

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 YYYYY => 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 $19052025 \bmod 107 = 33$ 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 kısım ile işaretli şu kısım:


```
-d cs:100
075A:0100  B8 XX XX 50 E8 04 00 58-C3 E2 FE 55 8B EC 53 50
075A:0110  52 56 8B 5E 04 8B F3 83-FB 02 7E 36 D1 E3 83 BF
075A:0120  5D 01 00 7F 23 8B DE 4B-53 E8 DF FF 58 50 E8 DA
```


“B8 1B 01” şeklinde olmalı.


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


**Furkan Çakmak**
17:47

14. hafta örneklerine ve dokümanına ekten ulaşabilirsiniz.

**Hafta-14.pdf**
PDF

 2 sınıf yorumu

**Oğuzhan Topaloğlu** 17:51
Hocam ikinci ödevde makine kodunu çevirirken herhangi bir disassembler'ı (online vb.) kullanabilir miyiz?
Makine kodunu elimle teker teker çevirmem çok uzun zaman alacak gibi duruyor.

**Furkan Çakmak** 17:51
Sonuç doğru oluşuyorsa istediğin şeyi kullanabilirsin.

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

```
1 0x0000000000000000: B8 1B 01      mov ax, 0x11b
2 0x0000000000000003: 50             push ax
3 0x0000000000000004: E8 04 00      call 0xb
4 0x0000000000000007: 58             pop ax
5 0x0000000000000008: C3             ret
6 0x0000000000000009: E2 FE         loop 9
7 0x000000000000000b: 55             push bp
8 0x000000000000000c: 8B EC         mov bp, sp
9 0x000000000000000e: 53             push bx
10 0x000000000000000f: 50             push ax
11 0x0000000000000010: 52             push dx
12 0x0000000000000011: 56             push si
13 0x0000000000000012: 8B 5E 04      mov bx, word ptr [bp + 4]
14 0x0000000000000015: 8B F3         mov si, bx
15 0x0000000000000017: 83 FB 02      cmp bx, 2
16 0x000000000000001a: 7E 36         jle 0x52
17 0x000000000000001c: D1 E3         shl bx, 1
18 0x000000000000001e: 83 BF 5D 01 00 cmp word ptr [bx + 0x15d], 0
19 0x0000000000000023: 7F 23         jg 0x48
20 0x0000000000000025: 8B DE         mov bx, si
21 0x0000000000000027: 4B             dec bx
```

```

22  0x0000000000000028:  53          push bx
23  0x0000000000000029:  E8 DF FF    call 0xb
24  0x000000000000002c:  58          pop ax
25  0x000000000000002d:  50          push ax
26  0x000000000000002e:  E8 DA FF    call 0xb
27  0x0000000000000031:  5A          pop dx
28  0x0000000000000032:  2B D8       sub bx, ax
29  0x0000000000000034:  43          inc bx
30  0x0000000000000035:  53          push bx
31  0x0000000000000036:  E8 D2 FF    call 0xb
32  0x0000000000000039:  58          pop ax
33  0x000000000000003a:  03 D0       add dx, ax
34  0x000000000000003c:  D1 E6       shl si, 1
35  0x000000000000003e:  89 94 5D 01  mov word ptr [si + 0x15d], dx
36  0x0000000000000042:  89 56 04     mov word ptr [bp + 4], dx
37  0x0000000000000045:  EB 10       jmp 0x57
38  0x0000000000000047:  90          nop
39  0x0000000000000048:  8B 9F 5D 01  mov bx, word ptr [bx + 0x15d]
40  0x000000000000004c:  89 5E 04     mov word ptr [bp + 4], bx
41  0x000000000000004f:  EB 06       jmp 0x57
42  0x0000000000000051:  90          nop
43  0x0000000000000052:  C7 46 04 01 00  mov word ptr [bp + 4], 1
44  0x0000000000000057:  5E          pop si
45  0x0000000000000058:  5A          pop dx
46  0x0000000000000059:  58          pop ax
47  0x000000000000005a:  5B          pop bx
48  0x000000000000005b:  5D          pop bp
49  0x000000000000005c:  C3          ret
50  0x000000000000005d:  00 00       add byte ptr [bx + si], al

