Proiect Structura si Organizarea Calculatoarelor.

Interactiunea C / Assembly.

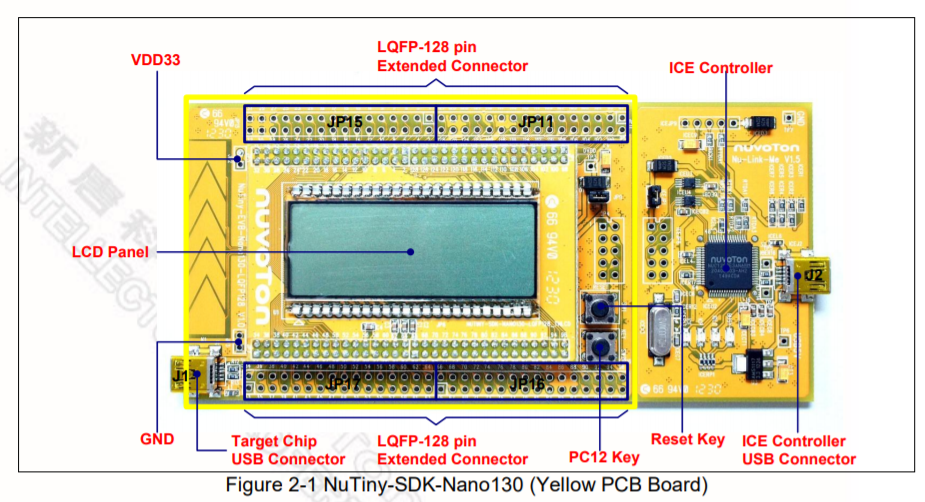
## Petrica Petru 1306B

# **Scopul proiectului:**

## **Crearea unei aplicatii pentru a demonstra interactiunea limbajului C cu Assembly prin apelul functiilor scrise in Assembly in interiorul codului C.**

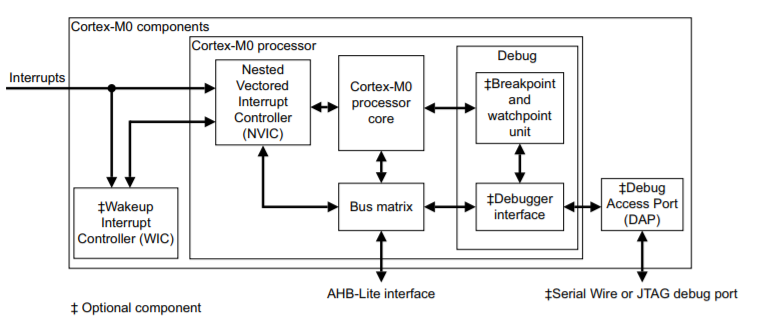
In practica, in cadrul sistemelor de performanta inalta si in special in sistemele embedded, in anumite situatii limbajul C nu ofera suficient control asupra resurselor hardware. Alteori compilatorul de C nu ofera nivelul de optimizare de care e nevoie si care se poate atinge scriind codul in asamblare. De asemenea, in cadrul proiectelor embedded nu este atat de evident avantajul limbajui C de a fi portabil din motiv ca codul, in general este specializat pentru o anumita platforma.

In cadrul acestui proiect mi-am propus sa realizez o aplicatie care sa demonstreze apelul unei functii scrise in asamblare din limbajul C. In cadrul acestui proiect voi lucra cu placa Nano130KE3BN furnizata de compania Nuvoton. Aceasta placuta dispune de un LCD display. Mi-am propus sa afisez un sir de caractere pe acest display.



