

1. Preparations. What do we need?
 - a) first of all we need the main file (the executable file) of the game we want to hack. It has no extension and it is found inside the 'game.app' folder on your ideoice. you can also get it from the .ipa package (open it with winrar)
 - b) ida pro v5.5
 - c) a hex-editor (i use hex workshop)
2. Let's begin!

PART 1

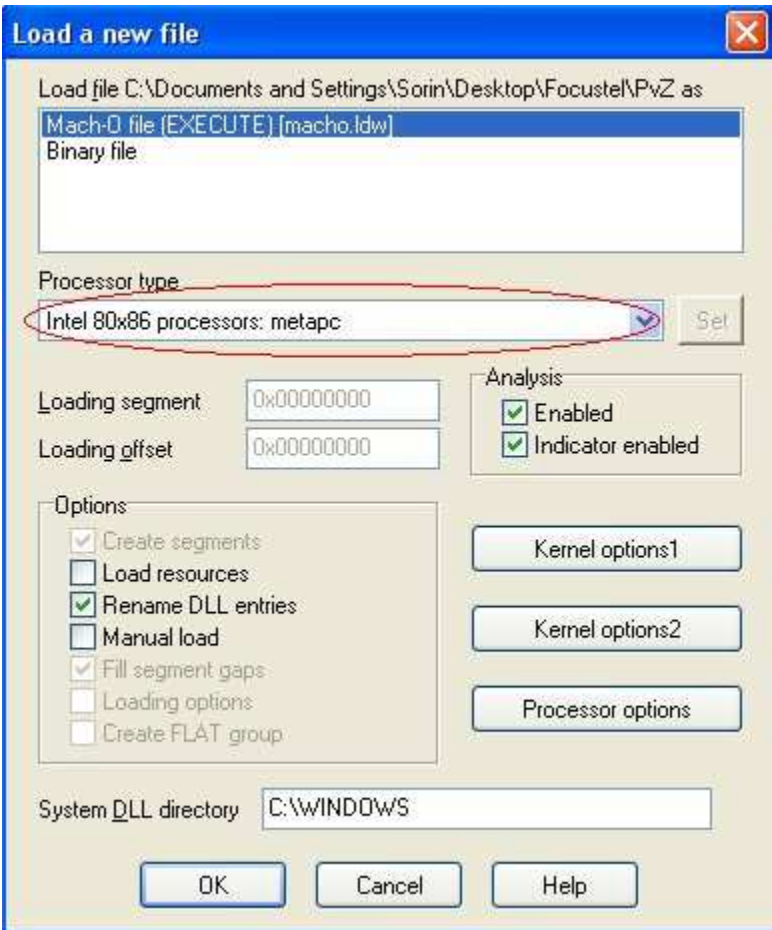
first i'm going to show you how to do a hack when you get an 'error' message like 'not enough money' etc

HACKING PLANTS vs ZOMBIES v1.0

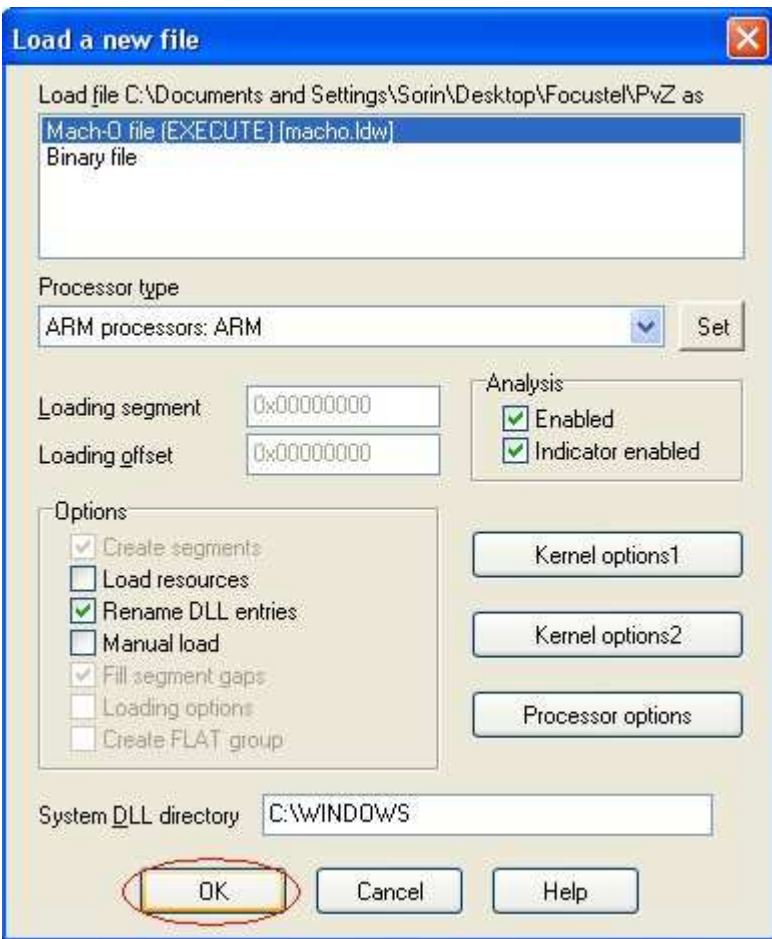
ok. play the game a little until you arrive at crazy dave's shop. try to buy something that you don't have money for. the following screen should appear:



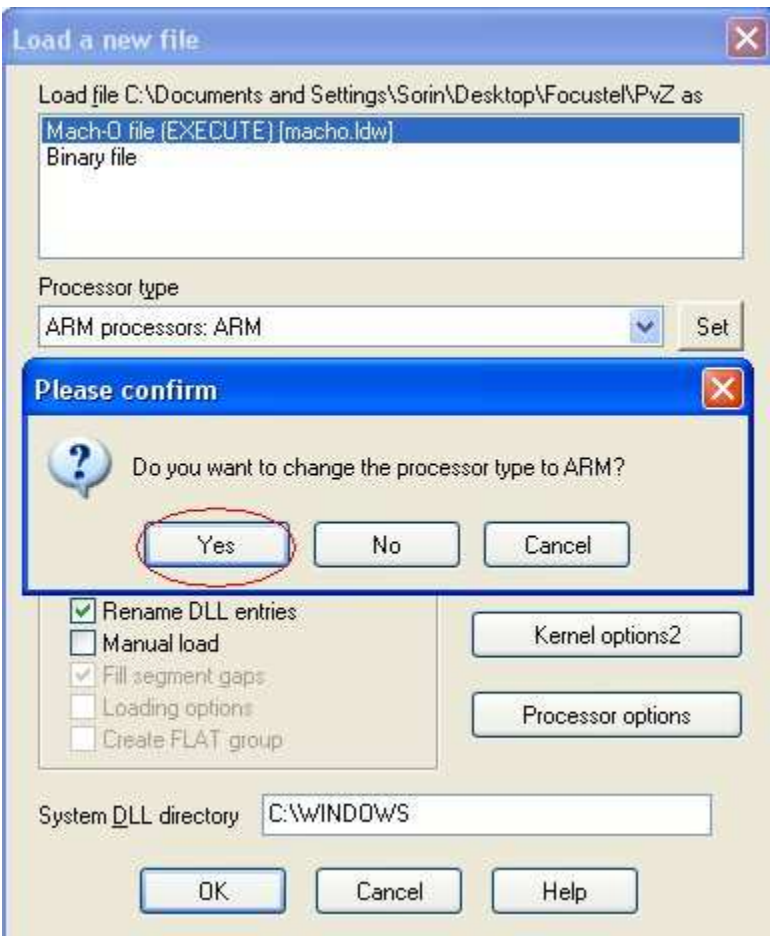
write down on a piece of paper that message or just remember it. now DRAG&DROP the 'PvZ' file into IDA shortcut on your desktop. the following dialog box should appear:



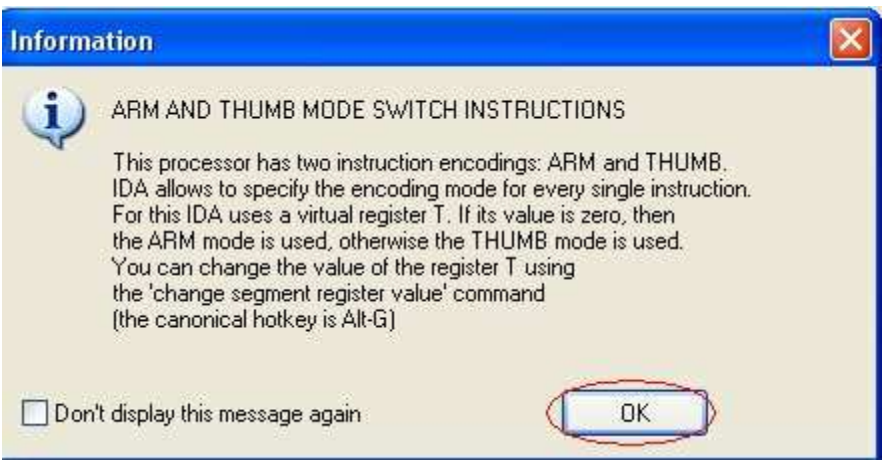
just choose 'ARM' as processor type from the drop-down list and click ok



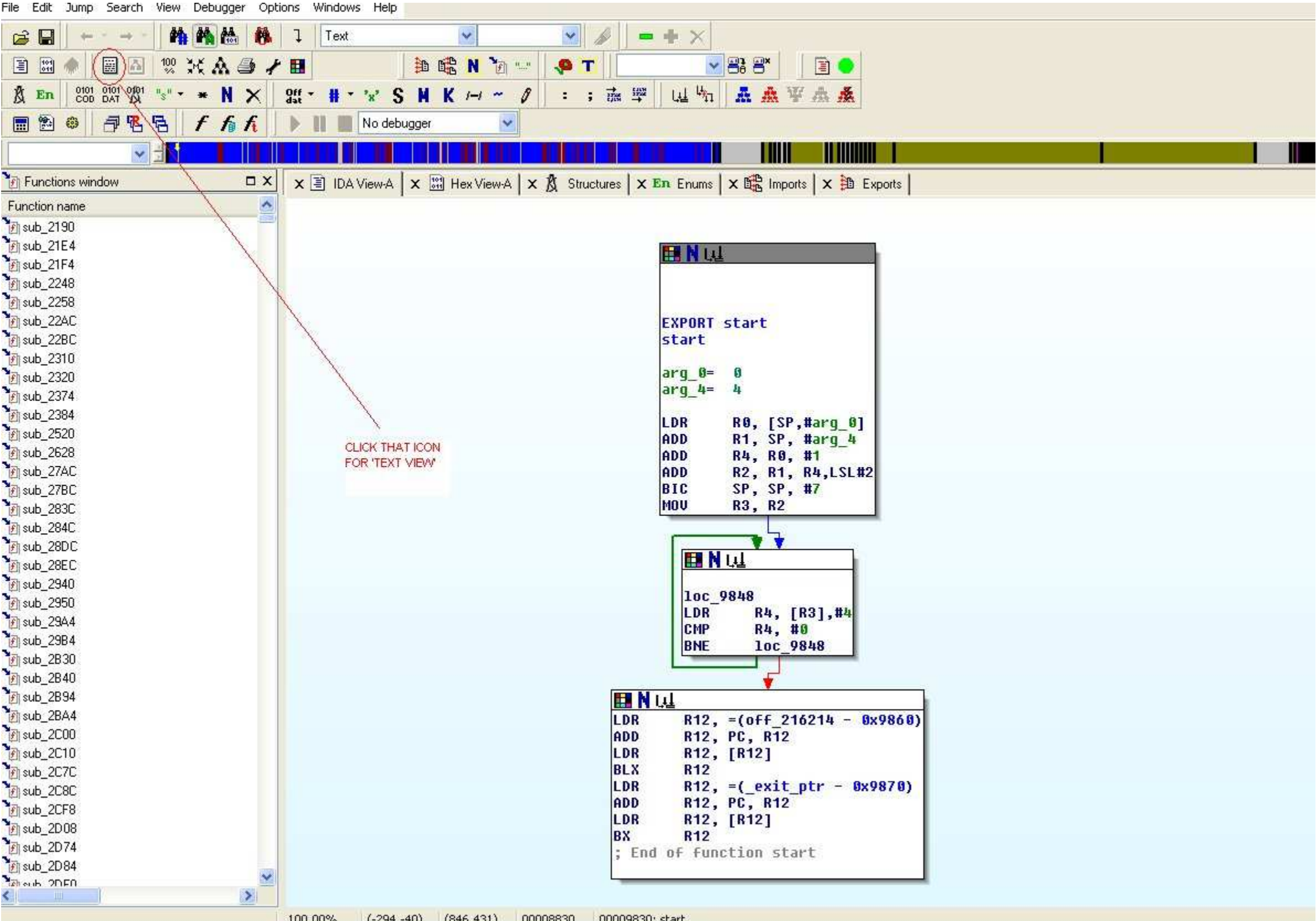
a message should appear that tells you if it's ok to change the processor type to arm. just press ok