```

Aslında instruction set ile hepsini teker teker okuyarak bir tablo oluşturacaktım ancak çok uzun sürecekti diye bunu yapmadım. DOSBOX ile online siteden aldığım sonucu kontrol ettim ve kendi bulduklarım ile 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:

```
-e cs:100
075A:0100 B8.B8 1B.1B 01.01 50.50 E8.E8 04.04 00.00 58.58
075A:0108 CB.C3 E2.E2 FE.FE 55.55 8B.8B EC.EC 53.53 50.50
075A:0110 52.52 56.56 8B.8B 5E.5E 04.04 8B.8B F3.F3 83.83
075A:0118 FB.FB 02.02 7E.7E 38.36 D1.D1 E3.E3 83.83 BF.BF
075A:0120 5D.5D 01.01 00.00 7F.7F 24.23 8B.8B DE.DE 4B.4B
075A:0128 53.53 E8.E8 DF.DF FF.FF 58.58 50.50 E8.E8 DA.DA
075A:0130 FF.FF 5A.5A 2B.2B D8.D8 43.43 53.53 E8.E8 D2.D2
075A:0138 FF.FF 58.58 03.03 D0.D0 D1.D1 E6.E6 89.89 94.94
075A:0140 5D.5D 01.01 89.89 56.56 04.04 EB.EB 12.10 90.90
075A:0148 90.8B 8B.9F 9F.5D 5D.01 01.89 89.5E 5E.04 04.EB
075A:0150 EB.06 07.90 90.C7 90.46 C7.04 46.01 04.00 01.5E
075A:0158 00.5A 5E.58 5A.5B 58.5D 5B.C3 5D.00 CB.00 00.FF
075A:0160 00.FF CB.FF 00.FF 00.FF 18.FF 18.FF DB.FF 3C.FF
075A:0168 E7.FF 3C.FF DB.FF 18.FF 18.FF 00.FF 00.FF 00.FF
075A:0170 00.FF 00.FF 80.FF C0.FF E0.FF F8.FF FE.FF F8.FF
075A:0178 E0.FF C0.FF 80.FF 00.FF 00.FF 00.FF 00.FF 00.FF_
```

Dump etme:

```
-d cs:100
075A:0100 B8 1B 01 50 E8 04 00 58-C3 E2 FE 55 8B EC 53 50 ...P...X...U..SP
075A:0110 52 56 8B 5E 04 8B F3 83-FB 02 7E 36 D1 E3 83 BF RV.^.....~6....
075A:0120 5D 01 00 7F 23 8B DE 4B-53 E8 DF FF 58 50 E8 DA l...#...KS...XP..
075A:0130 FF 5A 2B D8 43 53 E8 D2-FF 58 03 D0 D1 E6 89 94 .Z+.CS...X.....
075A:0140 5D 01 89 56 04 EB 10 90-8B 9F 5D 01 89 5E 04 EB l..U.....l..^...
075A:0150 06 90 C7 46 04 01 00 5E-5A 58 5B 5D C3 00 00 FF ...F...^ZXll....
075A:0160 FF FF FF FF FF FF FF FF-FF FF FF FF FF FF FF .....
075A:0170 FF FF FF FF FF FF FF FF-FF FF FF FF FF FF FF .....
-
```

-u komutu ile koda çevirme:

```

-u cs:100
075A:0100 B81B01      MOV     AX,011B
075A:0103 50          PUSH    AX
075A:0104 E80400      CALL    010B
075A:0107 58          POP     AX
075A:0108 C3          RET
075A:0109 E2FE        LOOP    0109
075A:010B 55          PUSH    BP
075A:010C 8BEC        MOV     BP,SP
075A:010E 53          PUSH    BX
075A:010F 50          PUSH    AX
075A:0110 52          PUSH    DX
075A:0111 56          PUSH    SI
075A:0112 8B5E04      MOV     BX,[BP+04]
075A:0115 8BF3        MOV     SI,BX
075A:0117 83FB02      CMP     BX,+02
075A:011A 7E36        JLE     0152
075A:011C D1E3        SHL     BX,1
075A:011E 83BF5D0100  CMP     WORD PTR [BX+015D],+00

```

```

-u
075A:0123 7F23        JG      0148
075A:0125 8BDE        MOV     BX,SI
075A:0127 4B          DEC     BX
075A:0128 53          PUSH    BX
075A:0129 E8DFFF      CALL    010B
075A:012C 58          POP     AX
075A:012D 50          PUSH    AX
075A:012E E8DAFF      CALL    010B
075A:0131 5A          POP     DX
075A:0132 2BD8        SUB     BX,AX
075A:0134 43          INC     BX
075A:0135 53          PUSH    BX
075A:0136 E8D2FF      CALL    010B
075A:0139 58          POP     AX
075A:013A 03D0        ADD     DX,AX
075A:013C D1E6        SHL     SI,1
075A:013E 89945D01    MOV     [SI+015D],DX
075A:0142 895604    MOV     [BP+04],DX

