Aufgabe 01

Simon Kramer

May 14, 2023

1 Aufgabe

Dieses Programm ist in Assembler geschrieben (.686). Das Programm soll beim Ausführen zuerst überprüfen, ob die CPU den Befehl, CPUID unterstützt. Falls die CPU den Befehl nicht unterstützt, dann soll das Programm mit Hilfe der Windows API beendet werden.

Sollte die CPU den Befehl unterstützen, soll der String "CPUID supported" ausgegeben werden. Gefolgt, in der nächsten Zeile, wird das Ergebnis des Befehls "CPUID" geprinted. Dies soll der "Prozessor Brand" String sein.

2 CpuId

CPUID ist ein Befehl, der benutzt werden kann um Informationen über die CPU abzufragen. Dabei kann das Register EAX benutzt werden um unterschiedliche Informationen abzufragen. EAX = 0, gibt dabei eine Zahl aus, die Zeigt ob zusätzliche Operationen unterstützt werden. EAX = 80000002h - 80000005h fragt den CPU Brand String ab.

3 Code

```
.data
NewLine dw 0A0Dh,0
CpuidSupportedMsg db "CPUID supported!",0
.code
...
push -11
call GetStdHandle
mov consoleOutHandle, eax
...
push 0
push 0
push 16
push OFFSET CpuidSupportedMsg
```

```
push consoleOutHandle
call WriteConsoleA
```

Der obere Code wird benutzt um Informationen in der Konsole auszugeben. Dafür müssen wir zuerst eine Referenz auf das Konsolenfenster schaffen. Dafür rufen wir die Methode "GetStdHandle" auf und übergeben ihr den Parameter "-11".

Dieser wird dem Befehl "WriteConsoleA" übergeben. Des weiteren wird auch der Speicherort des Textes, der ausgegeben wird, übergeben (im obigen Beispiel "CpuidSupportedMsg"). Zudem muss auch die Länge des ausgegebenen Wortes angegeben werden, (im obigen Beispiel "16").

Weil wir auch Zeilenumbrüche nutzten, müssen wir diesen auch als Wert im Speicher Definieren. Der Wert um eine neue Zeile zu beginnen ist "0A0D" (als Hexadezimalzahl).

```
; Check if cpuid is supported
; if supported : jmp
pushfd
pushfd
xor dword ptr[esp],00200000h
popfd
pushfd
pop eax
xor eax,[esp]
popfd
and eax,00200000h

;if(cpuid.isSupported) goto CpuIdSupported;
cmp eax, 0
jne CpuIdSupported
```

Dies ist Boilerplate Code aus dem Internet zur Überprüfung, ob die CPU den cpuid Befehl ausführen kann. Der Code wurde hier (https://wiki.osdev.org/CPUID) und musste leicht abgeändert werden, damit dieser in MASM funktioniert. Kurz, die CPU Modelle, die cpuid unterstützen erlauben es die eflags zu ändern und dort Werte zu speichern. Modelle die dies nicht erlauben können den Befehl cpuid nicht ausführen. Das macht man sich hier zu nutzte indem man den Wert der eflag abspeichert, verändert und schaut ob diese Veränderung übernommen wurde. Wenn nichts verändert wurde, sollte ein XOr-Gate 0 ausgeben, sonst eine andere Zahl. So kann man im darauffolgenden If-Statement überprüfen ob die Zahl nicht 0 ist und eine Entscheidung auf den Support des Befehls treffen. In diesem Programm, wenn der Befehl nicht unterstützt wird, würde das Programm enden und die Nachricht "CPU not Supported" ausgeben. Sonst springt das Programm zum Codeblock, der ausgeführt werden soll, wenn cpuid unterstützt wird.

```
; {
        .data
        cpuCounter dd 80000002h,0

        ...

        .code

        ...

        PrintLoop:

        ; cpuStr = cpuid(cpuCounter)
        ; Write(cpuStr)

        ;cpuCounter++
        mov ecx,cpuCounter
        add ecx, 1
        mov cpuCounter, ecx


        ;}do while(cpuCounter != 80000005h)
cmp ecx, 80000005h
jl PrintLoop
```

Für den ganzen "Processor Brand String", müssen EAX von 80000002h - 80000005h abgefragt werden. Um dies zu vereinfachen, nutzen wir eine Do-While-Schleife. Dafür definieren wir oben den Anfang mit "PrintLoop:" und schreiben an das Ende einen Jump-Befehl (jl in diesem Fall). Durch "cmp ecx, 80000005h" vergleichen wir den Wert von cpuCounter mit dem Wert 80000005h. jl bedeutet "jump if less than", demnach springen wir immer, wenn der cpuCounter kleiner als 80000005h ist.

Nun wurde der in ".data" der initial Wert auf 80000002h gesetzt und wird jedes mal vor der Überprüfung, in der Schleife inkrementiert.So decken wir alle EAX ab, die wir abfragen wollen.

Zum inkrementieren laden wir den Wert von cpuCounter kurzfristig ins ECX Register,inkrementieren mit "add ecx, 1' und speichern dann den Wert wieder in cpuCounter ab.

```
cpuid

;returns manufacturer ID
;cpuStr = eax + ebx + ecx + edx
mov dword ptr[cpuStr]   , eax
mov dword ptr[cpuStr+4] , ebx
mov dword ptr[cpuStr+8] , ecx
```

```
mov dword ptr [cpuStr+12], edx
; Write(cpuStr)
```

Bevor wir cpuid ausführen setzen wir den EAX Wert, auf den Wert, der im cpuCounter steht. Danach wird cpuid ausgeführt, welcher die Informationen über die CPU abfragt und die Werte in den Registern EAX, EBX, ECX und EDX abgespeichert. Da wir alle Werte jedoch als einen String ausgeben wollen, nutzen wir ein zuvor definiertes dword (cpuStr). Wir verweisen dann auf diese Speicher und speichern die Ergebniswerte von cpuid ab.

Jedes Register ist 4 Byte groß, wenn wir das hintereinander abspeichern wollen, müssen wir jeweils 4 Bytes weiter gehen. Also schreiben wir,EAX an die Stelle cpuStr+0, EBX an die Stelle cpuStr+4, ECX an die Stelle cpuStr+8 und EDX an die Stelle cpuStr+12. So steht der String von cpuStr+0 bis cpuStr+16. Wir geben das mit dem obigen Code aus und geben dem Konsolen Buffer eine Länge von 16 Bytes.