Öğr.Gör. Furkan ÇAKMAK BLM2021 ALT SEVİYE PROGRAMLAMA

**Son Teslim tarihi:** **09.01.2022**

Ödev 2

Oğuzhan Topaloğlu

Ç19052025 – Grup 3

*Bilgisayar Mühendisliği Bölümü,*

*Elektrik-Elektronik Fakültesi,*

*Yıldız Teknik Üniversitesi*

****

Istanbul, 2021

Instruction Set ile makine kodu okuma:

Burada 5-58 ve 5-59 sayfalarında verilen tabloları ve tanımları kullanarak kodu makine kodundan teker teker çevirmeye başladım.

Birkaç örnek veriyorum:

1- MOV AX, 1234H

MOV RW, data16 => 10111 ddd YYYY => 10111 000 1234H => B8H 1234H => “B8 34 12” makine kodu

2- PUSH AX

PUSH RW => 01010 rrr => 01010 000 => 50H

=> “50” makine kodu

Bu örneklerin çoğunu dosbox kullanarak da kontrol ediyorum, boş bir COM tipi asm dosyası oluşturdum ve kodumuzu yazdığımız yere tek bir komut yazarak sürekli kontroller yaptım.

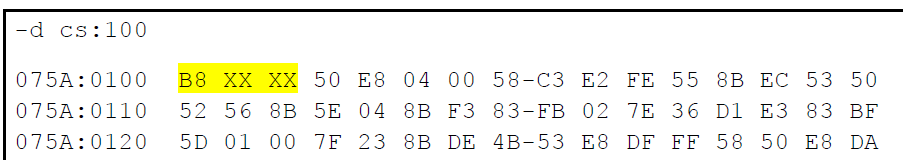
Ayrıca numaram 19052025 olduğu için 19052025 mod 107 = 33 oluyor yani AX = 250 + 33 = 283 gireceğim.

3- MOV AX, 283

MOV RW, data16 => 10111 000 011BH => B8H 011BH

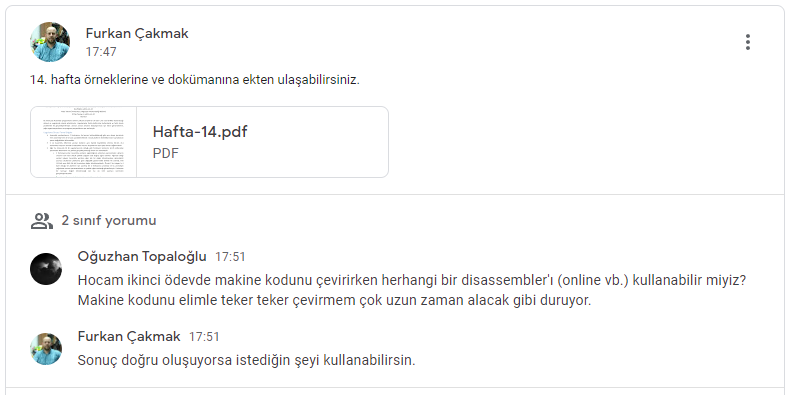
=> “B8 1B 01” makine kodu

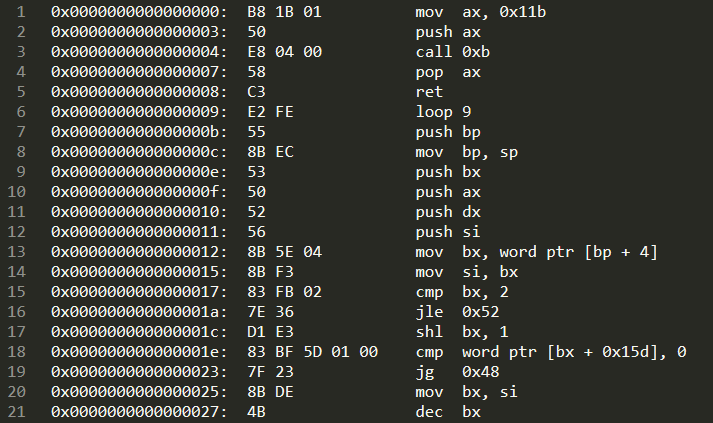
Yani bende sarı kısımla işaretli şu kısım:

