Mikroişlemciler, II. Uygulama Yönergesi

Amaç: Assembly dili programların 2-geçişli (2-pass) Assembler ile yapılması.
Hedef: Çalıştırıldığında, "Merhaba Dunya" mesajını ekrana yazdıran basit bir programın 2-geçişli Assembler'lar (tasm, tasm32, masm32) ile geliştirilmesi ve Borland'ın Turbo Debugger hata ayıklama aracı ile adım adım çalıştırılması.

Bu uygulamada, MicroSoft'un Macro Assembler uygulamasına rakip olarak Borland tarafından geliştirilen Turbo Assembler'ın 2. sürümü ile x86 Assembly dilinde yazılan programların derlenmesi ve bağlanarak (link) tek başına çalıştırılabilir (*.exe, stand alone executable) dosyaya dönüştürülmesi gerçekleştirilecektir. Çalıştırılan exe uzantılı dosyanın beklenen sonuçları üretmemesi durumunda, mikroişlemcinin kaydedici düzeyinde izlenerek programın potansiyel sorunlarını analiz etmek gerekebilmektedir. Program akışını izleyebilmek için, Borland tarafından bağımsız bir hata ayıklayıcı olarak geliştirilen ve pazarlanan fakat Turbo öneki kullanan Pascal, C ve C++ dillerinin, yazılım geliştiriciler tarafından yaygın kullanımı nedeniyle bu derleyiciler ile birlikte dağıtılmaya başlanan, kullanımı kolay Turbo Debugger hata ayıklama aracı kullanılacaktır. Turbo Assembler ve Turbo Debugger'ın kullanımı kısaca açıklandıktan sonra, bu araçlar kullanılarak Assembly dilindeki basit bir programın geliştirilmesi ve izlenmesi aşamaları örnek olarak verilecektir.

Uyarı: Bu oturumda kullanılan Turbo Assembler, TASM.EXE ve Turbo Degugger, TD.EXE 16-bit uygulamalar olduğu için 1. uygulama yönergesinde açıklandığı gibi, 32-bit işletim sistemlerinde doğrudan çalıştırılabilir olmasına karşılık, MicroSoft'un 64-bit işletim sistemlerinde çalıştırılması olanaksızdır. 64-bit işletim sistemi kurulu bilgisayarlarda bu yönergede anlatılan Turbo Assembler, TASM.EXE ve Turbo Debugger, TD.EXE'yi çalıştırabilmek için, açık kaynak kodlu QEMU işlemci emulatörüne grafik kullanıcı arayüzü sağlayan QemuSimpleBoot (qsib) kullanabilirsiniz. Qsib çalıştırıldığında, disk imajı (IMG) seçeneği işaretlendikten sonra, içinde 8-bit M68xx ve 16-bit x86 Assembly dilleri için gereken araçların bulunduğu çalıştırılabilir (bootable) disk imaj dosyasından işletim sistemini başlatarak bu programları çalıştırabilirsiniz. Yazdığınız programın oluşturulduğu/çalıştırıldığı 16-bit DOS ortamından, bilgisayarınızın dosya sistemine aktarılması için **MagicISO** adlı ISO kalıp hazırlama/düzenleme programını kullanabilirsiniz.

Bu yönergede açıklanan Assembly dili program, 2-geçişli bir Assembler ile geliştirilecek olması sayesinde, tek geçişli Assembler'ların yalnız geri yöndeki mutlak adresleri referanslama kısıtlaması ortadan kalkmaktadır. Tek geçişli (debug vb.) ve iki geçişli (Turbo Assembler vb.) Assembler'lar arasındaki bu fark, aşağıda debug ile Unassemble edilen örnek program parçasından açıklanabilir.