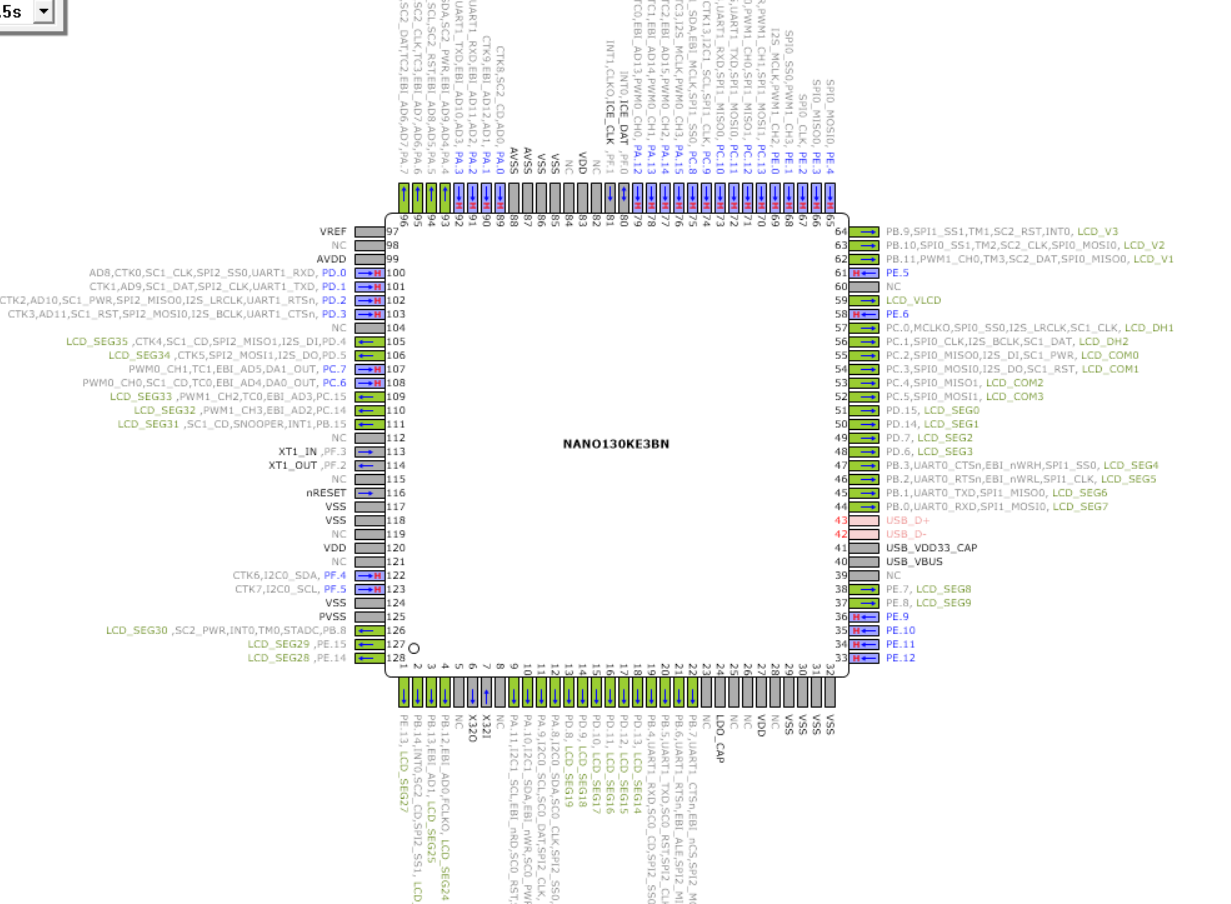
Aceasta dispune de un procesor Cortex-M0, ce implementeaza specificatia ARMv6-m, arhitectura de tip Von-Neumann.

## **Schiema procesorului Cortex-M0.**



## **Instrumente puse la dispozitie de compania Nuvoton.**

Pentru a incarca programul in memorie si a-l executa, voi folosi driverul pentru USB NuLink, acesta ofera si un set de instrumente pentru depanarea soft-ului.