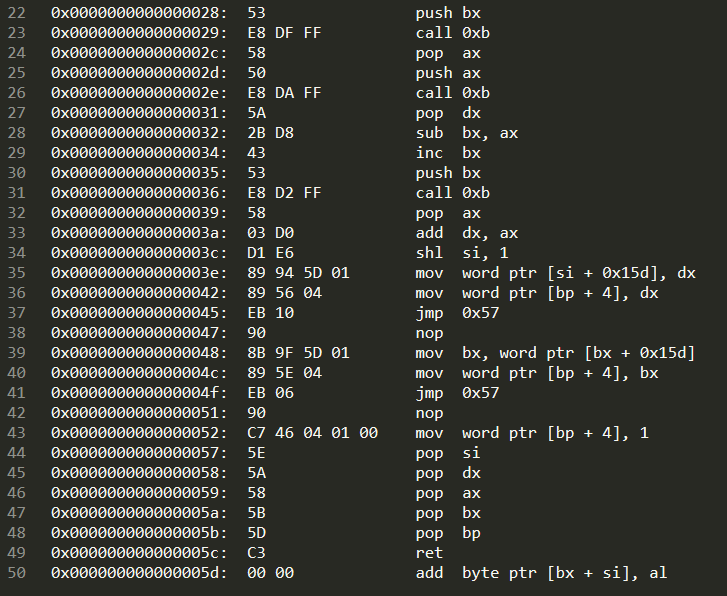


“B8 1B 01” şeklinde olmalı.

Böyle devam eder kodu okuyabilirdim ancak size sorduğum soruya şöyle bir yanıt verdiğiniz için:

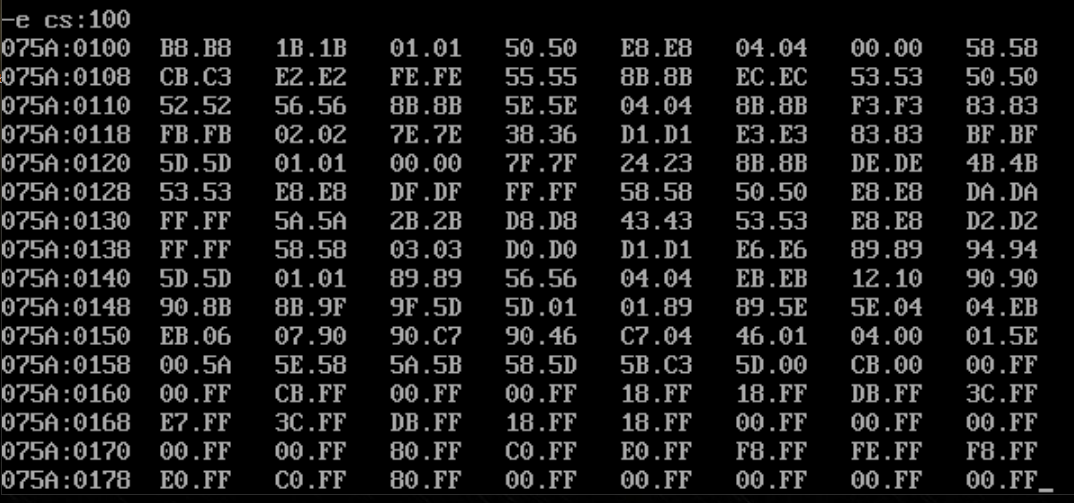
Gittim ve online bir x86 disassembler kullandım ve şu çıktıyı aldım:



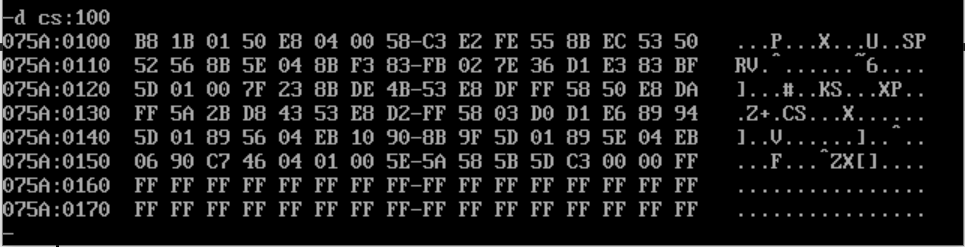


Aslında instruction set ile hepsini teker teker okuyarak bir tablo oluşturacaktım ancak çok uzun sürecek diye bunu yapmadım. DOSBOX ile online siteden aldığım sonucu kontrol ettim ve kendi bulduklarımla doğru olduğunu gördüm.

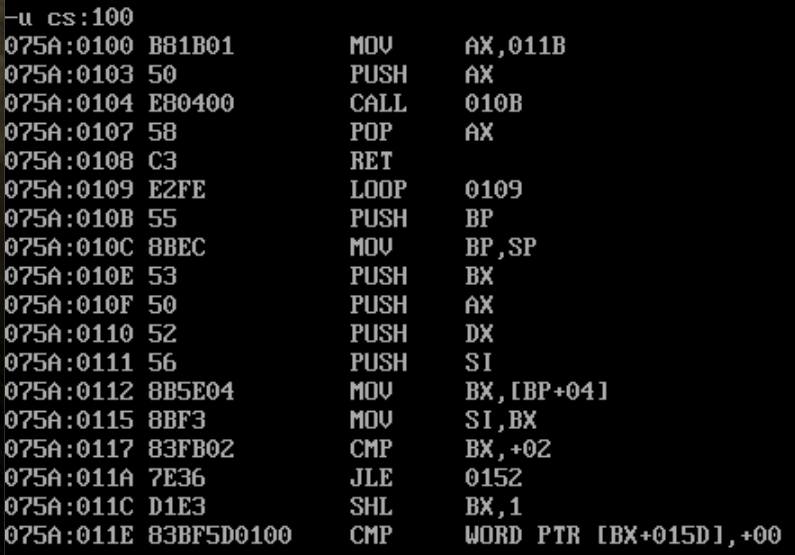
Bunu kontrolü de şöyle yaptım, öncelikle “e cs:100” komutu ile her değeri teker teker girdim sonra da “-t” ve “-t <sayı>” komutları ile trace ederek kodun nasıl çalıştığını inceledim. Ayrıca “-u ...” ve “-d ...” komutları ile de kontrol ettim:

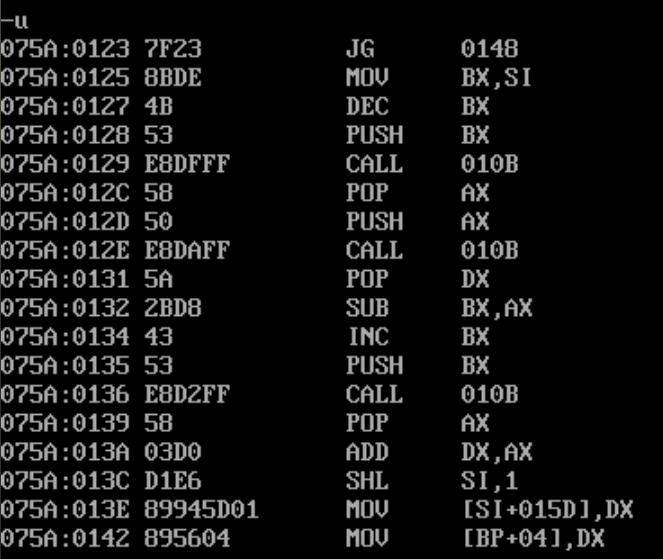
Doldurma:

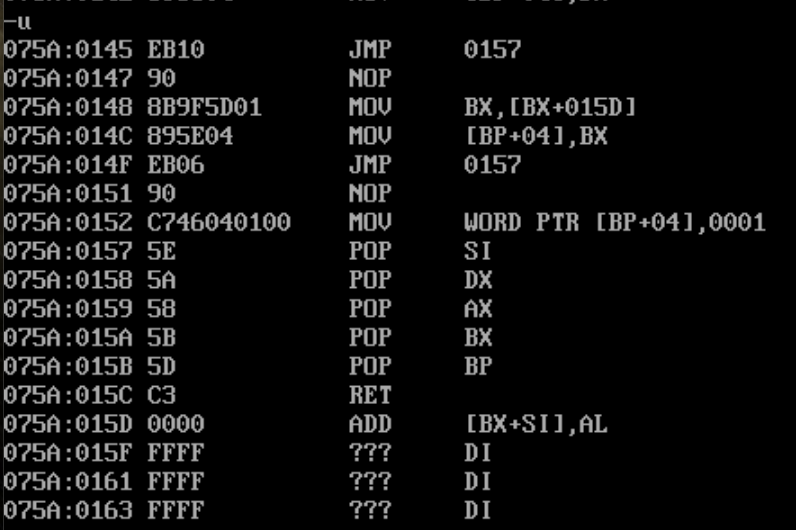
Dump etme:



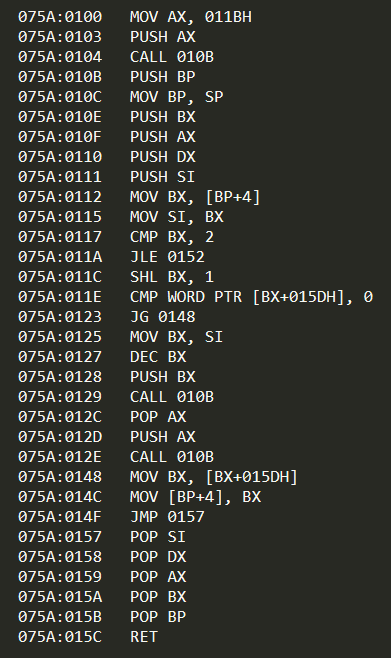
-u komutu ile koda çevirme:







Trace ederken de şöyle notlar aldım:

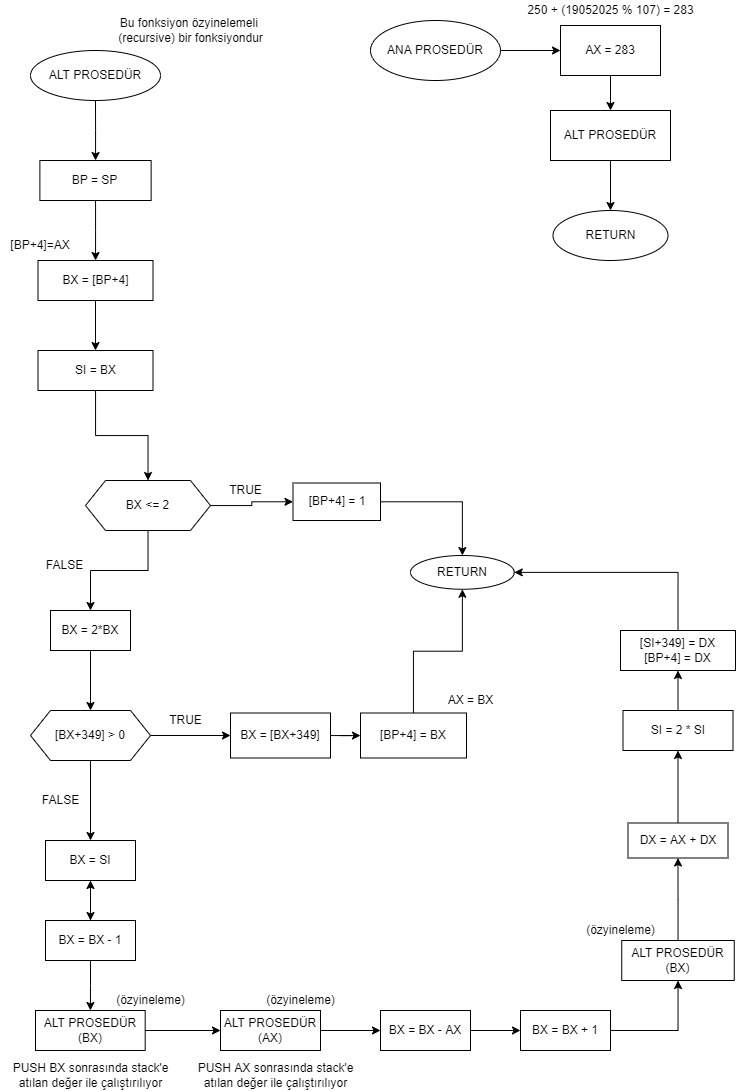


Sitenin bana verdiği her kod parçası bunda bulunmuyor ancak bu normal çünkü hem her satırdaki kodların boyutları (örneğin PUSH AX komutu 50 makine koduna karşılık geldiğinden boyutu 1) birbirinden farklı hem de benim yazdığımda arada boşluklar kaldığı apaçık ortada, örneğin 012E-0148 adresleri arasında bir sürü kod olması gerekiyor. Ayrıca -u komutu ile kodun tümünü görebiliyorum.

NOT: kodun başında “E2 FE” makine kodlu LOOP satırı var ki bu hiçbir zaman kod sırasında çalışıtırılmıyor.

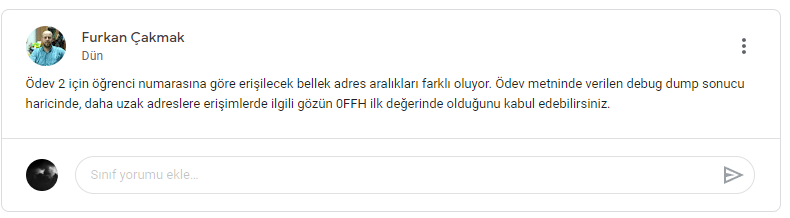
NOT: ayrıca 0123 adresindeki JG dallanmasında data segment hiç değişmiyor.

A seçeneği:



Burada PUSH, POP vb. komutlar üst seviye tanım olmadıklarından dolayı gösterilmedi, çoğu zaten alt prosedürün içinde değerleri korumaya yarıyor. Bir değişikliğe neden olan çok az PUSH/POP komutu var. Bunlardan birisine örnek vermek istersek 012C adresindeki POP komutunu verebiliriz.