Tek geçişli (debug vb.) Assembler kodu

```
17C2:014A      90              NOP
17C2:014B      8B164B3A        MOV     DX,[3A4B]
17C2:014F      ED              IN      AX,DX
17C2:0150      3B064D3A        CMP     AX,[3A4D]
17C2:0154      7F02            JG      0158
17C2:0156      EBF7            JMP     014F
17C2:0158      8B164F3A        MOV     DX,[3A4F]
17C2:015C      A1513A          MOV     AX,[3A51]
17C2:015F      EF              OUT     DX,AX
17C2:0160      90              NOP
```

2-geçişli (masm, tasm vb.) Assembler kodu

```
90                      NOP
8B164B3A                MOV     DX,[SEVIYE]
ED              TEKRAR: IN      AX,DX
3B064D3A                CMP     AX,[SINIR]
7F02                    JG      TAMAM
EBF7                    JMP     TEKRAR
8B164F3A        TAMAM:  MOV     DX,[VANA]
A1513A                  MOV     AX,[KAPAT]
EF                      OUT     DX,AX
90                      NOP
```

Bir depodaki sıvı seviyesini, **`SEVIYE`** adlı adreste bulunan bir algılayıcıdan okuduktan sonra, sıvı seviyesinin **`SINIR`** adresindeki değerden büyük olması durumunda, **`VANA`** adresinde yer alan çıkış kapısına, **`KAPAT`** adresindeki değeri göndermekte, seviye bu değerden küçük veya eşit olduğu durumlarda ise okuma ve karşılaştırma işine döngü içinde devam etmektedir. Döngünün başlangıç adresi olan $\texttt{014F}_{16}$ değeri, program tek geçişli olan debug ile yazılırken adres olarak bilindiği için **`JMP`** emir parametresi için doğrudan kullanılabilmesine karşılık, döngü dışına çıkıldığında gidilecek

adres (**TAMAM** etiketi ile belirtilen 0158_{16}) bilinemeyeceği için **JG** emirine parametre olarak kullanılacak değeri bu emiri yazarken bilmek olanaksızdır. Tek geçişli Assembler'ların bu kısıtlaması, kaynak kod üzerinden en az iki geçişle tüm (ileri ve geri yöndeki) referansları belirleyip sembol tablosuna yerleştirdikten sonra, ikinci geçişte kod üretimi yapan iki geçişli Assembler'lar ile ortadan kaldırılmıştır. İki geçişli Assembler, yukarıda sağ tarafta verilen ve sembolik adreslerin kullanıldığı program üzerinden ilk geçiş sonucunda aşağıdaki **sembol tablosu**nu oluşturur.

Sembolik Adres	Gerçek Adres
...	...
SEVIYE	**3A4B**
SINIR	**3A4D**
VANA	**3A4F**
KAPAT	**3A51**
...	...

Assembly gibi düşük seviyeli bir dilde bile, değişken değerlerinin tutulduğu adreslerin sembolik adresler biçiminde kullanımı, programın okunurluğunu ve böylece programcının kod geliştirme verimini büyük ölçüde artırmaktadır.

Turbo Assembler ile bu yönergede geliştirilmesi özetle anlatılacak olan program, 1. uygulama yönergesinde olduğu gibi 21. DOS kesmesinin 9. işlevini kullanarak ekrana bir karakter dizisi yazdırmaktadır. Turbo Assembler, tek geçişli Assembler, debug ile kıyaslanamayacak kadar gelişmiş bir 2-geçişli Assembler olup, komut satırından **tasm** adı ile başlatılmaktadır. Aşağıda Turbo Assembler 2.0'ın, 1. uygulama yönergesinde 64-bit işletim sistemleri için önerilen emülatör üzerinde **dos5asm** imajı içinden çalıştırıldığında kullanımını özetleyen ekran çıktısı görülmektedir.

```
Turbo Assembler  Version 2.0  Copyright (c) 1988, 1990 Borland International
Syntax:  TASM [options] source [,object] [,listing] [,xref]
/a,/s          Alphabetic or Source-code segment ordering
/c             Generate cross-reference in listing
/dSYM[=VAL]    Define symbol SYM = 0, or = value VAL
/e,/r          Emulated or Real floating-point instructions
/h,/?          Display this help screen
/iPATH         Search PATH for include files
/jCMD          Jam in an assembler directive CMD (eg. /jIDEAL)
/kh#,/ks#      Hash table capacity #, String space capacity #
/l,/la         Generate listing: l=normal listing, la=expanded listing
/ml,/mx,/mu    Case sensitivity on symbols: ml=all, mx=globals, mu=none
/mv#           Set maximum valid length for symbols
/m#            Allow # multiple passes to resolve forward references
/n             Suppress symbol tables in listing
/o,/op         Generate overlay object code, Phar Lap-style 32-bit fixups
/p             Check for code segment overrides in protected mode
/q             Suppress OBJ records not needed for linking
/t             Suppress messages if successful assembly
/w0,/w1,/w2    Set warning level: w0=none, w1=w2=warnings on
/w-xxx,/w+xxx Disable (-) or enable (+) warning xxx
/x             Include false conditionals in listing
/z             Display source line with error message
/zi,/zd        Debug info: zi=full, zd=line numbers only
A:\>_
```

