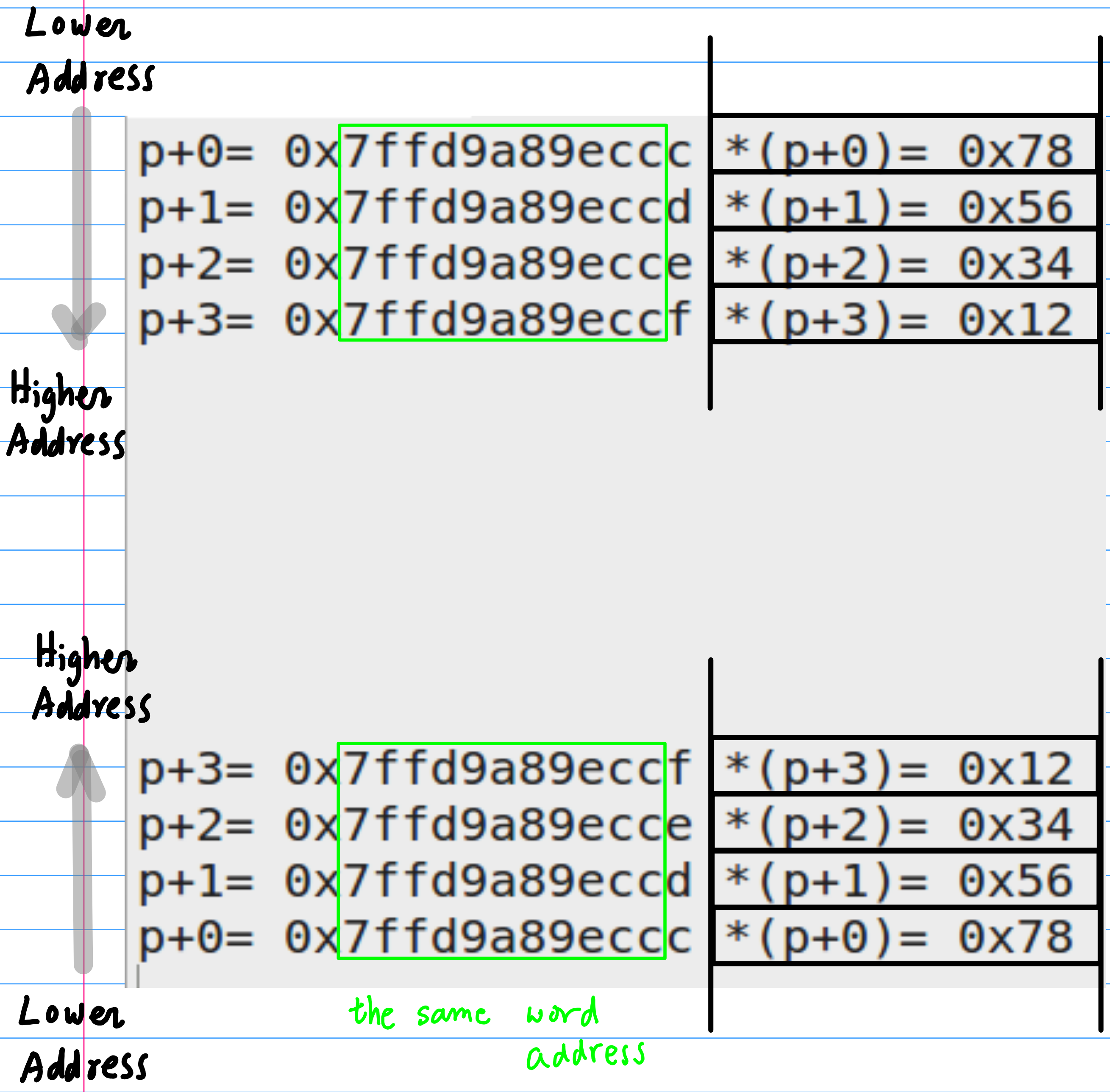





Word
Byte 3
Byte 2
Byte 1
Byte 0
Higher Address
Lower Address
1 2 3 4 5 6 7 8
Lower Address
Higher Address
1 2 3 4 5 6 7 8



```c
#include <stdio.h>


int main(void) {
  int   a = 0x12345678;
  char *p;

  // 0x78 = 0111_1000  LSByte  (Lower Address)
  // 0x56 = 0101_0110             |
  // 0x34 = 0011_0100             V
  // 0x12 = 0001_0010  MSByte  (Higher Address)

  // a = 1*16^7 + 2*16^6 + 3*16^5 + 4*16^4
  //   + 5*16^3 + 6*16^2 + 7*16^1 + 8*16^0
  // a = 12 34 56 78
  // Most Significant  (Leftmost)
  // Least Significant (Rigtmost)
  // MSByte      12 = 0001 0010
  // LSByte      78 = 0111 1000
  // MSB(Bit)    0
  // LSB(Bit)    0


  p = (char *) &a;

  printf("p+0= %p *(p+0)= 0x%02hhx \n", p+0, *(p+0));
  printf("p+1= %p *(p+1)= 0x%02hhx \n", p+1, *(p+1));
  printf("p+2= %p *(p+2)= 0x%02hhx \n", p+2, *(p+2));
  printf("p+3= %p *(p+3)= 0x%02hhx \n", p+3, *(p+3));

  // LSByte first,         MSByte last
  // LSByte lower address, MSByte higher address
  // Little Endian (Intel, ...)

  // Big Endian (Motorola, ...)

}
```