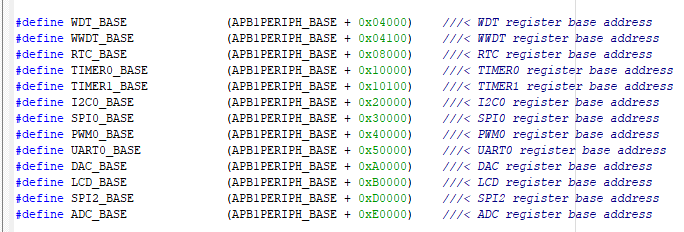


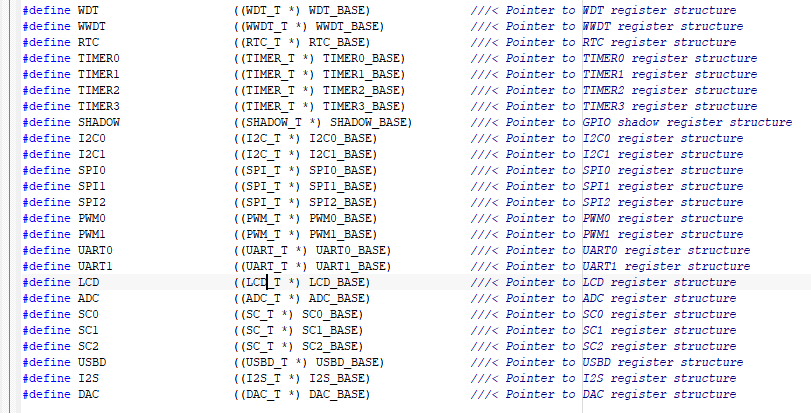
**NuTool PinView**

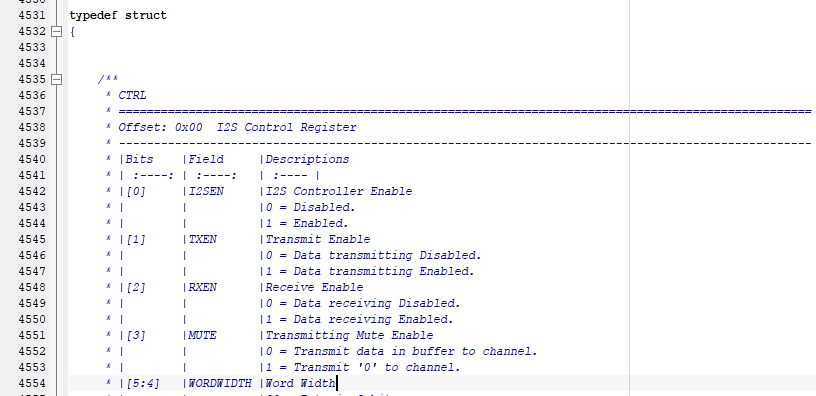
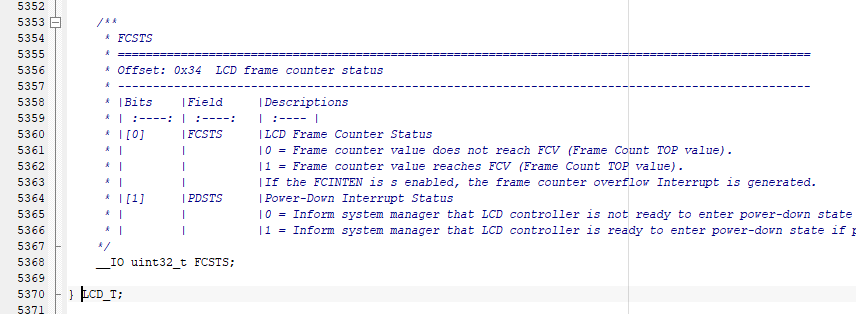
NuTool\_PinView ne ofera posibilitatea de a vizualiza configuratiile pinilor de pe placuta in timp real, astfel putem usor sa ne dam seama daca nu au fost activate functiile alternative ale portului, I/O pinii nu sunt configurati corect, etc.

## **Fisierele header puse la dispozitie de furnizor.**

Nuvoton pune la dispozitia programatorilor un set de fisiere header si surse C ce contin structuri mapate in memorie peste adresele unde sunt mapate componentele hardware, de asemenea in aceste fisiere este continuta documentatia pentru fiecare structura / registru / pin sub forma unor comentarii. Voi folosi aceste fisiere header in scopul de a evita scrierea acestor structuri, care sunt doar transpuse din datasheet, fiecare registru aferent unei componente este definit in aceastra structura prin intermediul unui volatile unsigned int si se mapeaza direct peste adresa sa reala ca memory-mapped I/O.

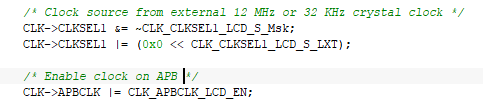




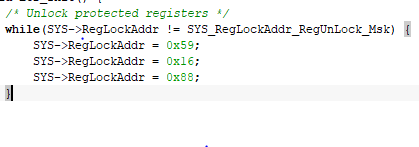


## **Setarea ceas-ului pentru LCD**

Pentru a afisa ceva pe componenta de LCD, este nevoie in primul rand de configurat sursa de clock pentru display, pentru asta vom scrie in registrul CLKSEL1, bit-ul aferent cristalului de frecventa joasa. De asemenea trebuie de activa linia clock-ului pentru LCD din cadrul APB-ului (Advanced Peripheral Bus) de pe placuta.