İlk satırdaki temel kullanım bilgisinden de görüldüğü gibi Assembler, **asm** uzantılı dosya içindeki emir kodlarından oluşan Assembly programı derleyerek, **obj** uzantılı çalıştırılabilir amaç (object) kod dosyası oluşturabilmektedir. Her ne kadar 1. uygulama yönergesinde açıklanan şekilde, debug'a komut satırından girilecek tüm veriler bir dosyaya (örneğin komutlar.txt) yazılarak yönlendirme ile;

```
A:\>debug < komutlar.txt
```

debug'ın bu dosyayı okuması ve çalıştırılabilir dosyayı üretmesi sağlanabilse de, Assembly dilinde programları bu yöntemle geliştirmek, sembolik değişken adları ve etiketlerin kullanımına izin veren 2-geçişli bir Assembler kullanımından çok daha zordur.

Assembly dilinde program geliştirirken kullanabileceğiniz basit bir araç **dos5asm** imaj dosyası içinde **util** dizini içinde bulunan **SHOW.EXE** adlı dosyadır. Sistemin çalıştırılabilir dosyaları için tanımlanan yolu (**autoexec.bat** içinde **set path=A:\DOS;A:\UTIL;A:\X86;A:\TURBO;A:\M68** satırı ile tanımlanan) içinde bulunduğu için her dizinden çalıştırılabilen bu araç ile programlarınızı düzenlemeniz gerekmektedir.

Ekrana bir metin mesajı yazacak program, 1. uygulama yönergesinde debug aracını kullanarak yazdığınız programın bir benzeri olduğu için aşağıdaki şekilde **asm** uzantılı bir metin dosyaya yazılmalıdır.

```
.model tiny                    ; COM dosya türü, (CS=DS)
.code                          ; Code Segment (CS) başlangıç adres tanımı
      mov   ax,@data           ; Data Segment (DS) başlangıç adresini AX'e al
      mov   ds,ax              ; DS'ye aktar
      mov   ah,9               ; DOS Int 21'in 9 işlevi (ptr ds:dx yazdır)
      mov   dx,offset metin    ; metin'in Data Segment (DS) içindeki yeri
      int   21h                ; DOS Int 21h servisinin 9. işlevini çağır
      mov   ah,4ch             ; DOS Int 21h'in 4ch işlevi, program sonlandır
      int   21h                ; DOS Int 21h servisinin 4ch işlevini çağır
.data                          ; Data Segment (DS) başlangıç adres tanımı
metin db    'Merhaba Dunya$'   ; Kodun bittiği noktadan başlayarak metin
end
```

Programın ilk satırında yer alan **.model tiny** satırı, işlemci üreticisi tarafından önerilen Bellek Modellerinden **com** uzantılı dosyalar için kullanılan ve bellek parçalarının (kod, veri, yığın ve ekstra) aynı değere eşit olduğu (**CS=DS=SS=ES**) 64 kB'dan küçük programlar (komutlar) için önerilen bellek modelini seçmesini Assembler'a bildirmektedir.

İkinci satırda bulunan **.code** ifadesi, kod parçasının başladığı noktayı bildirmektedir. Bu satırları izleyen iki satırda, verinin içinde bulunduğu **.data** (**D**ata **S**egment, **DS**) başlangıç adres, önce **ax** kaydedicisine ve daha sonra oradan **ds** kaydedicisine aktarılmaktadır.

Program kodlarının 1. uygulama yönergesindeki kaynak koda, **dx** kaydedicisi içine metin'in başlangıç adresinin atandığı satır dışında benzerliği dikkatinizi çekmiş olmalıdır. Programın bittiği noktadan sonra, **.data** ile başlayan kısımda, karakter dizisinin tanımı önündeki **metin** adı, bu dizi için programcı tarafından seçilen sembolik ad olup, Int 21, 9. işlev ile yazdırmak için dizinin adresini tutması gereken **dx** kaydedicisine **offset** öneki ile **ds** içindeki pozisyonu (ileri yönde referans olduğu için 1. uygulama yönergesinde programcı tarafından adresler sayılarak hesaplanan 0109_{16} değerine karşı düşen) kolaylıkla aktarılmaktadır.