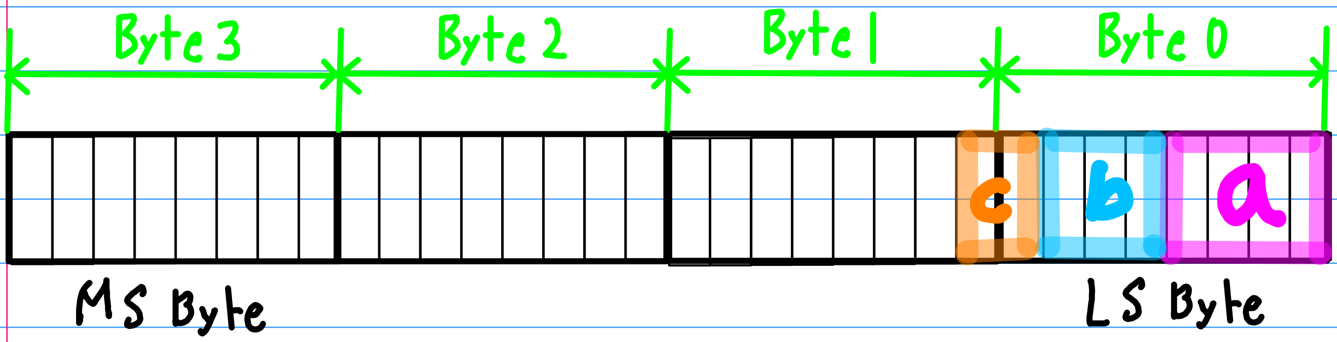
```
p+0= 0x7ffd9a89eccc
p+1= 0x7ffd9a89eccd
p+2= 0x7ffd9a89ecce
p+3= 0x7ffd9a89eccf
```

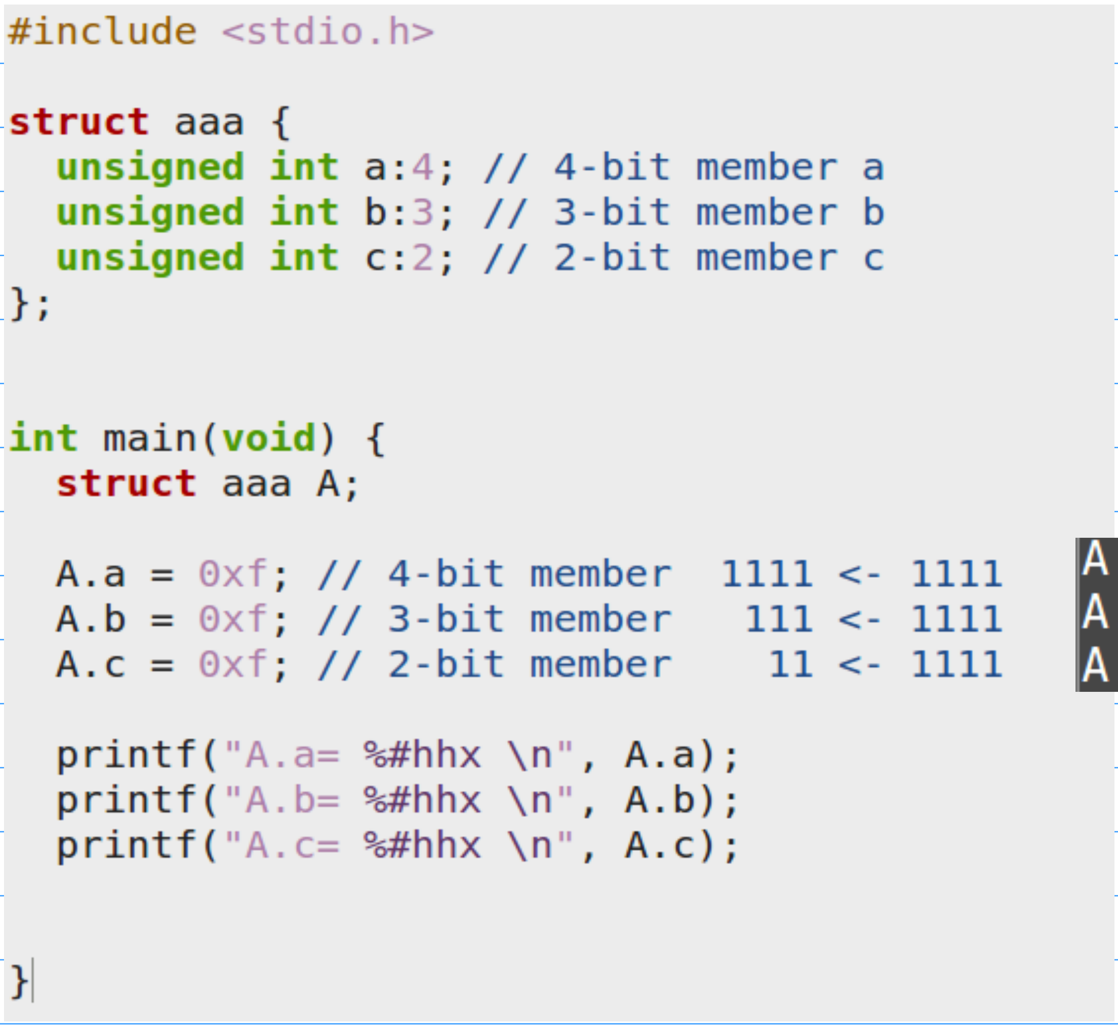
```
*(p+0)= 0x78
*(p+1)= 0x56
*(p+2)= 0x34
*(p+3)= 0x12
```

```
p+3= 0x7ffd9a89eccf
p+2= 0x7ffd9a89ecce
p+1= 0x7ffd9a89eccd
p+0= 0x7ffd9a89eccc
```