```

```
-u
075A:0145 EB10      JMP      0157
075A:0147 90        NOP
075A:0148 8B9F5D01   MOV      BX,[BX+015D]
075A:014C 895E04   MOV      [BP+04],BX
075A:014F EB06      JMP      0157
075A:0151 90        NOP
075A:0152 C746040100   MOV      WORD PTR [BP+04],0001
075A:0157 5E        POP      SI
075A:0158 5A        POP      DX
075A:0159 58        POP      AX
075A:015A 5B        POP      BX
075A:015B 5D        POP      BP
075A:015C C3        RET
075A:015D 0000      ADD      [BX+SI],AL
075A:015F FFFF      ???     DI
075A:0161 FFFF      ???     DI
075A:0163 FFFF      ???     DI
```

Trace ederken de şöyle notlar aldım:

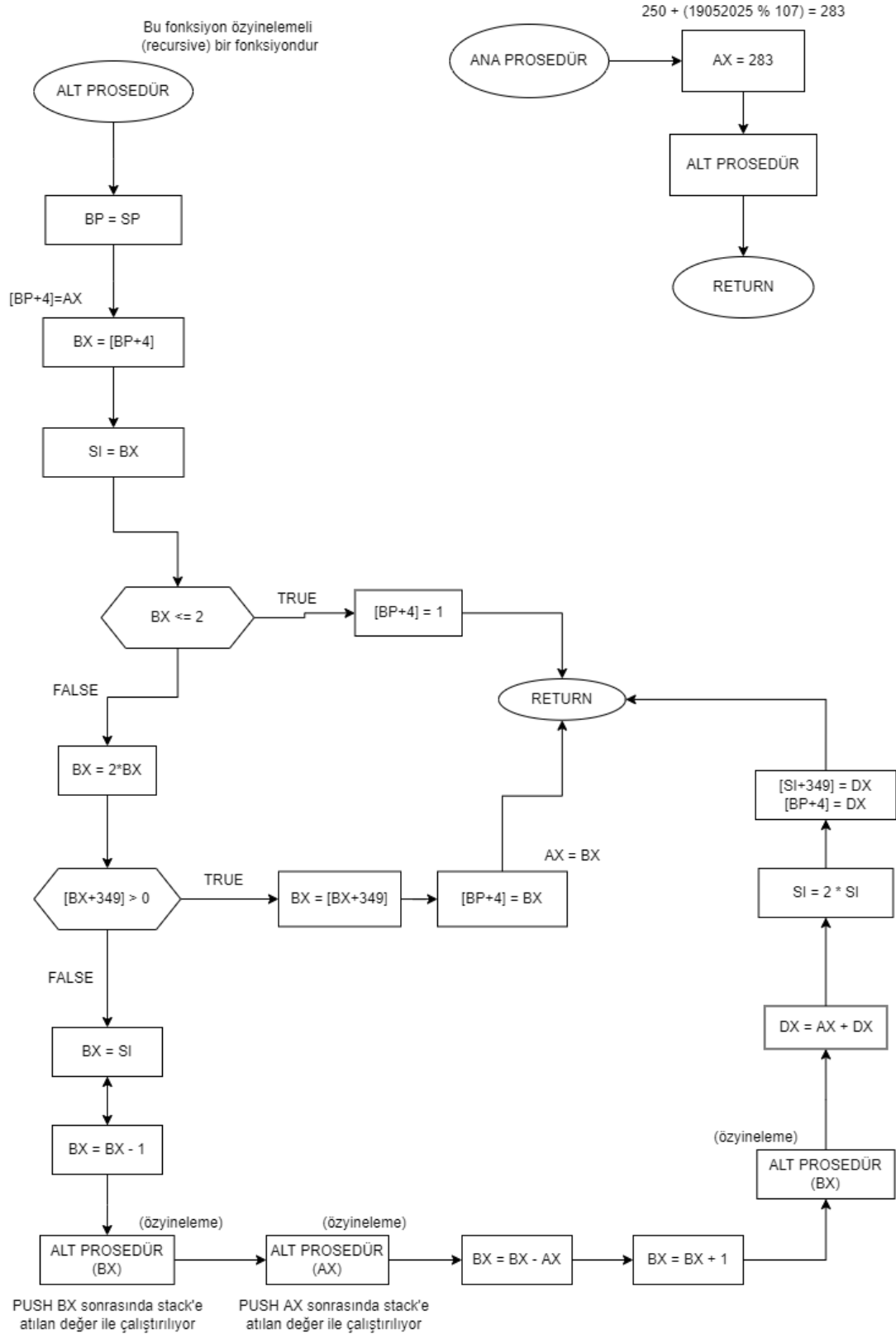
```
075A:0100  MOV AX, 011BH
075A:0103  PUSH AX
075A:0104  CALL 010B
075A:010B  PUSH BP
075A:010C  MOV BP, SP
075A:010E  PUSH BX
075A:010F  PUSH AX
075A:0110  PUSH DX
075A:0111  PUSH SI
075A:0112  MOV BX, [BP+4]
075A:0115  MOV SI, BX
075A:0117  CMP BX, 2
075A:011A  JLE 0152
075A:011C  SHL BX, 1
075A:011E  CMP WORD PTR [BX+015DH], 0
075A:0123  JG 0148
075A:0125  MOV BX, SI
075A:0127  DEC BX
075A:0128  PUSH BX
075A:0129  CALL 010B
075A:012C  POP AX
075A:012D  PUSH AX
075A:012E  CALL 010B
075A:0148  MOV BX, [BX+015DH]
075A:014C  MOV [BP+4], BX
075A:014F  JMP 0157
075A:0157  POP SI
075A:0158  POP DX
075A:0159  POP AX
075A:015A  POP BX
075A:015B  POP BP
075A:015C  RET
```