Programın diğer bir farklılığı, **int 20** ile programın sonlandırılması yerine, program sonlandırmak için DOS Int 21'in 4c. işlevini kullanmasıdır.

Ekrana 'Merhaba Dunya' dizisini yazdıran **asm** uzantılı **merhaba.asm** adlı dosyanın, **tasm** ile komut satırından derlenerek **obj** uzantılı **merhaba.obj** dosyasını üretmesi için;

```
A:\>tasm merhaba
```

yazılması yeterlidir. Bu aşamadan sonra üretilmiş olan **obj** uzantılı dosya, bu dosya içerisinde tanımlanmayan diğer sembolik referansların (sort, fact, rnd vb. işlevler veya pi, e gibi değişmezler)

çözümlenebileceği diğer **obj** uzantılı dosyalar ile birlikte bir araya getirilerek (bağlanarak, link edilerek) kendi başına çalıştırılabilir (stand-alone executable) dosya üretimi için kullanılacaktır. Bu yönergede anlatılan programda dosya içinde tanımlanmamış olmayan hiçbir sembolik isim yer almadığı için, bağlama işlemi Turbo Link (**tlink**) programını komut satırından;

```
A:\>tlink merhaba
```

ile çalıştırıp, **exe** uzantılı çalıştırılabilir dosya üretilecektir. Üretilen **exe** dosyanın, **T**urbo **D**ebugger **TD** ile hata ayıklama amacıyla satır satır çalıştırılması için **tasm**'ye **/zi** opsiyonu verilerek derleme yapılmalı ve **tlink**'e **/v** opsiyonu verilerek tüm hata ayıklama bilgisinin **exe** dosya içinde yer alması sağlanmalıdır. İçinde hata ayıklama amacı ile kaynak kod seviyesinde bilgi tutan bu dosyalar, Turbo Debugger, **td** ile komut satırından;

```
A:\>td merhaba
```

yazarak F7 veya F8 ile adım adım çalıştırılır ve kod içinde bulunan hatalar ayıklanmaya çalışılır.

Turbo Assembler (tasm), Linker (tlink) ve Debugger (td) programlarının, 32-bit işletim sistemleri için sürümleri de (tasm32, tlink32 ve td32) bulunduğundan, Assembly dilinde yazacağınız programlar, **Qsib** ile çalıştırılan 16-bit DOS ortamına gerek kalmadan doğrudan 64-bit MicroSoft işletim sistemlerinde derlenebilir ve çalıştırılabilir. 32-bit Turbo Assembler'ın 5. sürümünün, assembler, linker ve debugger'ı, gerekli diğer dosyalar ile birlikte sıkıştırılmış bir arşiv dosyada indirip çalıştırmanız için hazırlanmıştır. Bu dosyayı indirip bilgisayarınızda açtıktan sonra, ekrana karakter dizisi yazdıran aşağıdaki Assembly dili programı bir komut penceresinden derleyerek çalıştırabilirsiniz.

```
; --- Islemci secimi ---
.586
; --- Duz bellek modeli, ve WinApi fonksiyon cagrilari yontem secimi ---
.MODEL FLAT, STDCALL
; --- Standart cikis tutmaci (handle) degeri ---
 STD_OUTPUT_HANDLE equ -11
; --- WinApi icinden cagrilacak fonksiyon prototipleri ---
EXTRN GetStdHandle      :     NEAR
EXTRN WriteConsoleA     :     NEAR
EXTRN ExitProcess :     NEAR

; --- Veri kismi icinde yapilan tanimlamalar ---
.data
metin       DB "Merhaba Dunya",0
uzunluk     DD ?  ; Bilinmiyor, WriteConsoleA yazdigi chr sayisini saklayacak
reserve     DD 0  ; WriteConsoleA cagrisi reserve parametresi
HandleOut   DD ?  ; Bilinmiyor, Standart cikis (console) tutmaci saklanacak
; --- Kod kismi icinde kodun govdesi ---
.code
START:
; --- Standard cikis tutmacini al ---
      push  STD_OUTPUT_HANDLE
      call  GetStdHandle
      mov   HandleOut,eax
; --- Karakter Dizisi boyunu hesapla, ebx de geri dondur---
      push  offset metin
      call  StrLen
; --- Uzunluk ve diğer parametreleri hazirlayarak ekrana (Console) yazdir
      push  offset reserve    ; reserve (null, 0) olmasi yeterli
      push  offset uzunluk    ; yazilan karakter sayisi (geri donus degeri)
      push  ebx               ; metin boyu
      push  offset metin      ; metin adresi
      push  eax               ; cikis tutmaci
```

```
      call  WriteConsoleA
; --- Isletim sistemine geri don ---
      push 0
      call ExitProcess
;
StrLen PROC
; --- Kullanilacak tum kaydedicileri yigina sakla ---
      push  ebp
      mov   ebp,esp
      push  eax
      push  edi
; --- Metin sonu belirteci (0) gorene kadar say ---
      cld
      mov   edi,dword ptr [ebp+08h]
      mov   ebx,edi
      mov   ecx,100
      xor   al,al
      repne scasb
      sub   edi,ebx
      mov   ebx,edi
      dec   ebx
; --- Kaydedici içeriklerini yigindan geri al ve geri don ---
      pop   edi
      pop   eax
      pop   ebp
      ret   4
StrLen ENDP

END START
```