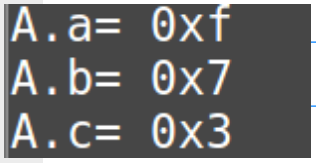
```
*(p+3)= 0x12
*(p+2)= 0x34
*(p+1)= 0x56
*(p+0)= 0x78
```

Bit Field

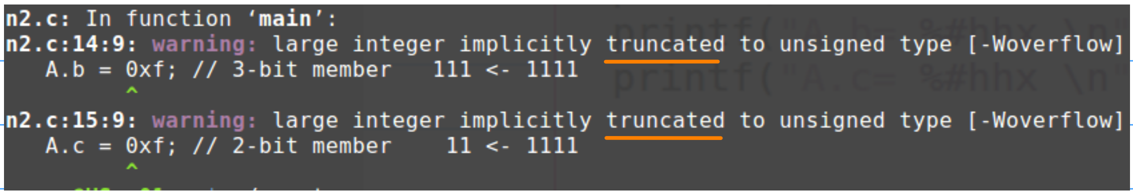




```c
#include <stdio.h>

struct aaa {
  unsigned int a:4; // 4-bit member a
  unsigned int b:3; // 3-bit member b
  unsigned int c:2; // 2-bit member c
};


int main(void) {
  struct aaa A;

  A.a = 0xf; // 4-bit member  1111 <- 1111
  A.b = 0xf; // 3-bit member   111 <- 1111
  A.c = 0xf; // 2-bit member    11 <- 1111

  printf("A.a= %#hhx \n", A.a);
  printf("A.b= %#hhx \n", A.b);
  printf("A.c= %#hhx \n", A.c);


}
```