07.01.2022 Tarihindeki Açıklama:

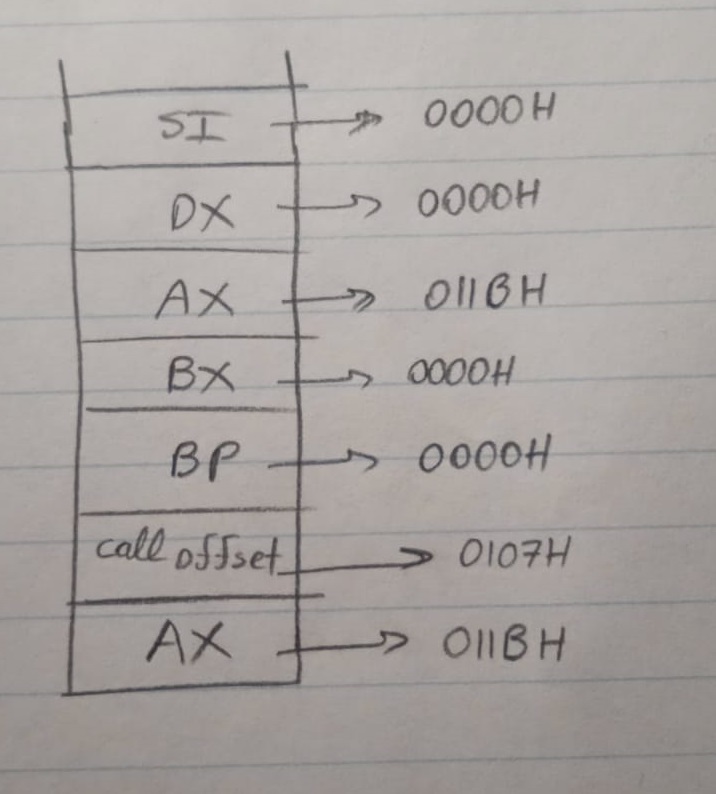


Bu açıklama yapılmadan önce sonsuz döngüye giriliyordu ve stack dolduğunda uygulama çökerek sonlanıyordu. Şimdi her şey değişecek ve uygulama bir bakıma bir jump ile sonlanacak.

Koda genel olarak bakacak olursak şunlar gerçekleşecek:

* AX’i stack’e at
* Call offset’i stack’e at ve 010B adresine atla
* BP’yi stack’e at
* BP = SP yap
* Sırayla BX, AX, DX, SI’yı stack’e at
* BX = [BP+4] yap yani BX = AX yap
* SI = BX yani SI = AX
* BX <= 2 mi? Yani AX <= 2 mi? Bu her zaman yanlış olacak çünkü BX = AX = 250 + (ogrNo mod 107)
* BX = 2 \* BX yani BX = 2 \* AX yap
* [BX+015DH] > 0 mı? Bu yaptığınız açıklamaya göre hep FF değeri elde edeceğinden her zaman doğru olacak ve atlama gerçekleşecek
* BX = [BX+015DH] yani BX=FF yap
* [BP+4] = BX yani AX=FF yap
* Sırayla SI, DX, AX, BX, BP değerlerini stack’ten çıkart. En son stack durumu yukarıdan aşağıya SI, DX, AX, BX, BP, call offset (bir bakıma IP) ve AX olduğundan dolayı burada SI, DX, AX, BX, BP değerleri eski değerlerini kazanacak ve ret ile alt prosedür sonlanacak. Burada RET ile geri döndükten sonra bir POP daha yapacağız ve AX yeniden öğrenci numarasını (283 yani 011BH değerini) kazanacak.

B seçeneği:



AX değerleri ana prosedürde öğrenci numarasının modu + 250 sayısının MOV komutu ile atanmasından sonra 011BH olarak gelmektedir.

Call offset (aslında bir bakıma IP) dediğimiz şey alt prosedür çalıştırıldıktan sonra stack’e atılan ret sonrasında döneceğimiz satırın adresidir.

BP,BX,DX,SI değerleri ise ilk başta 0000H olarak stack’e atılır.

AX atamasını 0103 adresinde, call atamasını 0104 adresinde, BP atamasını 010B adresinde, BX atamasını 010E adresinde, ikinci AX atamasını 010F adresinde, DX atamasını 0110 adresinde ve SI atamasını da 0111 adresinde bulunan komut satırları (PUSH komutları) yapmaktadır.

C seçeneği:

Code segment 100H’ye yapışık olduğundan data segment’in code segment’ten sonra gelmesi gerekiyor. Yani code segment aynı segment içinde data segment’in yukarısında tanımlı.

Burada uygulama bittiğinde AX = 283 olacak ve stack işlemleri sayesinde SI, DX, BX, BP yazmaçları ilk değerlerini koruyacak.

Uygulama bittikten sonra duyuruyu göz önünde bulunduracak olursak en sondaki her adres gözünde FF bulunacaktır.

Duyuru öncesinde sonsuz loop yüzünden uygulama bitmiyordu ve bir süreden sonra sondaki değerlerin hepsi 00H oluyor idi.