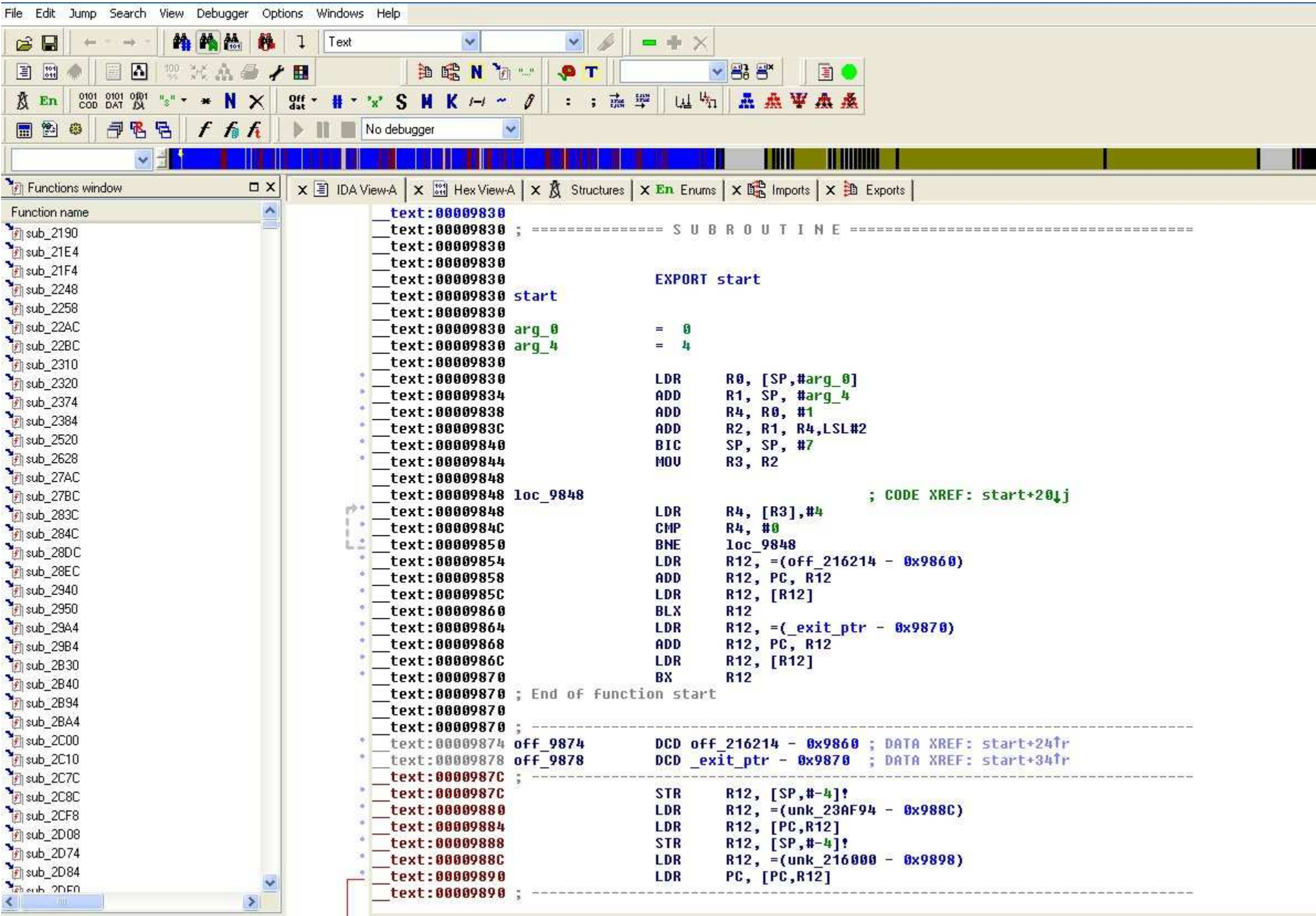
IDA will now be loading the 'PvZ' file. a message box will appear telling you something about 'thumb mode' and 'arm mode'. just click ok



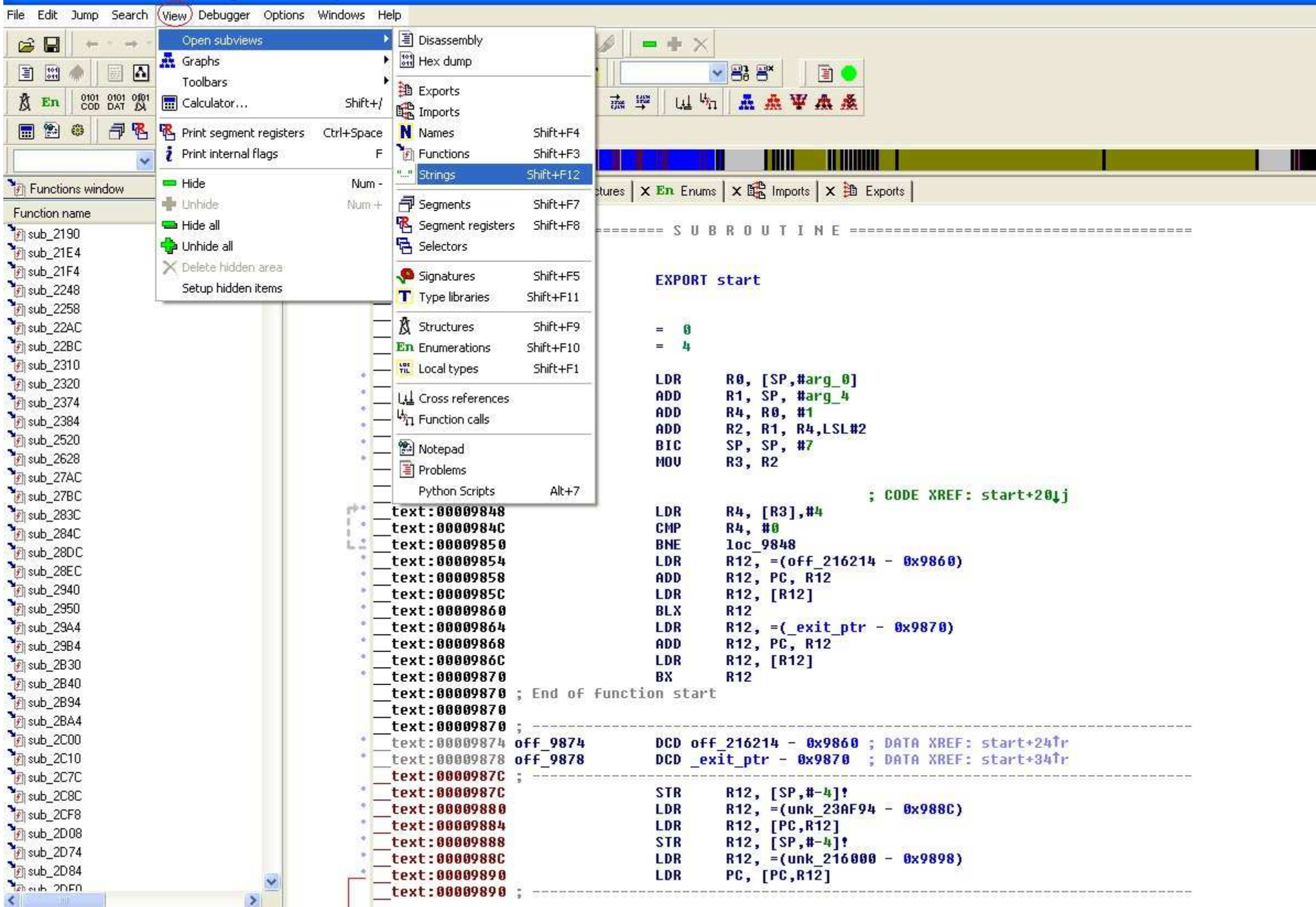
ok. now you'll have to wait until IDA finishes analysing the file. it's about 5 minutes or maybe less. when IDA finishes, the following screen should appear:



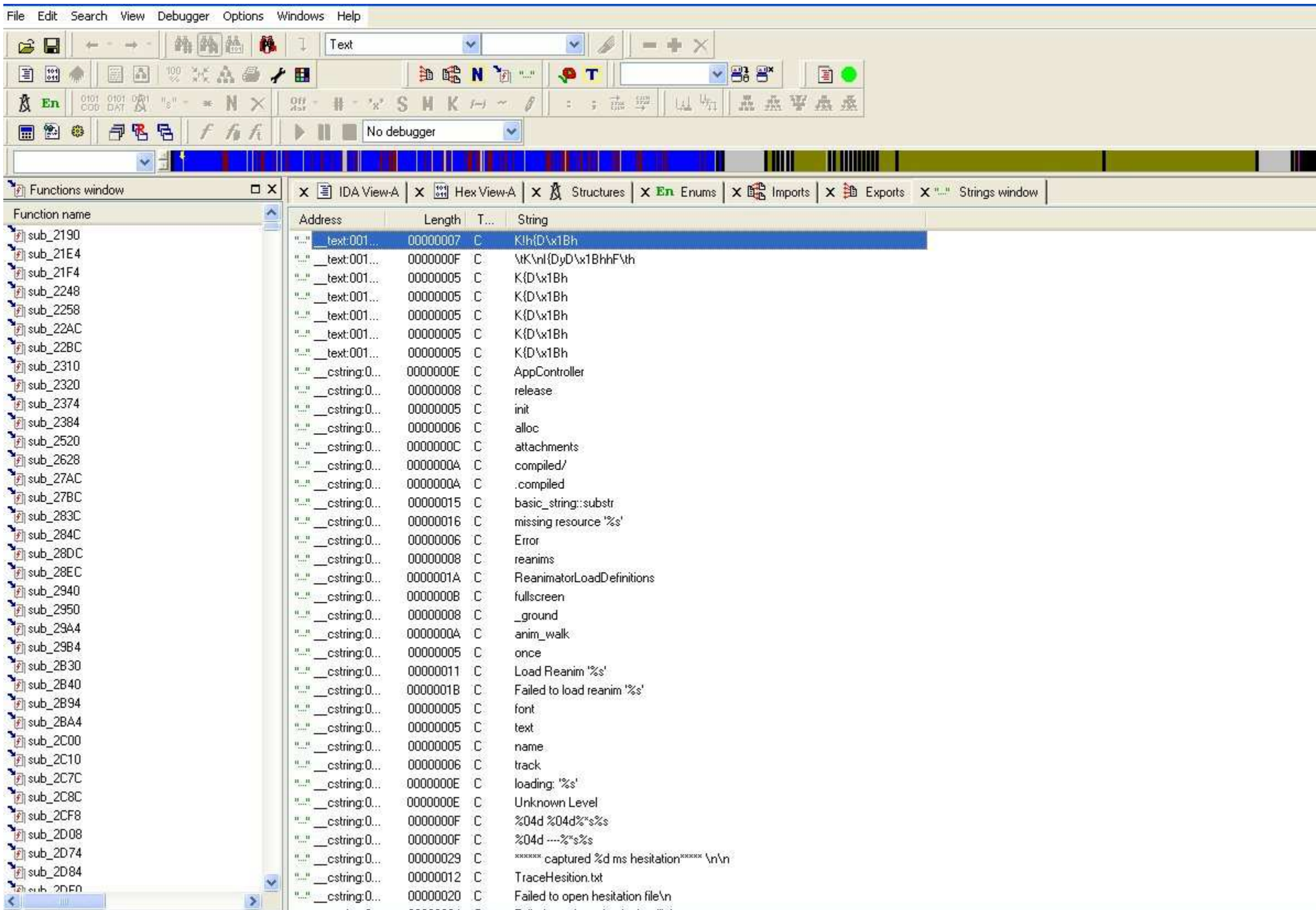
now click on 'TEXT VIEW' like in the above picture. this is for switching from 'graph view' to 'text view'. you should see the following:



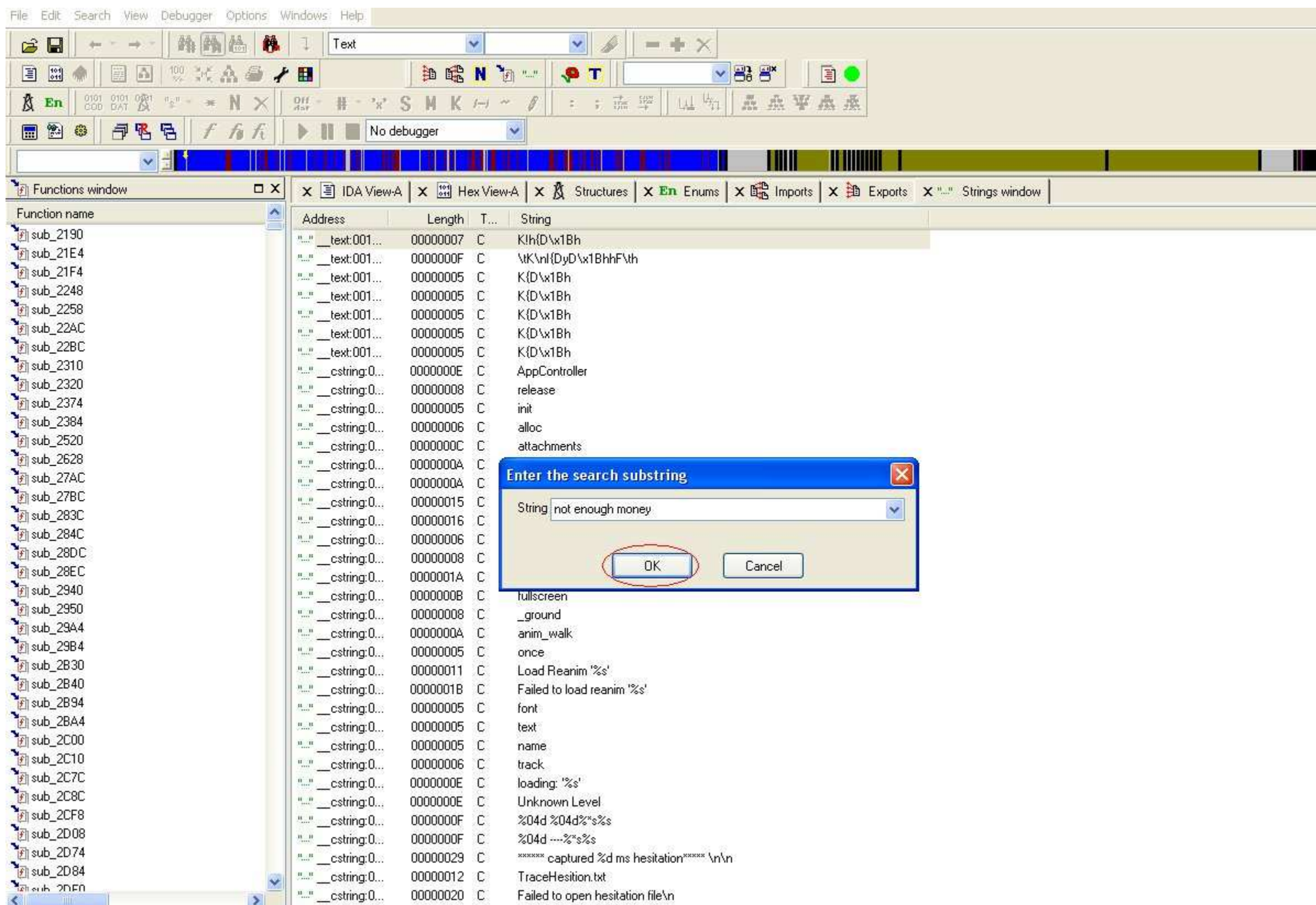
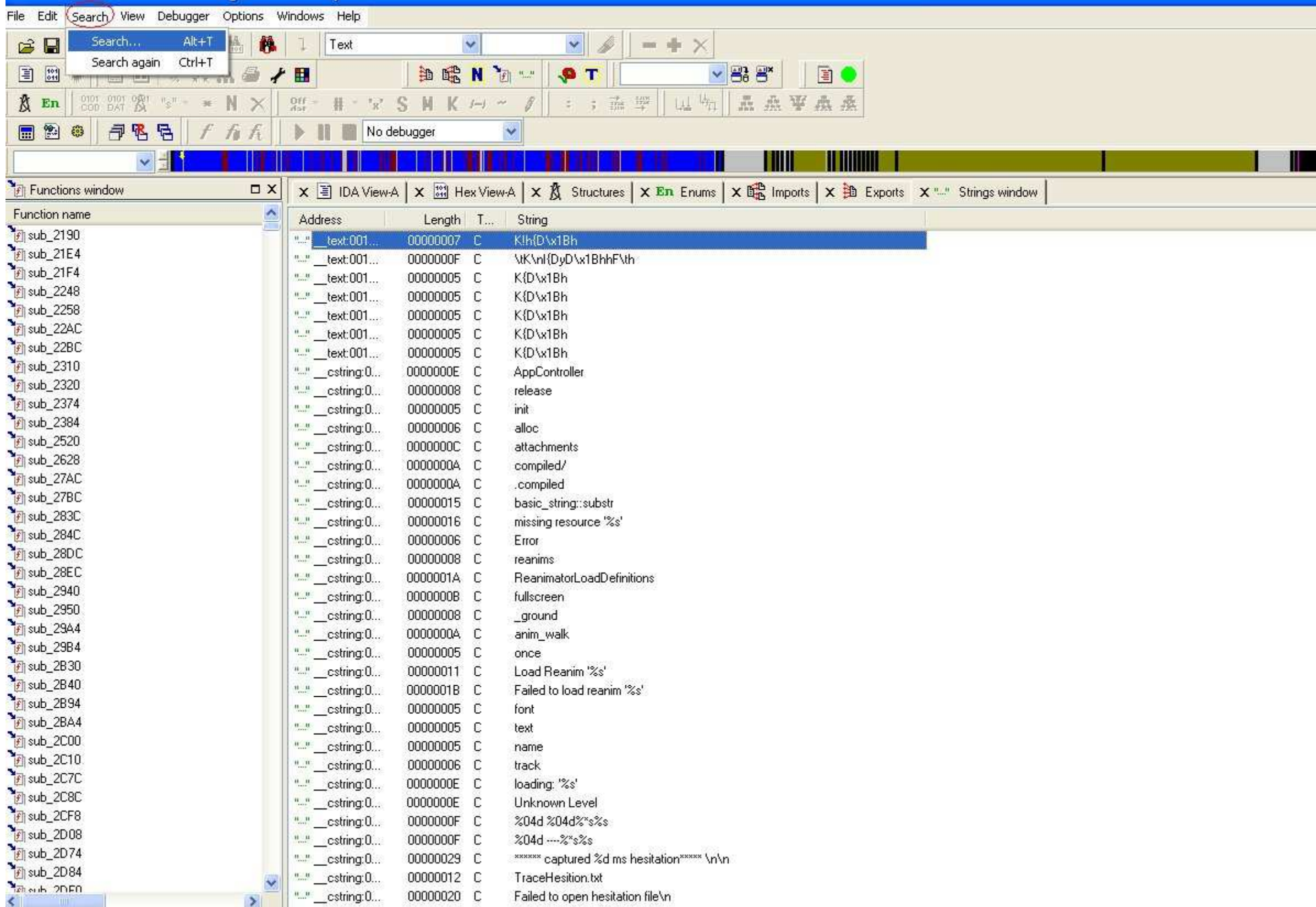
now we can begin hacking our game. we need to track down a 'not enough money' message, so open the STRINGS subview in ida: View->Open subview->Strings (or by pressing SHIFT+F12). here's the picture:

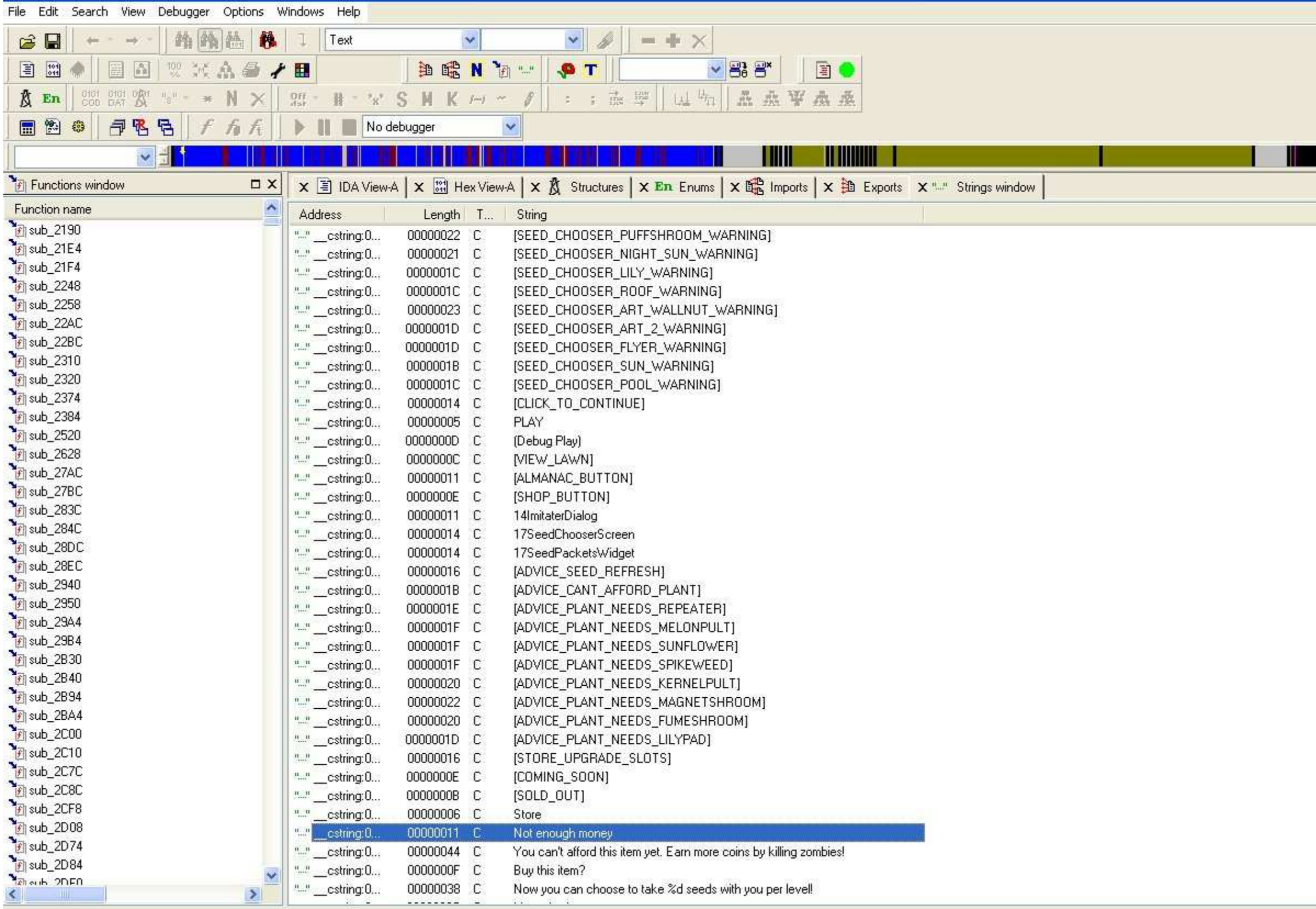


now you are in the STRINGS subview. all the game strings(messages, texts, errors etc) are displayed here. here's the picture:

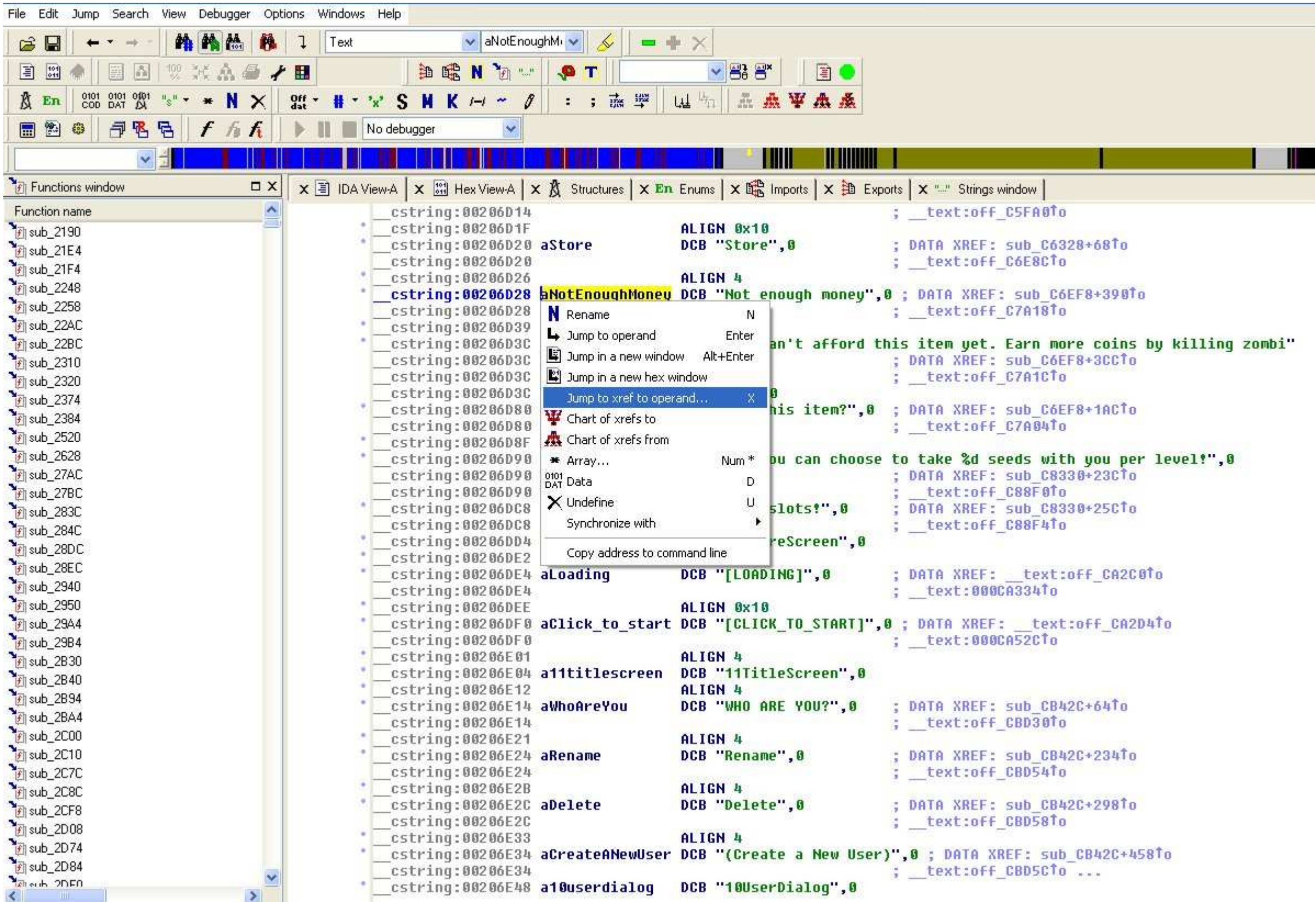


ok. we want to look for our 'error' message with 'not enough money'. so click on Search->Search... (or press ALT+T). here's the picture:

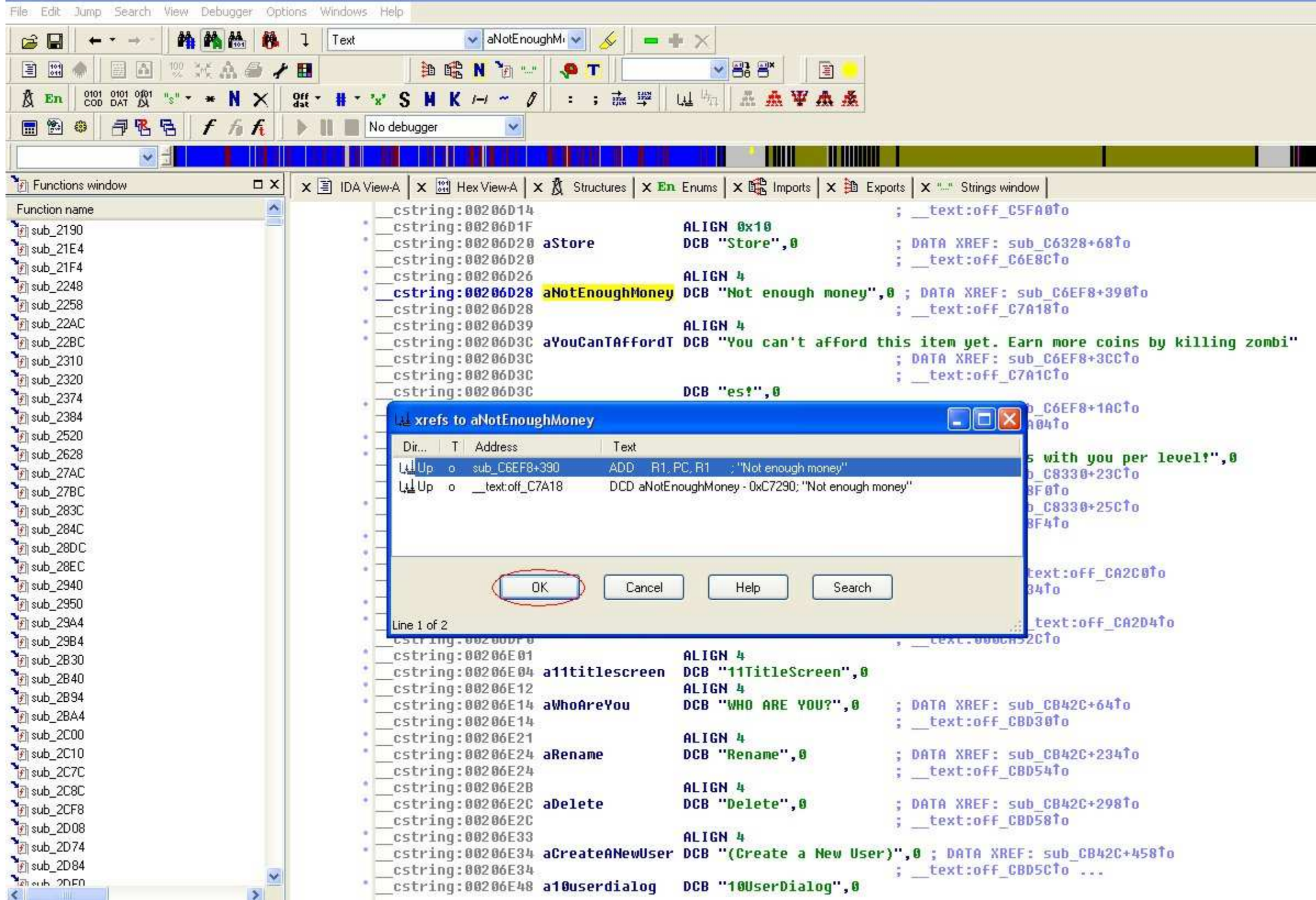




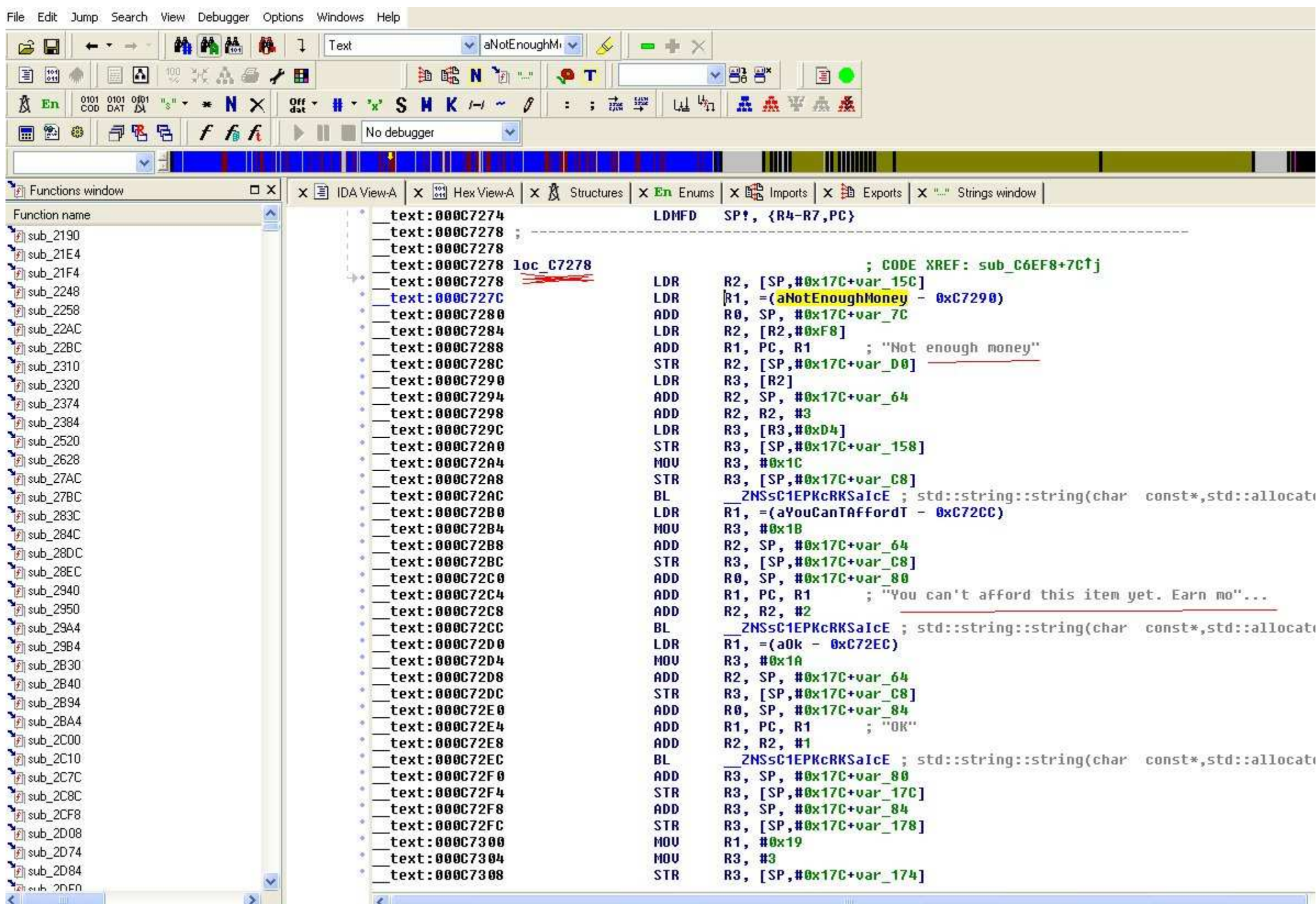
now we just double click on it and we will be here:



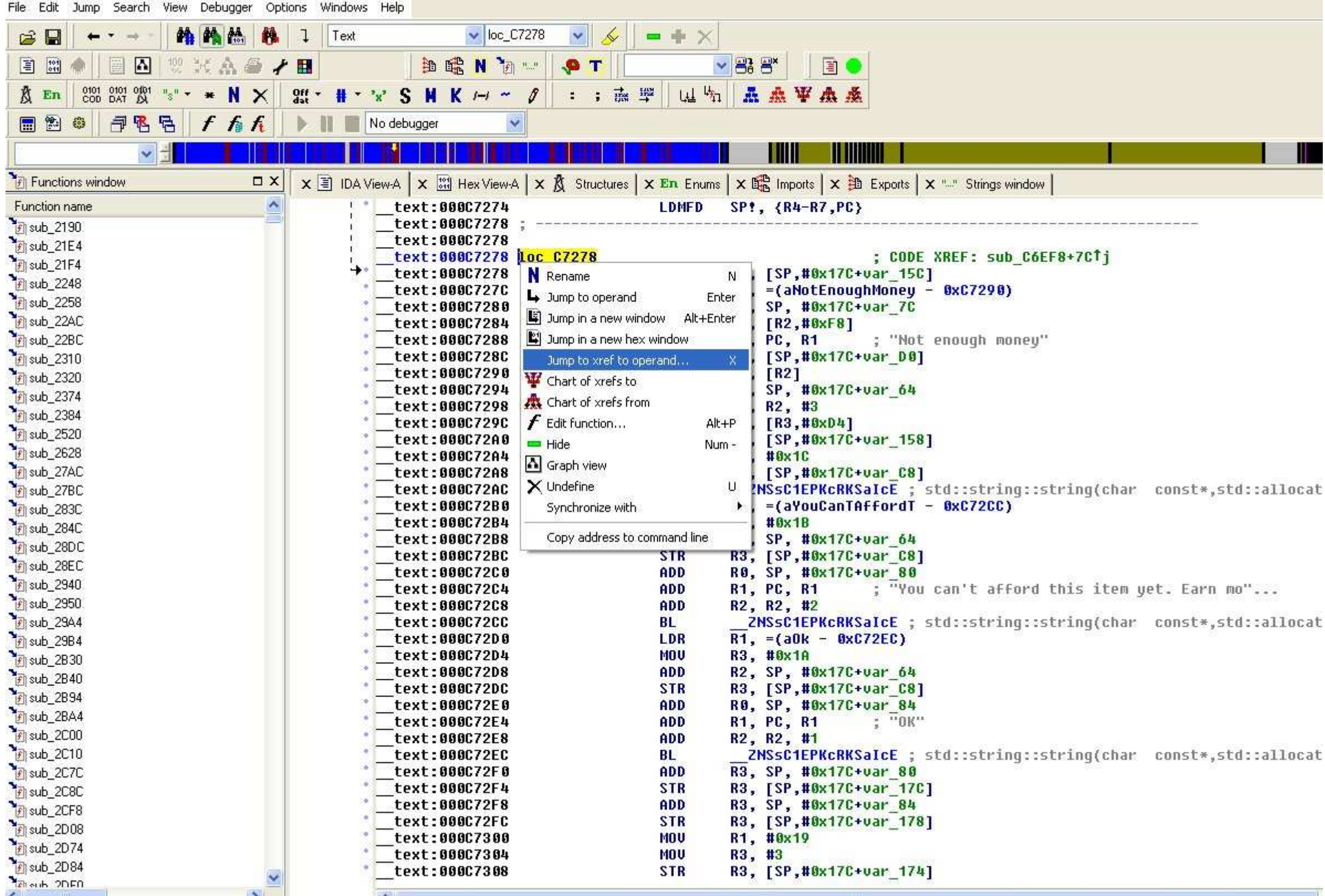
the 'aNotEnoughMoney' is the operand containing the 'not enough money' string. let's see where it's used in the program. to see this, we right-click on it and select 'jump to xref to operand' (or click on it and press 'X'). the following dialog should appear:



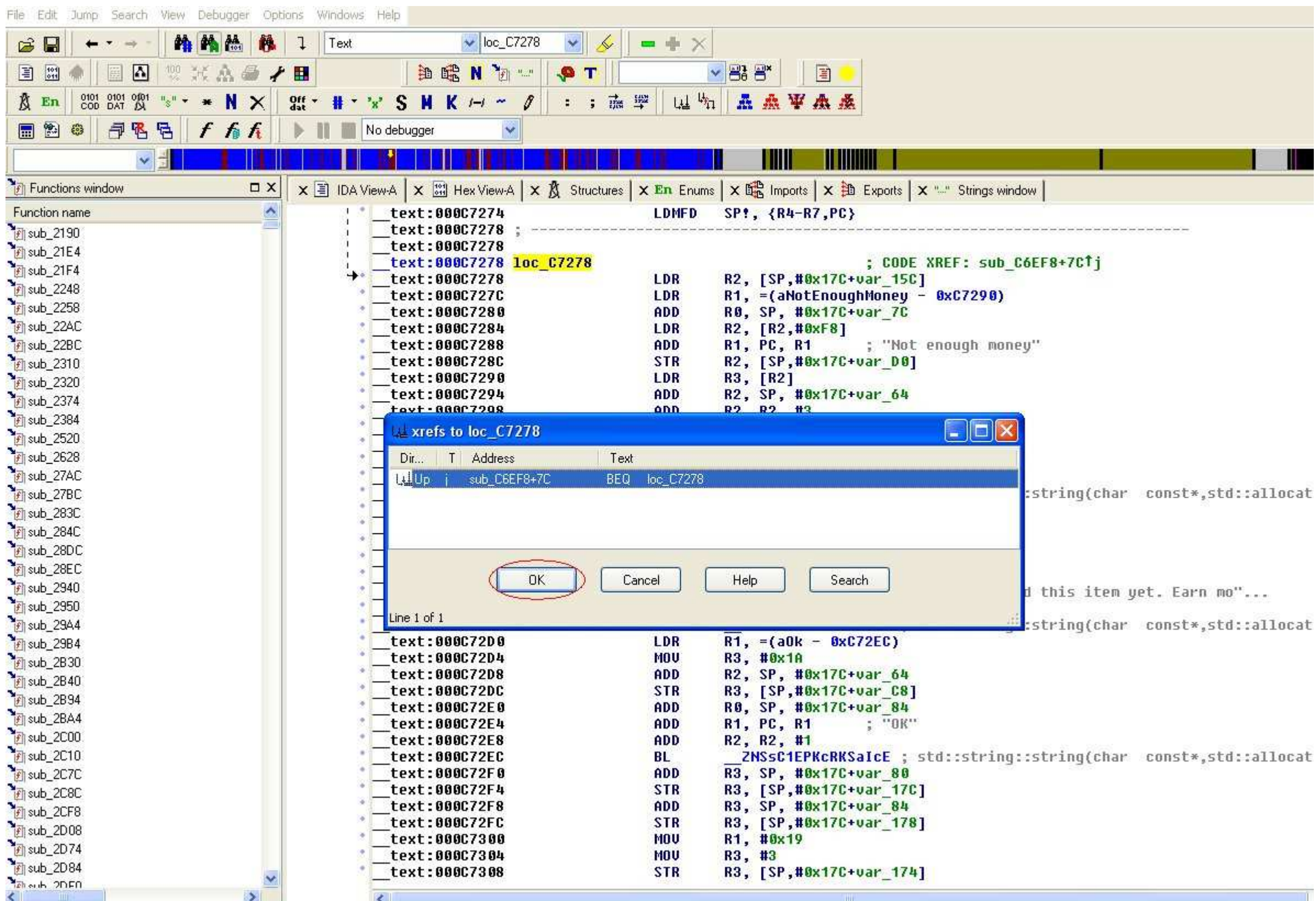
now click ok having the 'sub_C6EF8+390' line highlighted, you will be here:



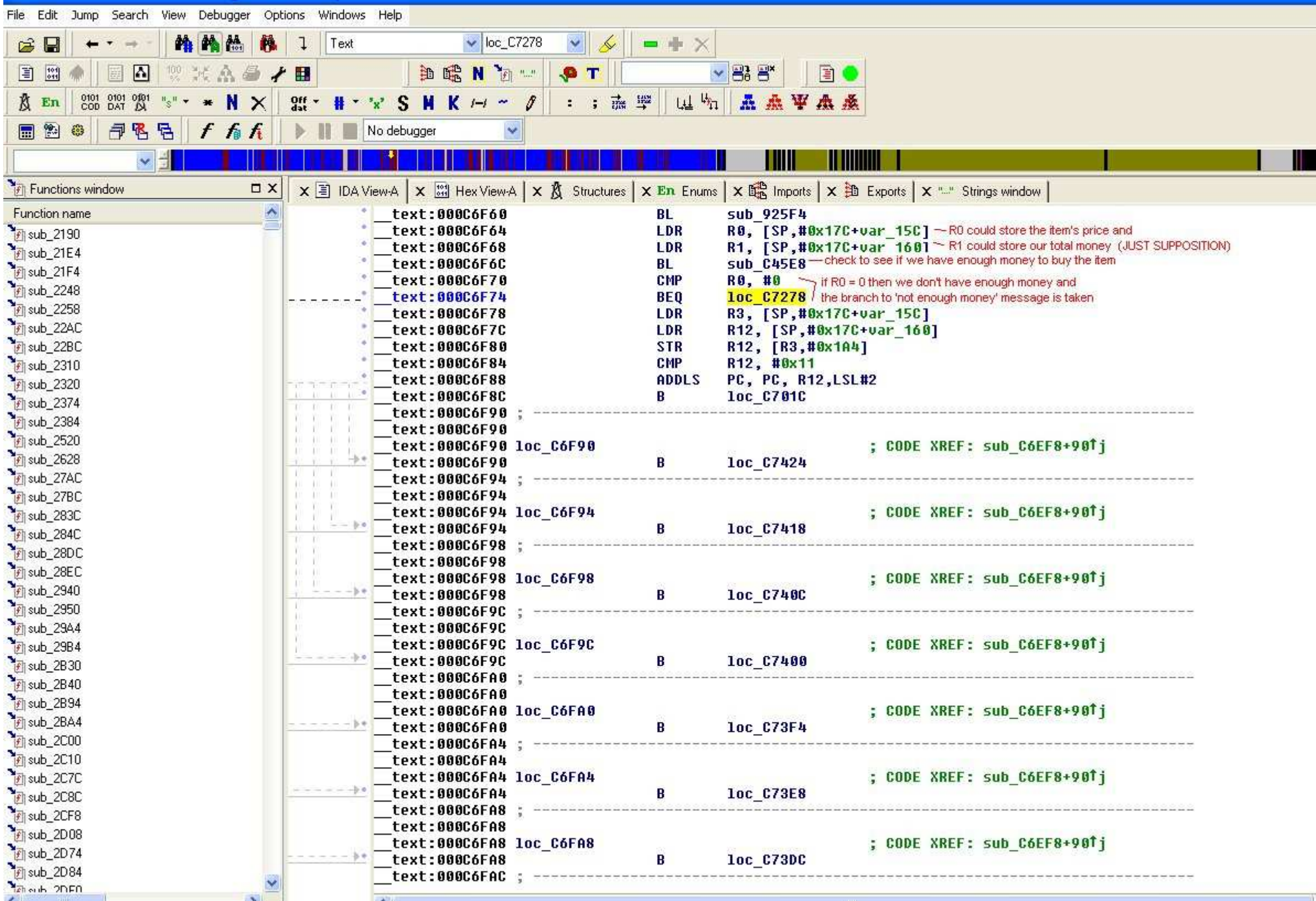
after you can see the 'not enough money' dialog message is build here (containing the 'you can't afford this item yet...' and the 'OK' button of the message dialog) we want to find where the game calls this 'not enough money'. if you look more carefully, you can see we have cross references for location 'loc_C7278' which is the beginning of our 'not enough money' message dialog. so let's see who calls this 'not enough money' routine. right click on 'loc_C7278' and choose 'jump to xref to operand' (or click it and press 'X'). here's the picture:



the followind should appear:



now just click ok, and the following should appear:



now comes the tricky and funny part. you must interpret the code. and you must also have some basic knowledge of ASM/ARM
in ARM we'll work with registers. they are R0,R1,R2...R12.

R0 is the most common register that returns a value from a function. let's say we have the function 'CreatePlayer'. after the function is executed, R0 could hold 1 or 0 (true or false). if it's 1 then the player has successfully been created. if it's 0 something went wrong. another example: function IsMissionUnlocked. if this function returns 1 in R0, then the mission is unlocked.

there are also instructions like 'mov R2, R4'. well, as it's name suggest, the value in R4 is moved(copied) to R2.

the B instruction stands for 'branch'. it's the equivalent of 'JMP' instruction in ASM.

the BL stands for 'branch with link'. it's the equivalent of 'call' instruction in ASM. so 'BL sub_C45E8' will call the function located at 'C45E8'

the CMP stands for 'compare'. so the code

CMP R0, #0

BEQ loc_C45E8

can be interpreted as: IF R0 = 0 then go to location C45E8

(the BEQ stands for 'branch if equal')

ok. so now we know that if R0=0 the 'not enough money' message is showed. what can we do about it? well, we can wipe out these 2 instructions: the CMP R0,#0 and BEQ loc_C45E8. how to do this??? we can't just delete the instructions...

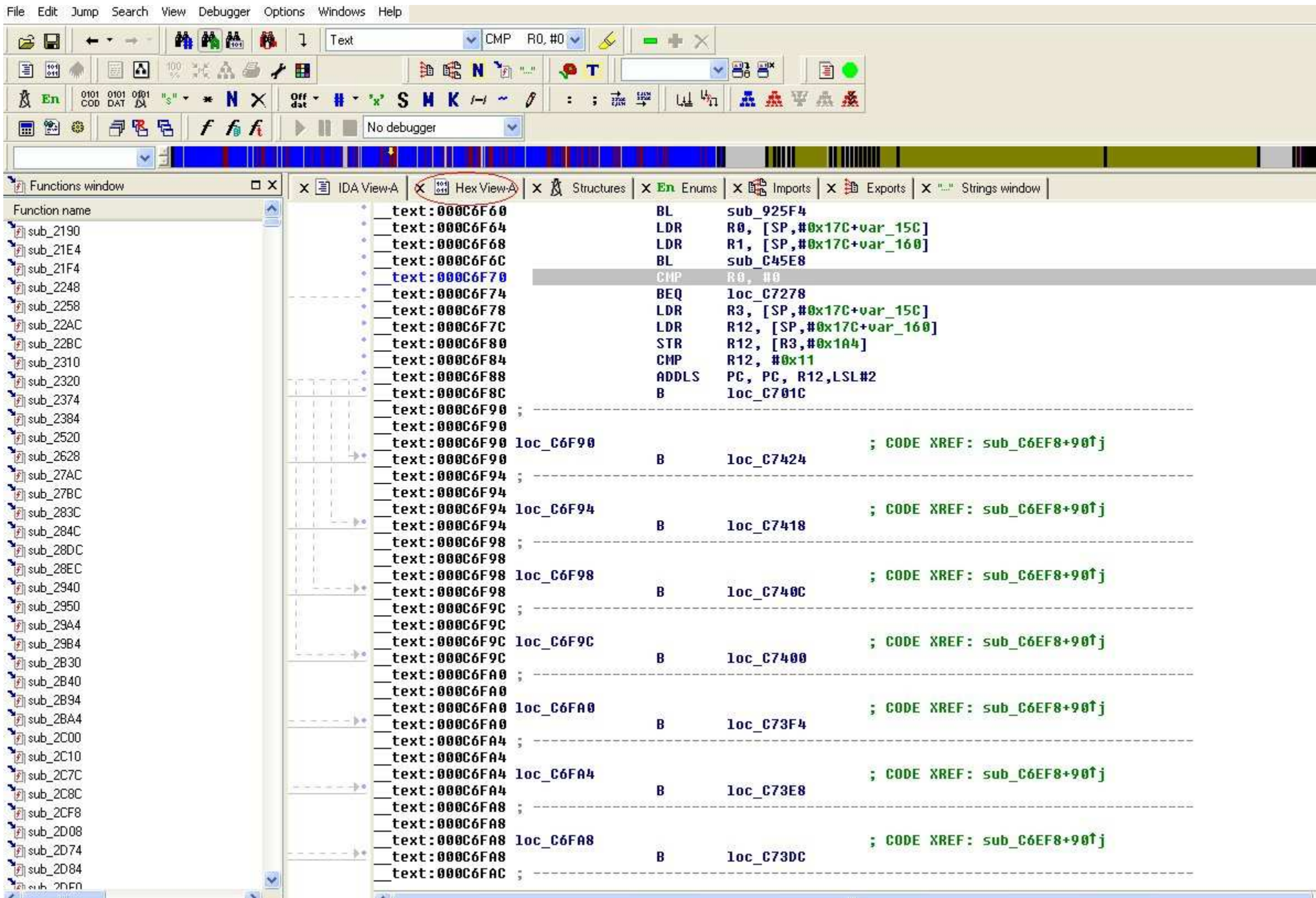
well, in ARM as well as ASM, there is one instruction that does nothing. it's called NOP (no operation)

so all we have to do is to overwrite the 'CMP R0, #0' with 'NOP' and the 'BEQ loc_C45E8' with another NOP.

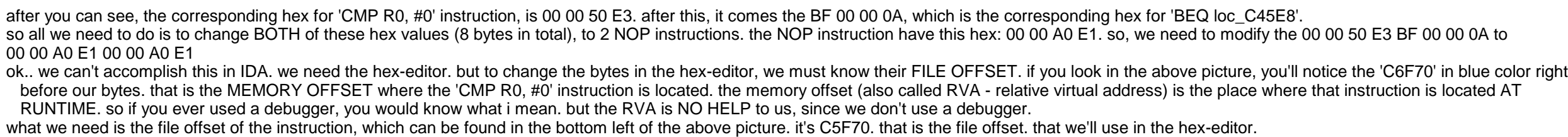
unfortunately, we can't do this in IDA. that's why we'll use a hex-editor. but there's one little problem: in the hex-editor, we can't just write ARM instructions. all we can do is modify the HEX that stands for the ARM instructions.

to see the corresponding HEX values of ARM instructions, we can switch to HEX-VIEW in IDA.

let's highlight the 'CMP R0, #0' and push the 'Hex View-A' button in IDA. here's the picture:



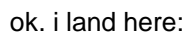
ok. here's what we got:

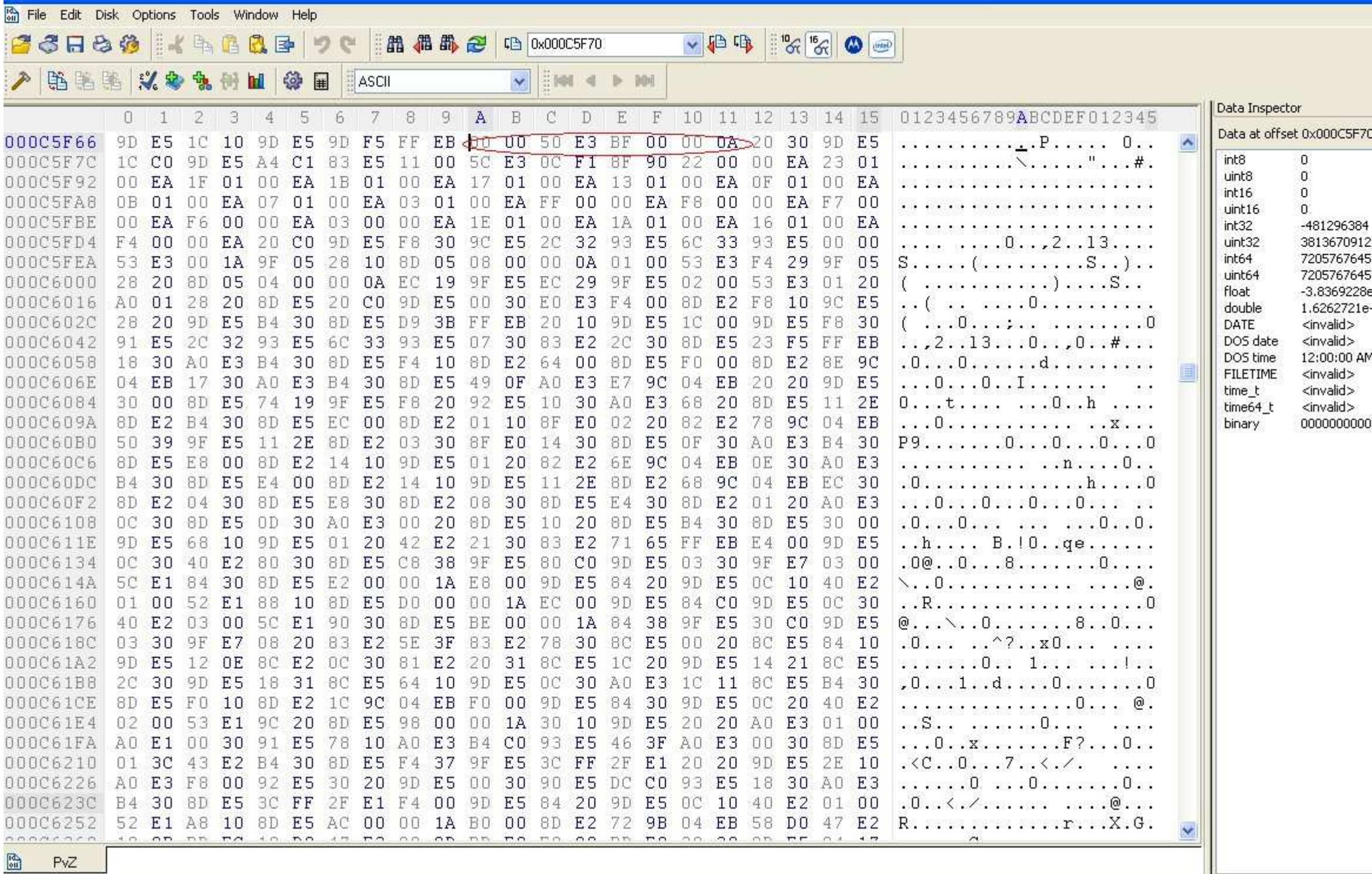


ok. so the summary is this: we know the file offset (C5F70), the bytes of that offset (00 00 50 E3 BF 00 00 0A) which represent the instructions `CMP R0, #0` ; `BEQ loc_C45E8`, and we need to change these bytes to 2 NOP instructions (00 00 A0 E1 00 00 A0 E1). By doing this, the game will never show us the 'not enough money' message.

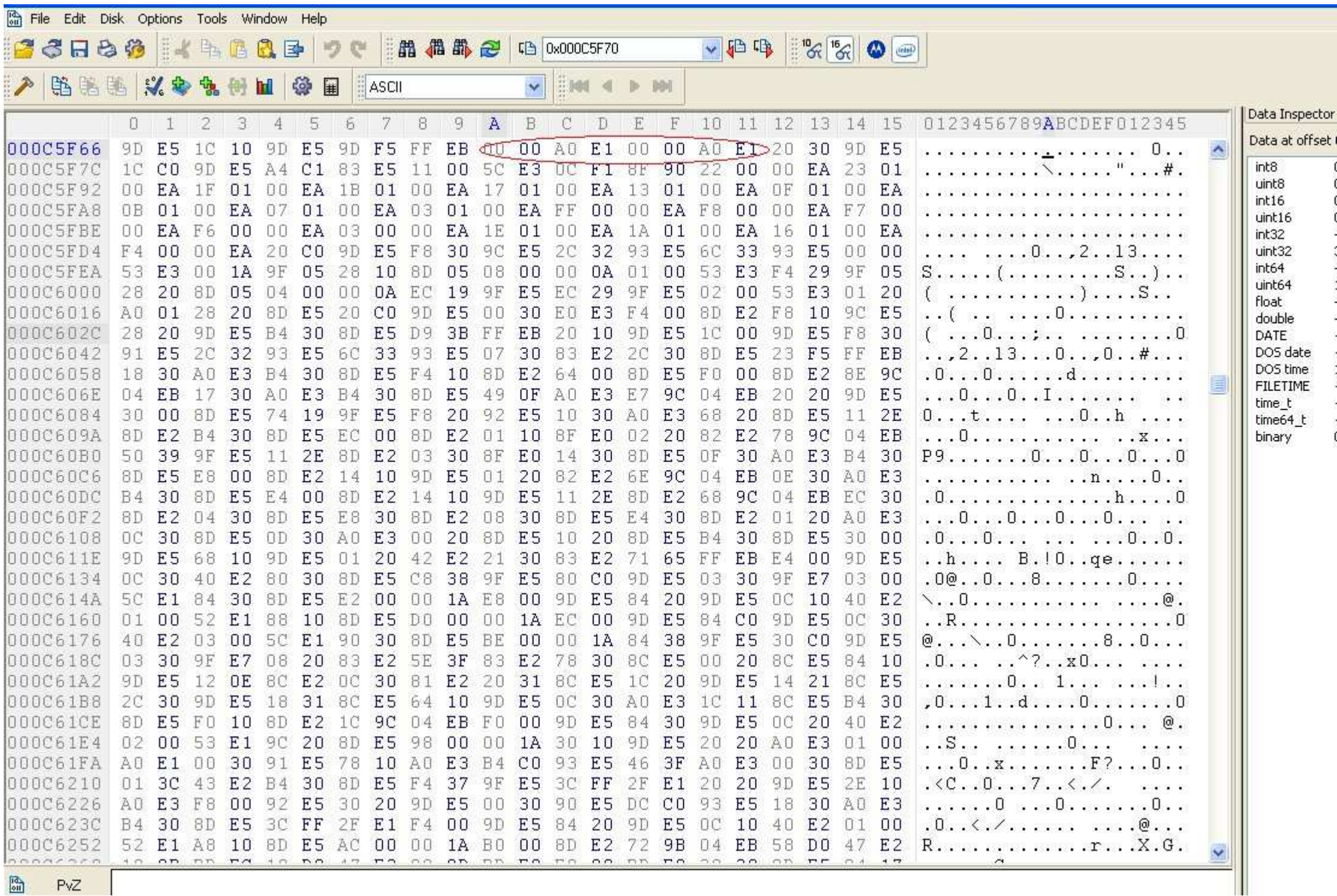
like i said before, i'm using HEX WORKSHOP as my hex-editor

so i open the 'PvZ' file in hex workshop, i press CTRL+G (go to offset), and i type the offset we took from IDA (C5F70). here's the picture:





right at the bytes we need to change. just change the '00 00 50 E3 BF 00 00 0A' to '00 00 A0 E1 00 00 A0 E1' and save the file



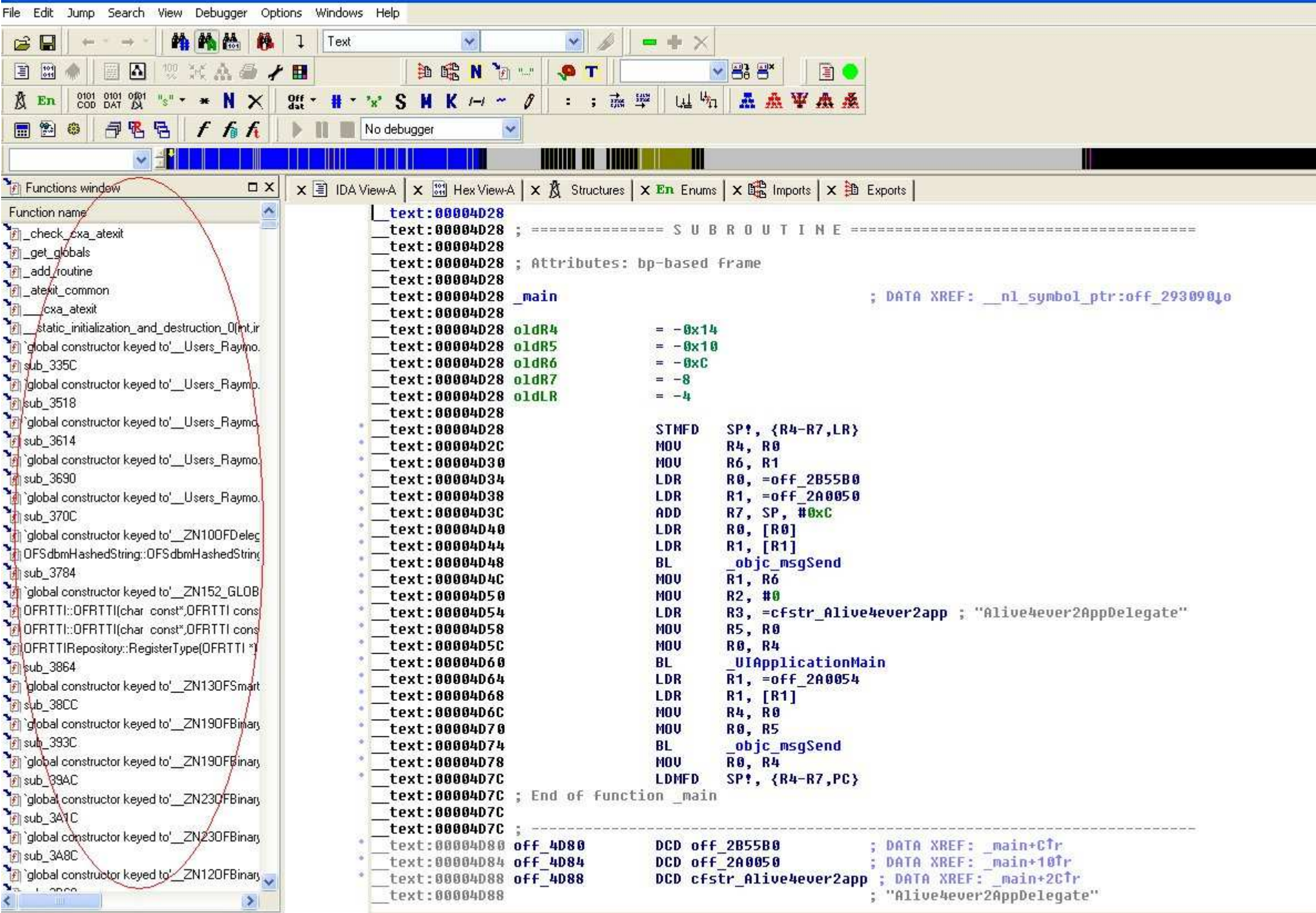
now all that it's left is to put back the 'PvZ' file on your idevice, chmod to 775 and sign it. (check <http://iphonecheating.blogspot.com/2010/04/guide-about-installing-hacks.html> if you don't know how) and, of course, enjoy your first HACKED GAME

PART 2

in this part i'll show you how to look up for the game functions responsible for different things that you're interested: for example [the routine that manipulates enemies' HP in Alive 4](#) ever returns