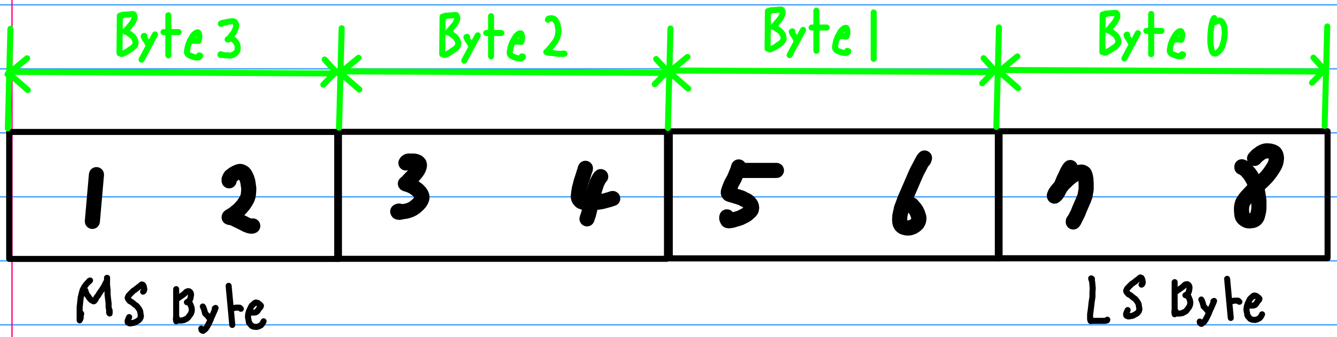


```
A.a= 0xf
A.b= 0x7
A.c= 0x3
```

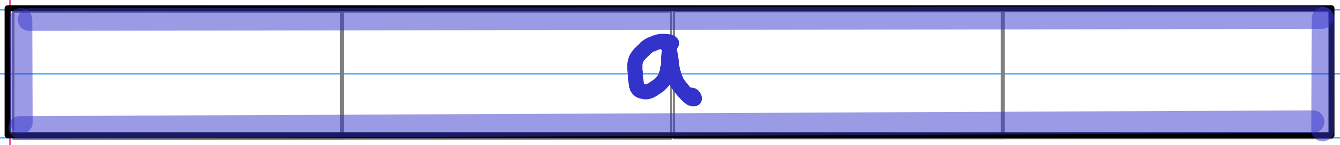


```
n2.c: In function ‘main’:
n2.c:14:9: warning: large integer implicitly truncated to unsigned type [-Woverflow]
   A.b = 0xf; // 3-bit member   111 <- 1111
         ^
n2.c:15:9: warning: large integer implicitly truncated to unsigned type [-Woverflow]
   A.c = 0xf; // 2-bit member    11 <- 1111
         ^
```