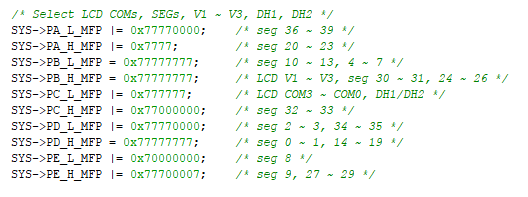


Acesti registri sunt protejati, pentru scrierea in ei, este nevoie de a-i debloca, pentru aceasta este nevoie scrierea registrului SYS->RegLockAddr cu anumite constante.

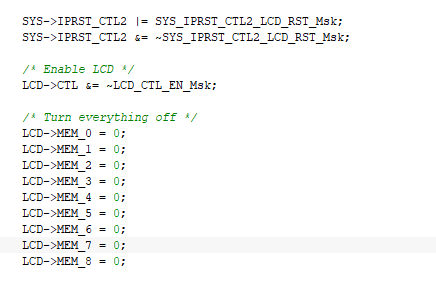


## **Configurarea pinilor.**

De asemeneaeste necesara configurarea pinilor aferenti pentru controlul LCD-ului, anume pinii vor reprezenta segmentele si com-urile, controlul segmentelor de pe display este prin intermediul I/O pinilor, care au functie alternativa SEG, acestia sunt multiplexati prin intermediul bitilor COM 3 – 0 pentru a permite controlul display-ului folosind cat mai putini pini.

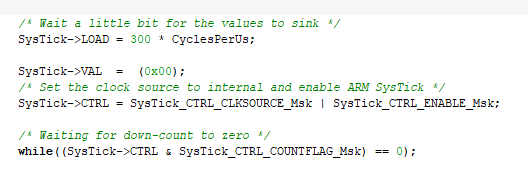


## **Curatarea display-ului.**

In continuare, in exemplele Demo, a fost recomandat sa resetam LCD-ul, dupa care sa scriem in toti bitii aferenti segmentelor valoarea 0 pentru a curata display-ul, dupa care asteptam un numar de cicli pentru ca LCD-ul sa se reseteze.

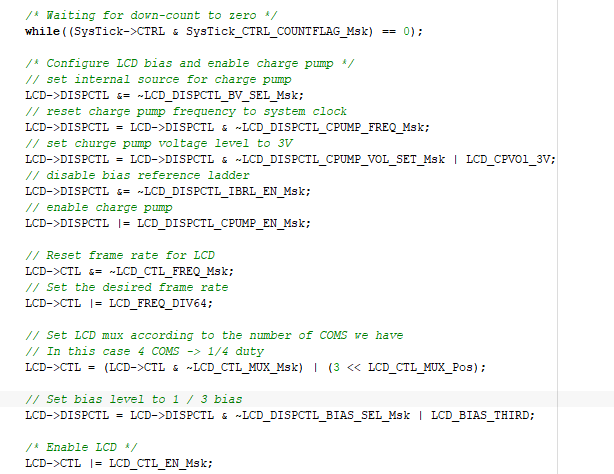
## **Introducerea unui delay pentru sincronizarea operatiilor.**

Este recomandata asteptarea unui interval de timp scurt inainte de a continua operatiile cu display-ul. Pentru delay vom folosi SysTick, clock prezent in cadrul procesoarelor ce implementeaza ARM-v6m.



