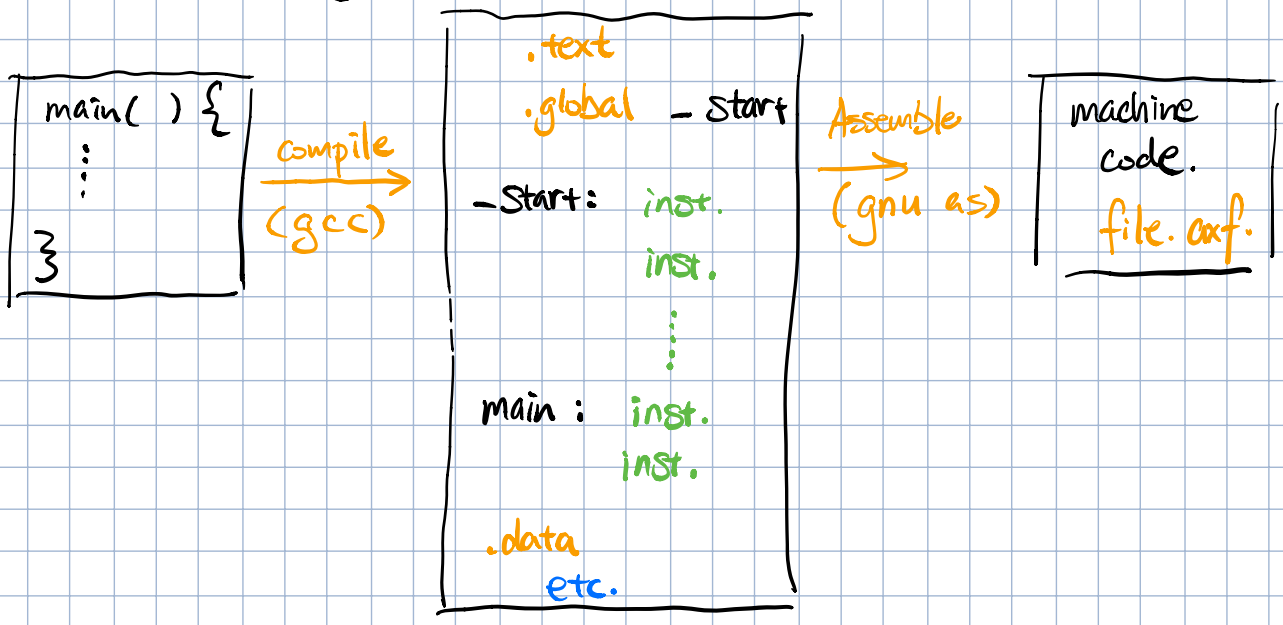
Using C code

- When you compile C code the compiler generates Assembly code then machine code:



Notes:

- If you have more than one C file, Then each one is compiled / assembled to create separate machine code files. (called object files) These object files (*.o) are combined by the linker into file.axf.

- _start is not main. Instead, it is the code called the common start-up code. sequence. It initialize sp. and then perform various steps needed to start up a C program. (sets initial variable values, sets uninitialized global variables to "0," ...). In CPUlator, or Amp. search for main to find the start of your code.

Reading / Writing to I/O devices (using C)

- in Assembly code, to read sw / write to LEDR:

```
        mov   R1, #0
        movT  R1, #0xFF20        ) or  LDR R1, =0xFF200000
LOOP:   LDR   R3, [R1, #0X40]    // read FF200040 (sw)
        STR   R3, [R1]           // write to LEDR
        B
```

- in C code:

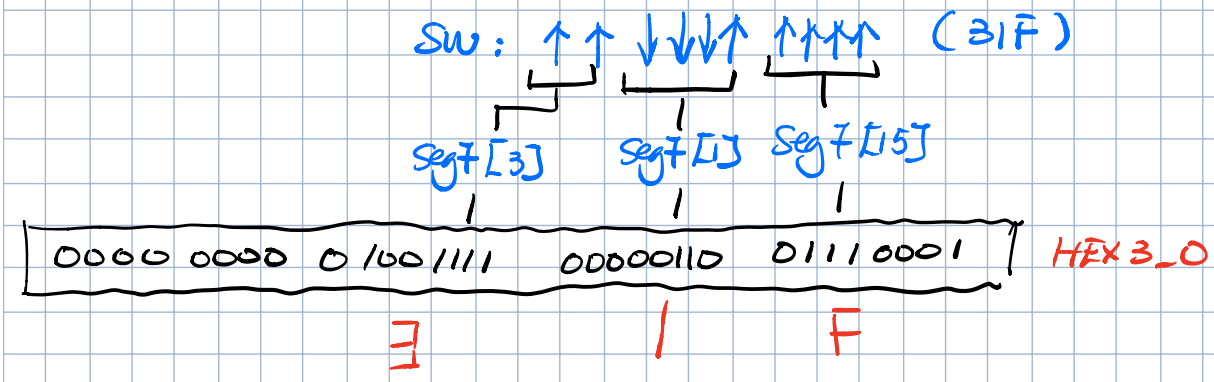
```
volatile    int   *LEDR_ptr = 0xFF200000;
volatile    int   *sw_ptr   = 0xFF200040;
int value;
While (1) {
    value = *sw_ptr;
    *LEDR_ptr = value;          ) (*)
}
```

(*) using *ptr is called pointer dereferencing. This is how we read / write to a specific addr.

volatile: ensures that the variable will always be accessed by using its memory address. Without volatile the sw_ptr might be read only once outside the loop into a register (e.g. R3) and then this register would be used in each iteration of the loop. You must always use volatile for pointers to I/O devices

- write C code to display SW on LEDR & HEX3-0

```
int seg7[] = { 0x3F, 0x06, 0x5B, 0x4F, ..., 0x71 };
```




```
int main() {
    int value;
    volatile int * LEDR_ptr = 0xFF200000;
    volatile int * SW_ptr = 0xFF200040;
    volatile int * HEX3_0_ptr = 0xFF200020;
    while (1) {
        value = *SW_ptr; // read SW
        *LEDR_ptr = value; // write LEDR
*HEX3_0_ptr = seg7[value & 0xF] | seg7[value>>4 & 0xF]<<8 |
                                  seg7[value>>8]<<16;
                                  // write HEX3_0
    }
}
```

Address pointers

```
0000025c <main>:
25c: e3a01000   mov   r1, #0
260: e34f1f20   movt  r1, #0xff20          // address of LEDR

264: e3002730   movw  r2, #0x730           // address of seg7[] array
268: e3402000   movt  r2, #0

26c: e5913040   ldr   r3, [r1, #0x40]
270: e5813000   str   r3, [r1]

274: e7d2c443   ldrb  r12, [r2, r3, asr #8]
278: e203000f   and   r0, r3, #15
27c: e7d20000   ldrb  r0, [r2, r0]
280: e180080c   orr   r0, r0, r12, lsl #16
284: e7e33253   ubfx  r3, r3, #4, #4
288: e7d23003   ldrb  r3, [r2, r3]
28c: e1803403   orr   r3, r0, r3, lsl #8
290: e5813020   str   r3, [r1, #0x20]      // write to HEX3_0

294: eafffff4   b  26c <main+0x10>
```

ubfx r3,r3,#4,#4 is equiv. to:
lsr r3,#4
and r3,#0xF