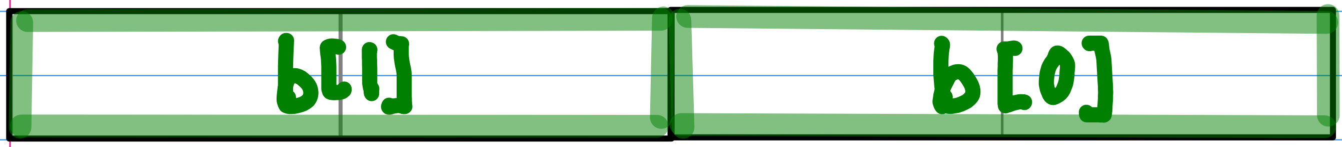
Union



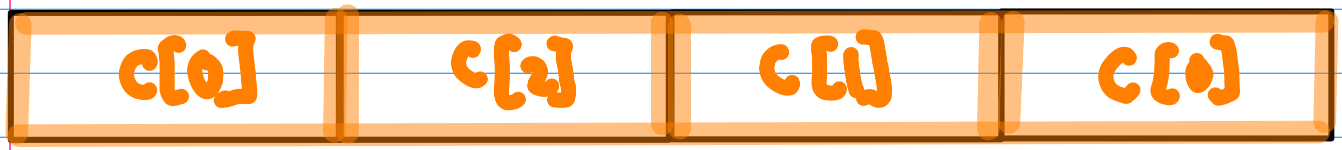
integer view

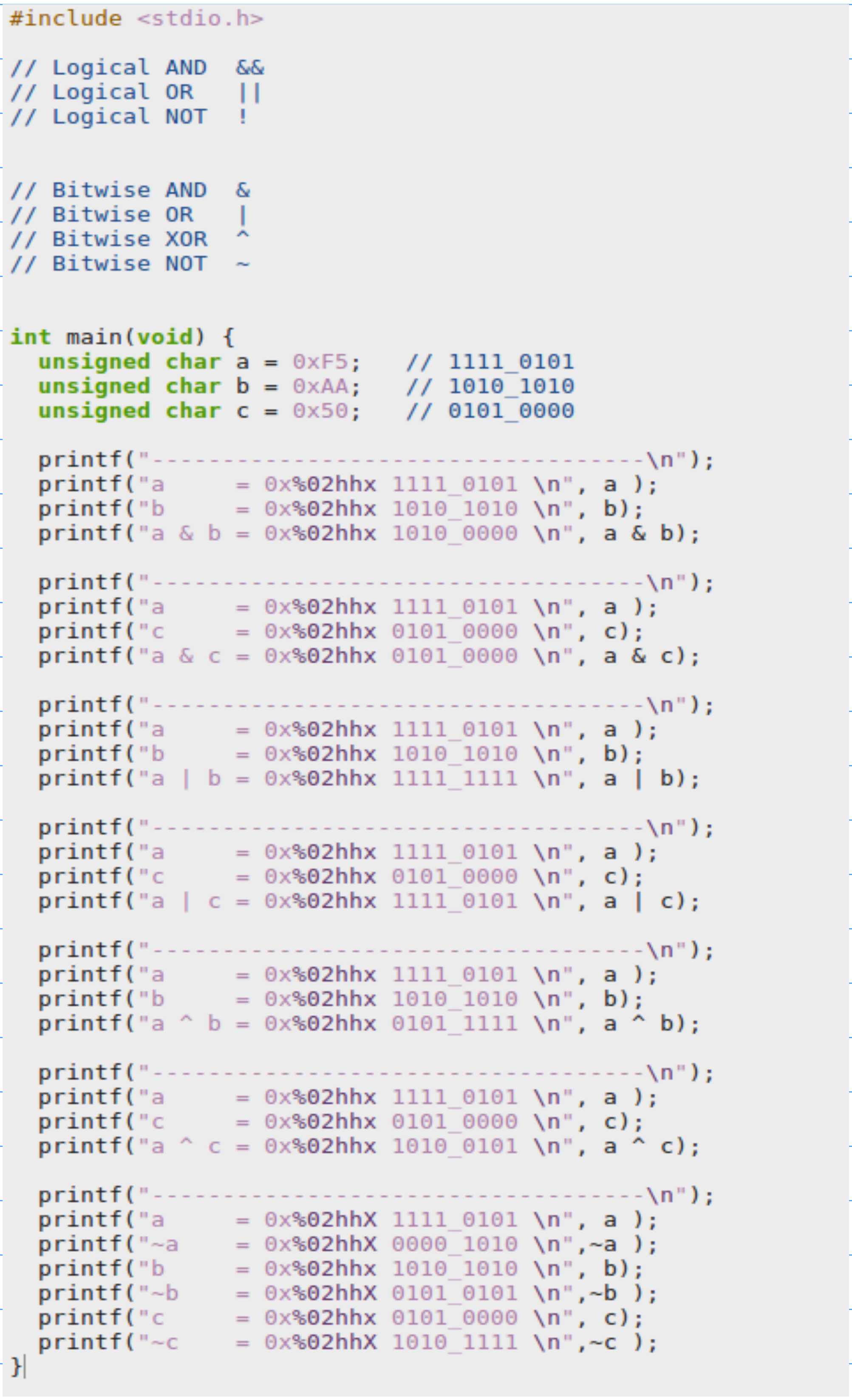


Short view

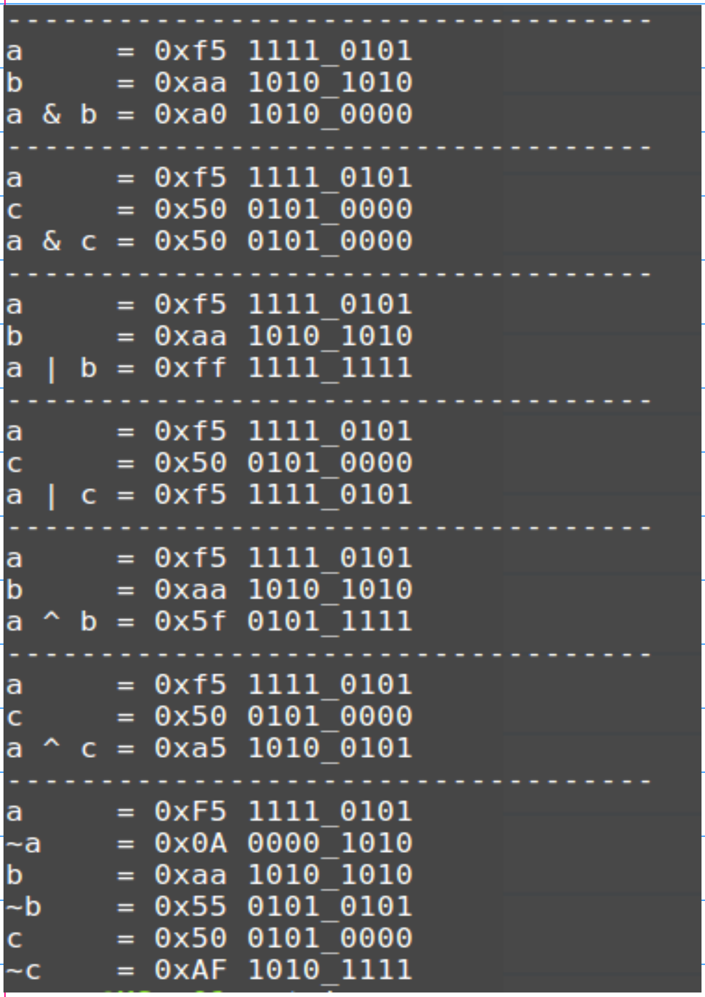


Char view

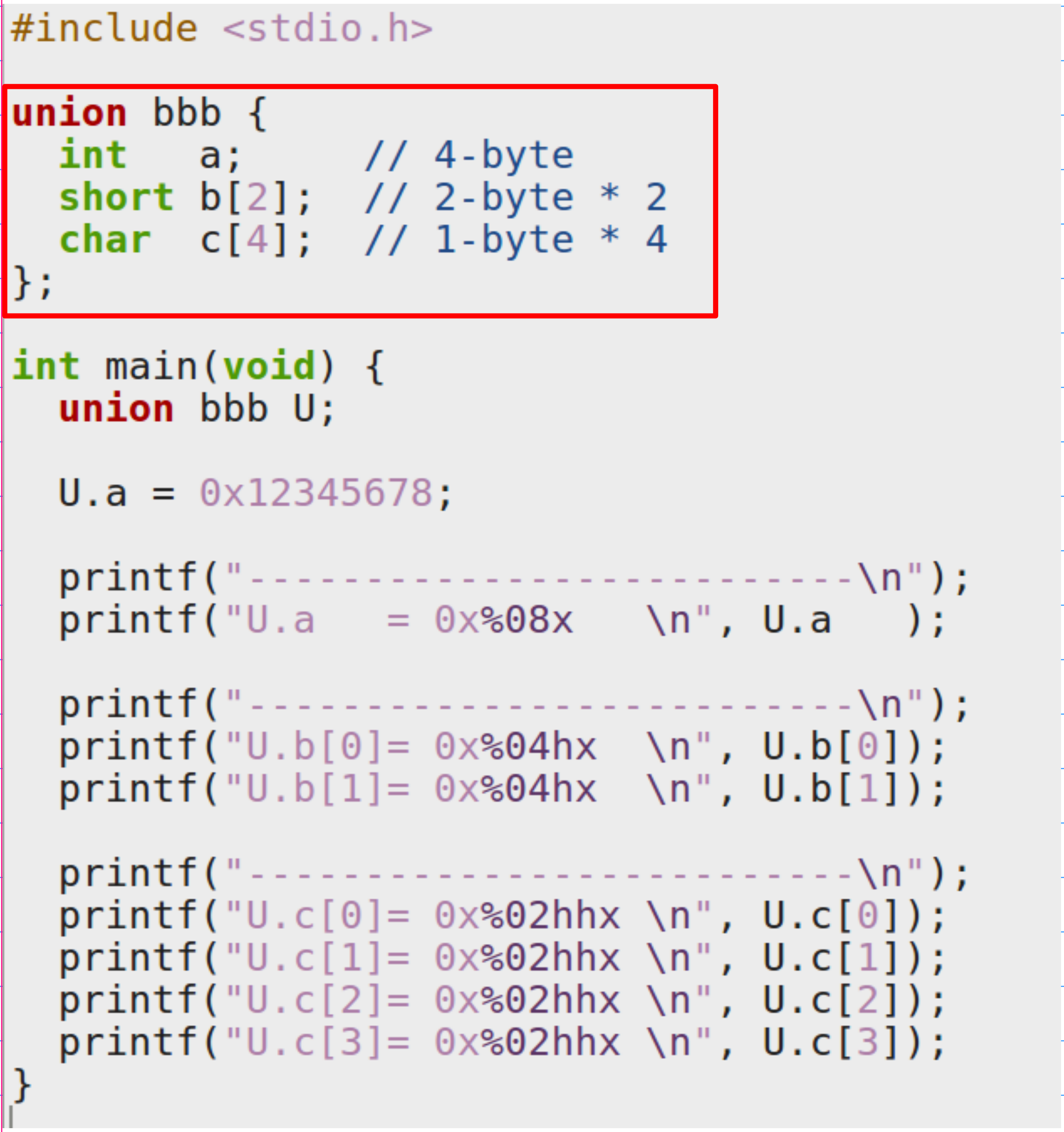




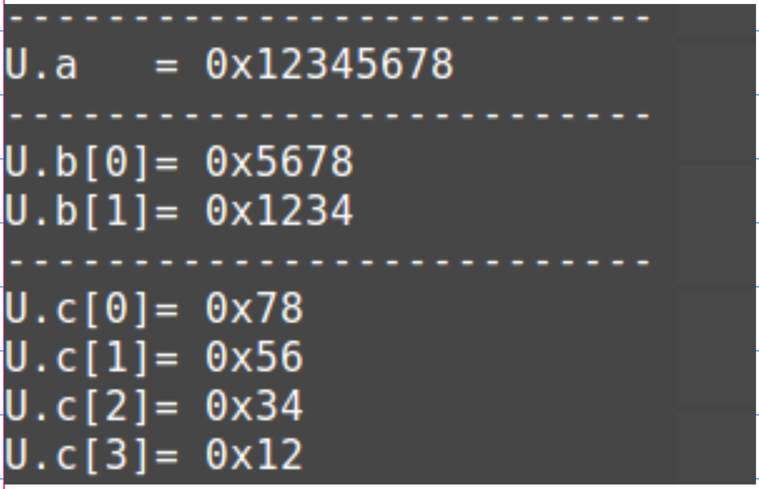
```c
#include <stdio.h>

// Logical AND  &&
// Logical OR   ||
// Logical NOT  !


// Bitwise AND  &
// Bitwise OR   |
// Bitwise XOR  ^
// Bitwise NOT  ~


int main(void) {
  unsigned char a = 0xF5;   // 1111_0101
  unsigned char b = 0xAA;   // 1010_1010
  unsigned char c = 0x50;   // 0101_0000

  printf("-----------------------------------\n");
  printf("a     = 0x%02hhx 1111_0101 \n", a );
  printf("b     = 0x%02hhx 1010_1010 \n", b);
  printf("a & b = 0x%02hhx 1010_0000 \n", a & b);

  printf("-----------------------------------\n");
  printf("a     = 0x%02hhx 1111_0101 \n", a );
  printf("c     = 0x%02hhx 0101_0000 \n", c);
  printf("a & c = 0x%02hhx 0101_0000 \n", a & c);

  printf("-----------------------------------\n");
  printf("a     = 0x%02hhx 1111_0101 \n", a );
  printf("b     = 0x%02hhx 1010_1010 \n", b);
  printf("a | b = 0x%02hhx 1111_1111 \n", a | b);

  printf("-----------------------------------\n");
  printf("a     = 0x%02hhx 1111_0101 \n", a );
  printf("c     = 0x%02hhx 0101_0000 \n", c);
  printf("a | c = 0x%02hhx 1111_0101 \n", a | c);

  printf("-----------------------------------\n");