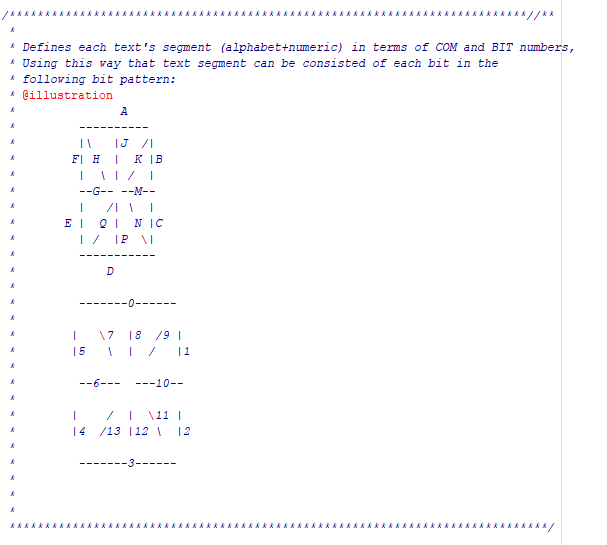
**Configurare LCD.**

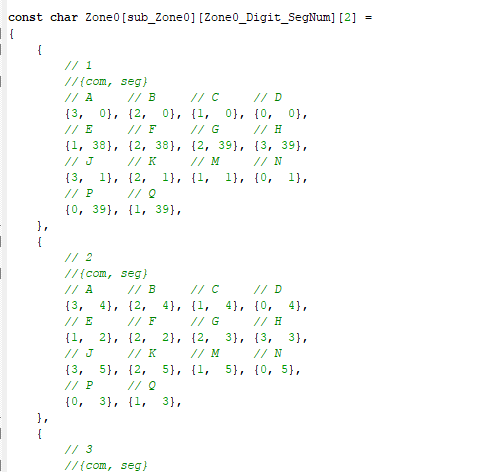
In continuare urmeaza configurarea charge pump-ului pentru LCD, acesta permite aplicarea unui voltaj mai mare decat Vcc pentru o definire mai pronuntata a segmentelor pe LCD, dupa care activam LCD-ul setand bitul de enable din cadrul registrului CTL.

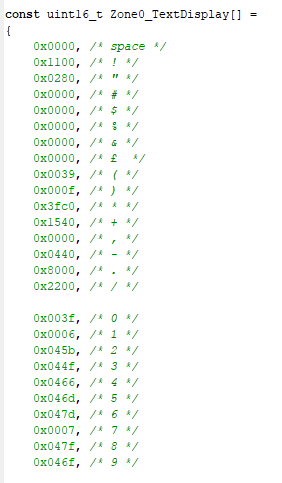


**Afisarea caracterelor pe LCD.**

Pentru afisarea mai usoara a caracterelor pe display, dispunem de un sir de definitii a com-urilor si seg-urilor ce trebuie selectate pentru a afisa un anumit caracter.