HACKING ALIVE 4-EVER RETURNS v1.1.2

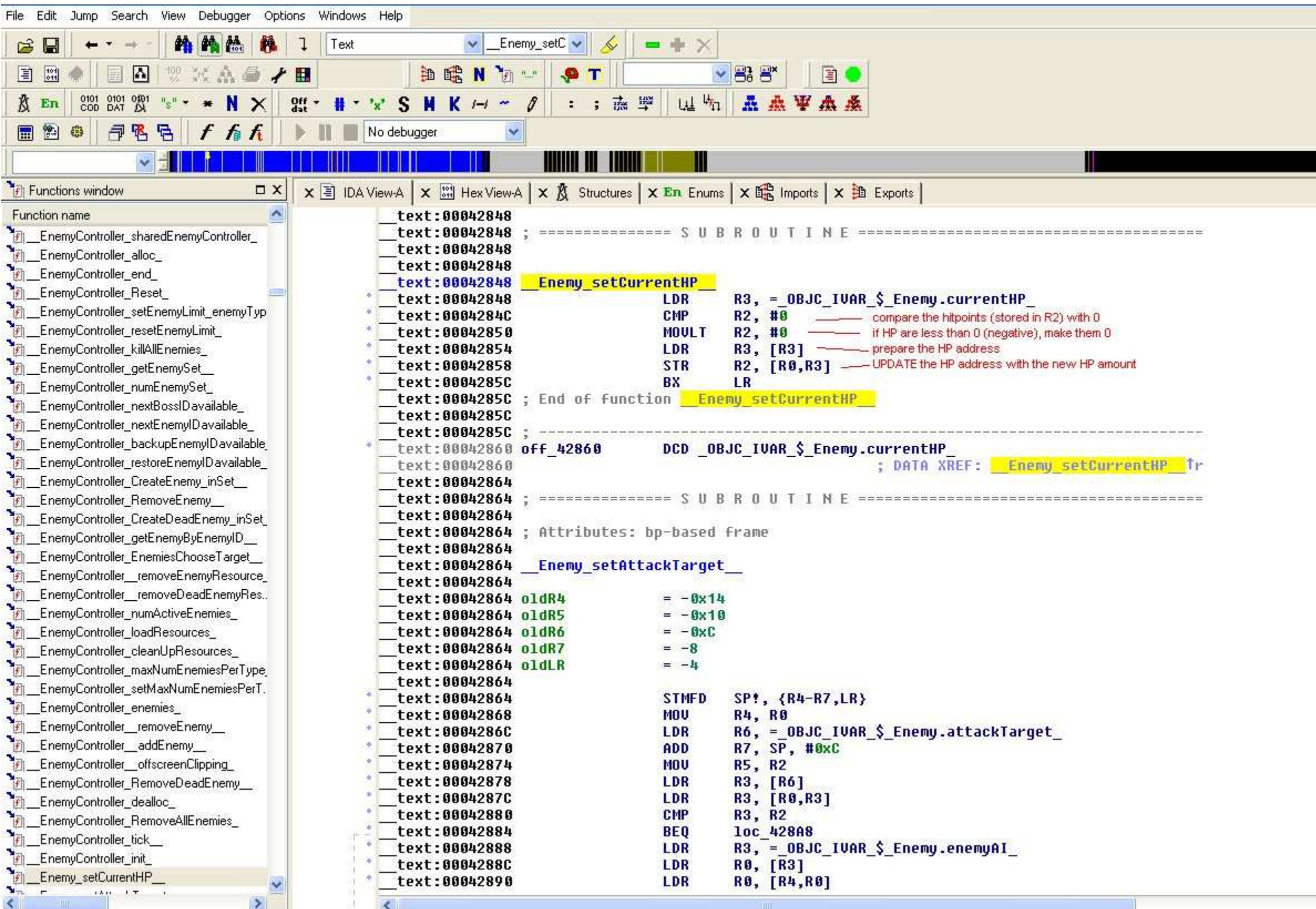
first of all, load up the 'A4ER' in IDA. use the above method learned with plants vs zombies example. wait until IDA finishes the analysis, switch to text mode view.



ok... in the left most part of the screen is the 'Functions window'. all the functions/procedures used by the game are displayed there. in most cases, the names of the functions the game developers used are displayed there. for example if the producers of the game used in their source code the function 'GetPlayerHitPoints', then the function responsible for getting player's hit points will have the same name when IDA disassembles the file(in most cases). that's because Objective C is the standard for iphone applications. but how can this help us?? simple: just use your imagination/intuition. you must search for keywords that interests you. here's a list you may give a try on every game: unlock, locked, hitpoints, HP, life, kill, mana, MP, money, gold, player, enemy, speed etc. just use your imagination. if you want to search for character's speed, then the keyword is 'speed'. now, we will search for HP. so click inside 'Functions window' in IDA, press 'ALT+T' and type 'HP' for search keyword and press OK. the first function that contains the 'HP' string is '___MTouchEvent_touchPosition_'. this does not involve hitpoints, so we press CTRL+T to search next. IDA should now find: '___MTouchEvent_setTouchPosition_'. not interesting, search next! ___MTouchEvent_setTouchPosition_ not interesting search next ___MTouchEvent_touchPhase_ search next ___MTouchEvent_setTouchPhase_ search next

..... search next until you find this function: ___Enemy_setCurrentHP_

ok.. double click on it, and IDA will take you to the function. here's the picture:



ok, now to interpret the code:

```
___text:0004284C    CMP    R2, #0
___text:00042850    MOVL   R2, #0
___text:00042854    LDR    R3, [R3]
___text:00042858    STR    R2, [R0,R3]
```

ok. the name of the function tells us that this function sets the current hp of enemies. the function may be called when the enemy is created, and the game sets his HP. also the function may be called when we shoot an enemy and the game must update his HP. i assumed that the new HP of enemies is held in R2. the game compares R2 to 0. and if it's below 0, it puts 0 in R2. so, basically, if we damage an enemy that had HP=4 and the damage amount was 10, the R2 should hold the value 4-10=-6. -6 is less than 0, so the enemy hp is set to 0 (the movlt instruction means 'mov if lower than') the 'LDR R3, [R3]' prepares the HP address, and the 'STR R2, [R0,R3]' will update the enemy HP with the new value after he had taken damage. (str means 'store')

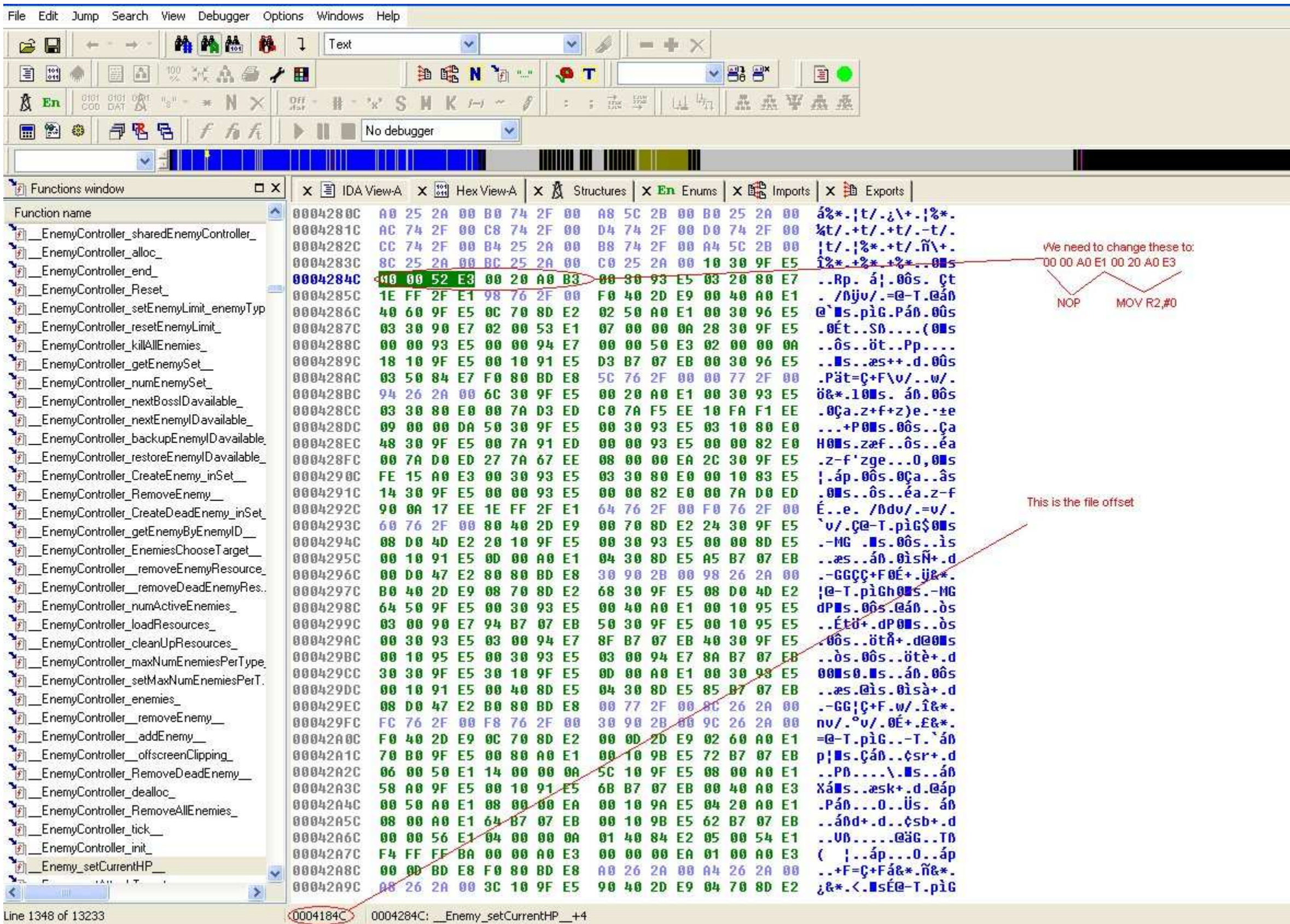
ok. so how do we make 1 hit kill? we supposed that the function is called when we damage the enemies. so if the enemy have 100 HP, and the damage amount is 10, the function checks if the new HP stored in R2 (100-10) is below 0. in our example it isn't, so the game will update enemy HP with the new value (90). what we need to do in order to make 1 hit kill is to ALWAYS update the new HP (R2) with 0. how to do that? very simple! we only have to alter 2 instructions: the 'CMP R2, #0' needs to be changed to NOP (because we no longer need a compare, we'll always make HP=0) the 'MOVL R2, #0' needs to be changed to 'MOV R2, #0' (because we always need to make R2=0)

so the hacked code should look like this:

```
__text:0004284C      NOP
__text:00042850      MOV     R2, #0
__text:00042854      LDR     R3, [R3]
__text:00042858      STR     R2, [R0,R3]
```

ok. as you learnt in plants vs zombies example, we can't make any changes in IDA, so we'll use the hex-editor.
you already know the hex value for NOP (it's 00 00 A0 E1).
the hex value for 'MOV R2, #0' is 00 20 A0 E3

so, knowing these, select the 'CMP R2, #0' in IDA, press the 'Hex View-A' and note the file offset of the instructions we need to change. here's the picture



open the 'A4ER' in the hex-editor, go to offset 4184C, and change the 00 00 53 E3 00 20 A0 B3 to 00 00 A0 E1 00 20 A0 E3

and ENJOY, because you've hacked alive 4-ever returns for 1 hit kill

ENDING

to be able to hack iphone games, you NEED at least the basics of ASM/ARM. so if you didn't understand anything from this guide, learn some ARM and check it again
i tried to keep it as simple as i could.

for more iphone game hacks, be sure to check: <http://iphonecheating.blogspot.com/>

all the best,
HBK