  printf("a     = 0x%02hhx 1111_0101 \n", a );
  printf("b     = 0x%02hhx 1010_1010 \n", b);
  printf("a ^ b = 0x%02hhx 0101_1111 \n", a ^ b);

  printf("-----------------------------------\n");
  printf("a     = 0x%02hhx 1111_0101 \n", a );
  printf("c     = 0x%02hhx 0101_0000 \n", c);
  printf("a ^ c = 0x%02hhx 1010_0101 \n", a ^ c);

  printf("-----------------------------------\n");
  printf("a     = 0x%02hhX 1111_0101 \n", a );
  printf("~a    = 0x%02hhX 0000_1010 \n",~a );
  printf("b     = 0x%02hhx 1010_1010 \n", b);
  printf("~b    = 0x%02hhX 0101_0101 \n",~b );
  printf("c     = 0x%02hhx 0101_0000 \n", c);
  printf("~c    = 0x%02hhX 1010_1111 \n",~c );
}
```



```
---------------------------------
a       = 0xf5 1111_0101
b       = 0xaa 1010_1010
a & b = 0xa0 1010_0000
---------------------------------
a       = 0xf5 1111_0101
c       = 0x50 0101_0000
a & c = 0x50 0101_0000
---------------------------------
a       = 0xf5 1111_0101
b       = 0xaa 1010_1010
a | b = 0xff 1111_1111
---------------------------------
a       = 0xf5 1111_0101
c       = 0x50 0101_0000
a | c = 0xf5 1111_0101
---------------------------------
a       = 0xf5 1111_0101
b       = 0xaa 1010_1010
a ^ b = 0x5f 0101_1111
---------------------------------
a       = 0xf5 1111_0101
c       = 0x50 0101_0000
a ^ c = 0xa5 1010_0101
---------------------------------
a       = 0xF5 1111_0101
~a      = 0x0A 0000_1010