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 çalıştırılmıyor.


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:

**Furkan Çakmak**
Dün⋮

Ödev 2 için öğrenci numarasına göre erişilecek bellek adres aralıkları farklı oluyor. Ödev metninde verilen debug dump sonucu haricinde, daha uzak adreslere erişimlerde ilgili gözün 0FFH ilk değerinde olduğunu kabul edebilirsiniz.



➤

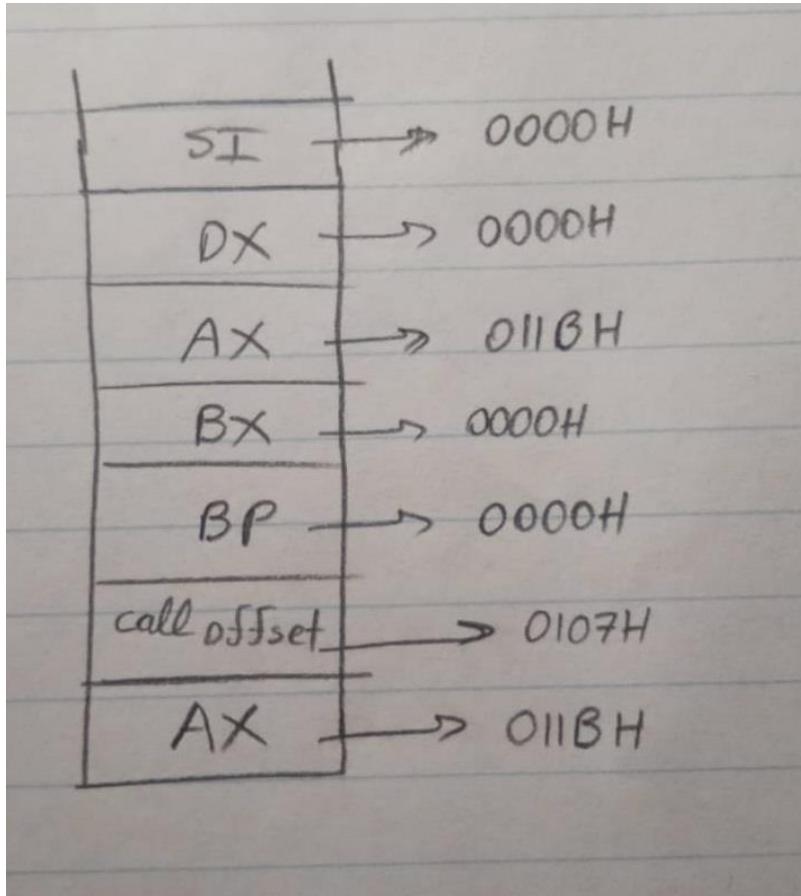
Bu açıklama yapılmadan önce sonsuz döngüye giriliyordu ve stack dolduğunda uygulama çökerek sonlanıyordu. Şimdi her şey değişecek ve uygulama bir bakıma bir jump ile sonlanacak.

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.