Bu programı derlemek ve Turbo Debugger ile çalıştırmak için **tasm5** dosyasını açtığınızda oluşan **tasm32** dizini içinde komut satırından;

```
C:\Users\Ali\Desktop\tasm32>tasm32 /ml /zi merhaba
C:\Users\Ali\Desktop\tasm32>tlink32 /Tpe /v merhaba,,, import32.lib
C:\Users\Ali\Desktop\tasm32>td32 merhaba
```

yazılması, veya bu üç programı gerekli parametreler ile çalıştıran **asm.bat** adlı dosyanın;

```
C:\Users\Ali\Desktop\tasm32>asm merhaba
```

ile başlatılması yeterlidir.

Komut satırından Assembly dilinde program geliştirmenin zorluklarını azaltmayı hedefleyen bir çok program bulunmaktadır. Bu programlar arasında, programcıların en çok tercih ettikleri, MASM32 64-bit MicroSoft işletim sistemleri ile sorunsuz çalışmakta ve Assembly dilinde derlenmiş bir Tümleşik Geliştirme Ortamı (**I**ntegrated **D**evelopment **E**nvironment, **IDE**) sunmaktadır. Bunun yanında birçok hazır kütüphanesi ile program yazmayı çok basitleştirmektedir. Aşağıda, uzunluğu kullanıcı tarafından verilen bir karakter dizisini komut satırına bastıran basit bir Assembly programın kaynak kodu verilmiştir.

```
.586
.model flat, stdcall
option casemap :none

include C:\masm32\include\msvcrt.inc
include C:\masm32\include\kernel32.inc

includelib C:\masm32\lib\kernel32.lib
includelib C:\masm32\lib\msvcrt.lib
```

```
.data
      metin       db "Merhaba Dunya",10
      uzunluk     dd 0Eh
.code
start:
      mov   eax, uzunluk
      push  eax
      push  offset metin
      push  1
      call  crt__write
      push  0
      call  ExitProcess
end start
```

MASM32 ile yazılan yukarıdaki programda, metin boyunu hesaplayan bir altprogram kullanılmadığı için kullanıcının metin boyunu uzunluk adlı değişkene yazması gerekmektedir. MASM32 ile, yükleme anında Assembly dilindeki kaynak kodundan derlenen, çok hızlı bir metin düzenleyicinin kullanıldığı tümleşik geliştirme ortamı (IDE) desteklenmektedir. IDE içinde Assembly dili komutlar hakkında içeriğe duyarlı (context sensitive) yardım alma ve WinApi işlevlerinin parametreleri hakkında yardım almak amacı ile birçok online dokümandan yardım alma ihtiyacı büyük oranda azalmaktadır.

Kaynaklar :

http://mattst88.com/programming/AssemblyProgrammersJournal
http://win32assembly.programminghorizon.com/tutorials.html
http://www.madwizard.org/programming/tutorials/