b       = 0xaa 1010_1010
~b      = 0x55 0101_0101
c       = 0x50 0101_0000
~c      = 0xAF 1010_1111
```



```c
#include <stdio.h>

union bbb {
  int   a;      // 4-byte
  short b[2];   // 2-byte * 2
  char  c[4];   // 1-byte * 4
};

int main(void) {
  union bbb U;

  U.a = 0x12345678;

  printf("--------------------------\n");
  printf("U.a    = 0x%08x   \n", U.a   );

  printf("--------------------------\n");
  printf("U.b[0]= 0x%04hx  \n", U.b[0]);
  printf("U.b[1]= 0x%04hx  \n", U.b[1]);

  printf("--------------------------\n");
  printf("U.c[0]= 0x%02hhx \n", U.c[0]);
  printf("U.c[1]= 0x%02hhx \n", U.c[1]);
  printf("U.c[2]= 0x%02hhx \n", U.c[2]);
  printf("U.c[3]= 0x%02hhx \n", U.c[3]);
}
```



```
--------------------------
U.a    = 0x12345678
--------------------------
U.b[0]= 0x5678
U.b[1]= 0x1234
--------------------------
U.c[0]= 0x78
U.c[1]= 0x56
U.c[2]= 0x34
U.c[3]= 0x12
```

gcc

If you only want some of the stages of compilation, you can use -x (or filename suffixes) to tell gcc where to start, and one of the options -c, -S, or -E to say where gcc is to stop. Note that some combinations (for example, -x cpp-output -E) instruct gcc to do nothing at all.

-E means: stop after the preprocessing stage; do not run the compiler proper. The output is in the form of preprocessed source code, which is sent to the standard output (or to the output file if -o is specified).