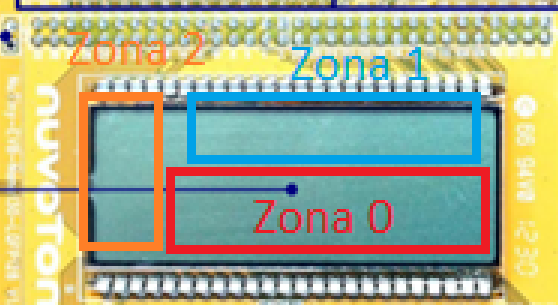






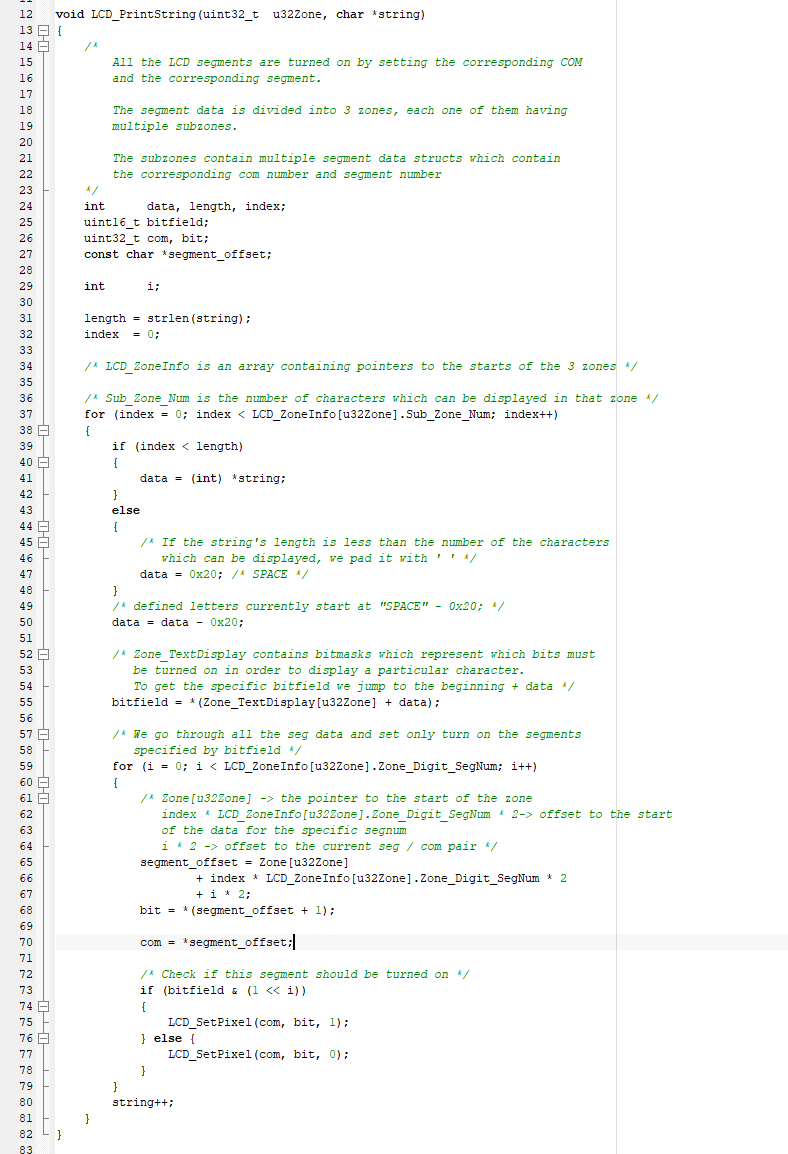
## **Zonele de display**

LCD-ul este impartit in 3 zone, in fiecare zona pot fi afisate anumite caractere / segmente, eu voi afisa text in zona 0.



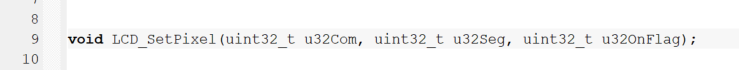
## **Functia pentru afisarea sirului de caractere pe LCD.**

Urmatoarea functie primeste un sir de caractere si citeste pentru fiecare caracter din structurile o masca aferenta segmentelor care trebuie activate si seteaza com-ul si seg-ul respectiv. Aceasta functie apeleaza functia LCD\_SetPixel, pe care o voi implementa in Assembly.