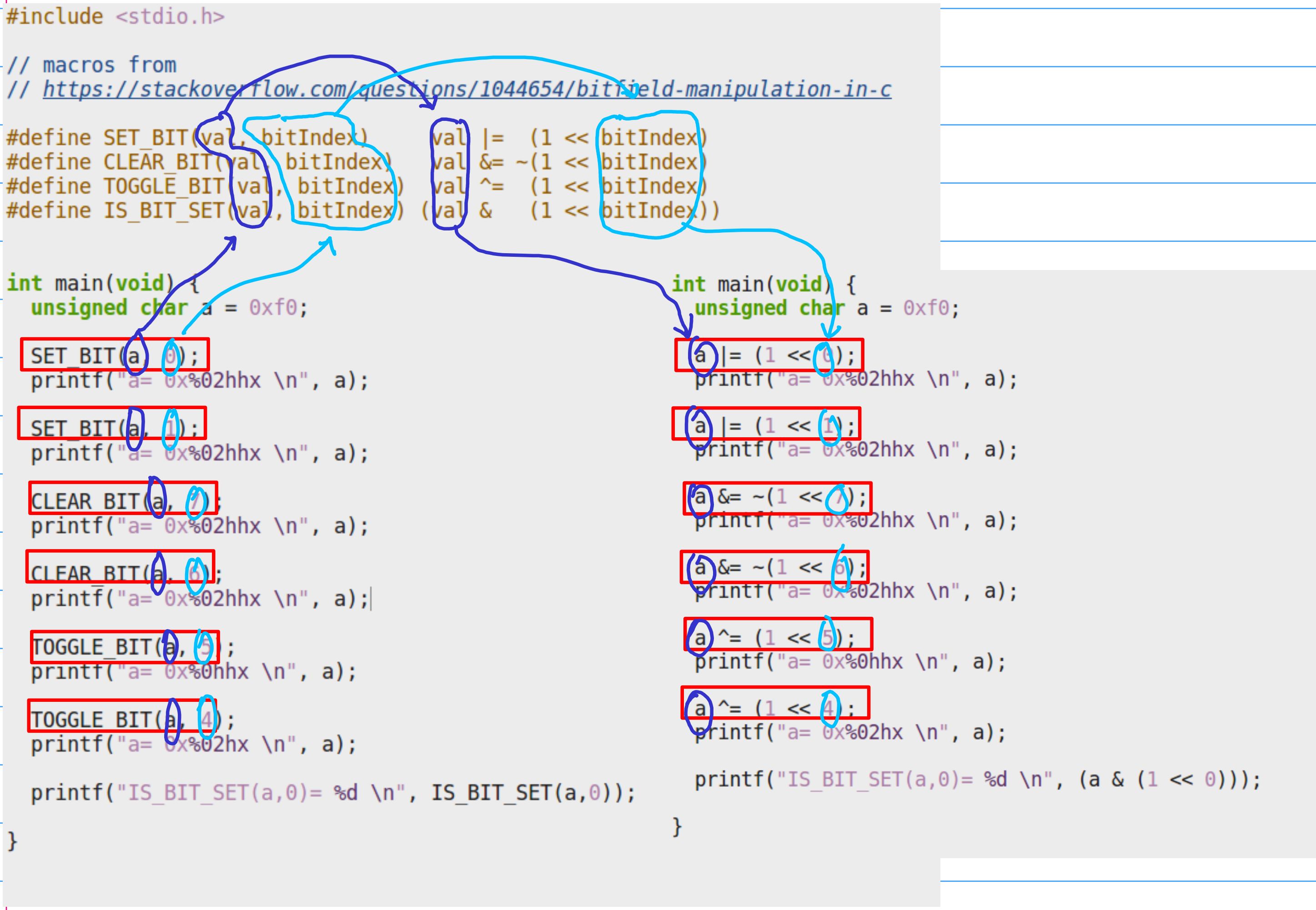
If you use the -E option, nothing is done except preprocessing.

-save-temps

Store the usual "temporary" intermediate files permanently; place them in the current directory and name them based on the source file.

Thus, compiling foo.c with -c -save-temps would produce files foo.i and foo.s, as well as foo.o.

This creates a preprocessed foo.i output file even though the compiler now normally uses an integrated preprocessor.



```c
#include <stdio.h>

// macros from
// https://stackoverflow.com/questions/1044654/bitfield-manipulation-in-c

#define SET_BIT(val, bitIndex)      val |=  (1 << bitIndex)
#define CLEAR_BIT(val, bitIndex)    val &= ~(1 << bitIndex)
#define TOGGLE_BIT(val, bitIndex)   val ^=  (1 << bitIndex)
#define IS_BIT_SET(val, bitIndex) (val &   (1 << bitIndex))


int main(void) {
  unsigned char a = 0xf0;

  SET_BIT(a, 0);
  printf("a= 0x%02hhx \n", a);

  SET_BIT(a, 1);
  printf("a= 0x%02hhx \n", a);

  CLEAR_BIT(a, 7);
  printf("a= 0x%02hhx \n", a);

  CLEAR_BIT(a, 6);
  printf("a= 0x%02hhx \n", a);

  TOGGLE_BIT(a, 5);
  printf("a= 0x%0hhx \n", a);

  TOGGLE_BIT(a, 4);
  printf("a= 0x%02hx \n", a);

  printf("IS_BIT_SET(a,0)= %d \n", IS_BIT_SET(a,0));

}
```

```c
int main(void) {
  unsigned char a = 0xf0;

  a |= (1 << 0);
  printf("a= 0x%02hhx \n", a);

  a |= (1 << 1);
  printf("a= 0x%02hhx \n", a);

  a &= ~(1 << 7);
  printf("a= 0x%02hhx \n", a);

  a &= ~(1 << 6);
  printf("a= 0x%02hhx \n", a);

  a ^= (1 << 5);
  printf("a= 0x%0hhx \n", a);

  a ^= (1 << 4);
  printf("a= 0x%02hx \n", a);

  printf("IS_BIT_SET(a,0)= %d \n", (a & (1 << 0)));
}
```