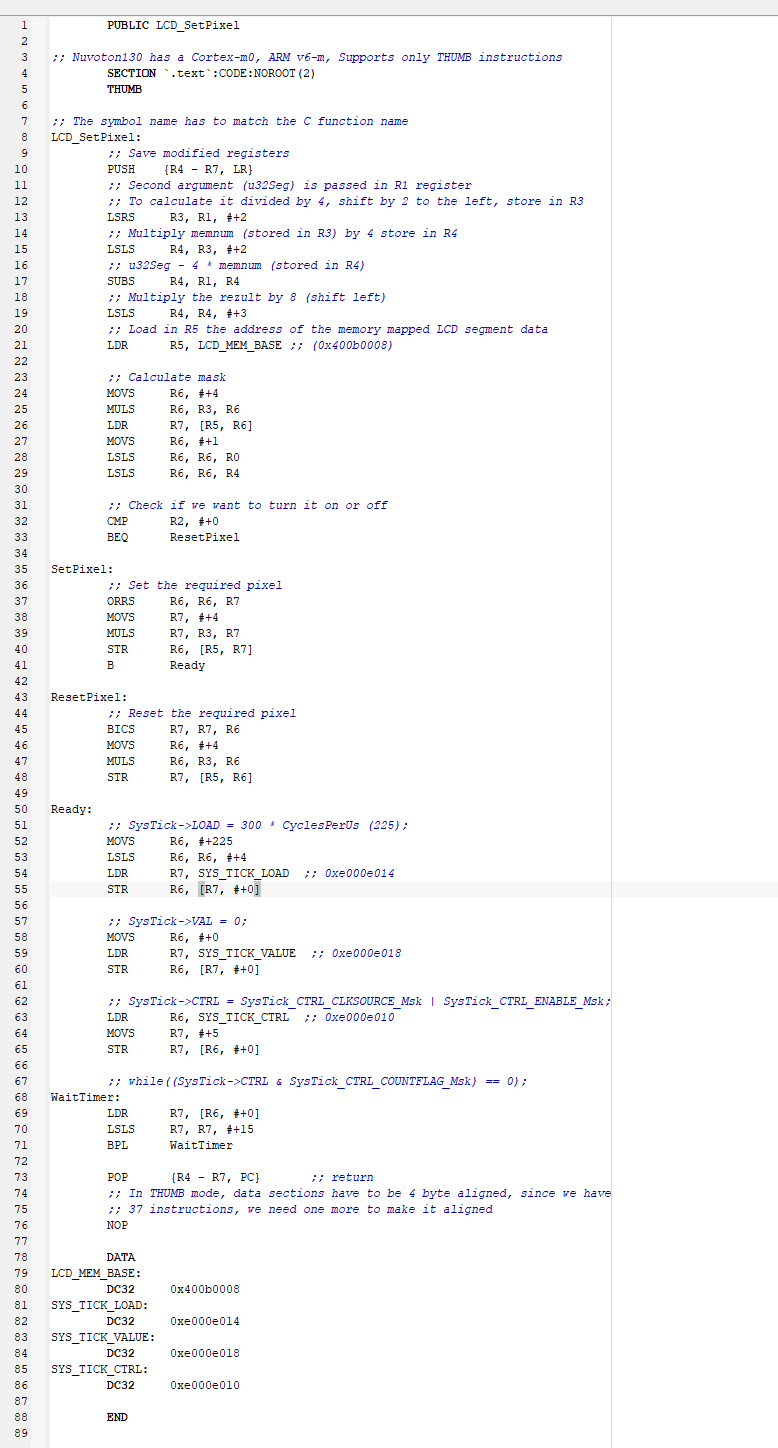
## **Declararea functie LCD\_SetPixel in fisierul sursa C.**

Pentru a putea apela in limbajul C o functie, aceasta trebuie sa fie declarata si marcata cu extern, functiile sunt implicit declarate extern in cazul cand nu sunt marcate cu **static.** In fisierul .asm va trebui declarata o eticheta de tip **PUBLIC** cu aceeasi denumire ca si functia declarata in C.



## **Implementarea functiei LCD\_SetPixel in limbaj de asamblare.**

Pentru scrierea codului in asamblare este necesar sa stim ca toate procesoarele ARM-v6m cu un set de instructiuni RISC, suporta doar setul de instructiune THUMB pe 16 biti, din acest motiv este necesar sa declaram alinierea sectiunii de cod la 2 bytes (**NOROOT(2)**) si activarea setului de instructiuni **THUMB**. Parametrii functiei sunt transmisi prin registrii **R1**, **R2**, **R3**, valoarea de return este scrisa in **R1**, iar la iesirea din functie este necesara scrierea de pe stiva a valorii de return in registrul **PC**. De asemenea, orice sectiune de date trebuie aliniata la 4 bytes, din aceasta cauza, la sfarsitul codului este inserata o instructione **NOP** pentru a avea un numar de 38 de instructiuni, care va fi echivalentul a 38 \* 2 = 76 bytes.



# 

**Concluzii:**

In cazul aplicatiilor embedded este vitala intelegerea detailata a platformei de lucru si a resurselor aferente acesteia. Limbajul C ne expune o interfata simplificata pentru a genera codul pentru sistemul pe care lucram, insa fara o cunoastere buna a codului masina generat este dificil de depanat codul, in special atunci cand o componenta hardware nu produce rezultatele dorite. Este uneori necesar sa scriem unele parti ale proiectului in limbaj de asamblare pentru a obtine acces la resursele low-level ale sistemului. Pentru a putea folosi aceste componente, este nevoie de o intelegere buna a modului in care compilatorul de C genereaza codul in asamblare si cum putem interschimba cu usurinta aceste 2 limbaje.

**Referinte**:

ARMv6-M Architecture Reference Manual

NuMicro Family Nano 100 Series Datasheet

<https://github.com/PetricaP/ProiectSOC/references>

<https://en.wikipedia.org/wiki/Liquid-crystal_display>

<https://en.wikipedia.org/wiki/Charge_pump>

<https://www.pacificdisplay.com/lcd_multiplex_drive.htm>