Blekinge Tekniska Högskola

DV1493/DV1613 – Datorteknik

Bibliotek för in- och utmatning (Intel x64 assembler)

Författare: Carina Nilsson

Innehåll

1 Inledning

Syftet med den här lilla projektlabben är att bli bekant med ytterligare en processortyp, Intel (eller AMD som är kompatibel). Assemblerspråk för denna processortyp finns i olika syntaxvarianter. I projektuppgiften ska AT&T-syntax som dominerar i Unix-världen användas. Det är den som `gcc (gas)` följer. Det finns en annan variant som kallas Intel-syntax som är omvänd och som används i Windows-världen.

Uppgiften består i att skriva ett litet bibliotek med rutiner för in- och utmatning av text. Biblioteksrutinerna ska kunna anropas från ett högnivåprogram skrivet i t ex programspråket C. Det är då viktigt att hålla sig till samma konventioner som kompilatorn gör för hur register och stack används. Annars kommer rutinerna inte att fungera tillsammans med högnivåkoden. In- och utmatning på lägsta nivå är ganska besvärligt och innefattar hantering av avbrott, men själva avbrottshanteringen ingår inte i den här uppgiften.

2 Utförande och redovisning

Uppgiften görs lämpligen i grupper om två. Grupper med tre eller fler deltagare accepteras inte! Observera att uppgiften tar väsentligt mycket mer tid i anspråk än de schemalagda handledningstillfällena. När hela uppgiften är implementerad laddas koden upp i inlämningsmappen i lärplattformen.

Observera att det är inte tillåtet att kopiera kod som någon annan än gruppmedlemmarna har konstruerat.

3 Förutsättningar

I Unix/Linux betraktas alla enheter som filer, och att läsa från tangentbordet är att läsa från ”filen” stdin. Att skriva till terminaldisplayen är att skriva till ”filen” stdout. För den råa överföringen får endast de färdiga C-rutinerna `fgets` och `puts` användas. De rutinerna blir gränssnittet mot systemet och avbrottshanteringen, och gör att avbrott inte behöver hanteras av de biblioteksrutiner som ska skrivas inom ramen för uppgiften. Rutinen `fgets` används på följande sätt: `fgets(buffert, antal, fil)` i C, där `buffert` är adressen till ett minnesutrymme till vilket den lästa raden ska sparas, `antal` är det maximala antalet tecken att läsa och `fil` anger varifrån läsningen ska ske. Rutinen läser normalt till dess att ny rad (Newline = EOL) eller filslut (End of File = EOF) påträffas. Strängen läggs i bufferten och avslutas med NULL-tecknet (ASCII-kod 0). I assemblerspråk för Intel x64 kan ett anrop till `fgets` se ut så här:

```
1  someCode:
2
3          movq    $buf, %rdi      # lägg i buf, där buf är en bit reserverat minne
4          movq    $5, %rsi        # högst 5-1=4 tecken (NULL räknas ju också)
5          movq    stdin, %rdx     # från standard input stdin=$0 om ej def.
6          call    fgets
```

Det finns en motsvarande rutin `puts` för utmatningen, som i C anropas enligt: `puts(buffert)` . Funktionen skriver ut minnesinnehållet som `buffert` refererar till som en sträng, tills ett NULL-tecken påträffas.

4 Uppgift

Uppgiften går ut på att implementera rutiner för att hantera inmatning och utmatning av data. För att klara av det behöver man reservera plats för två olika systembuffertar, en för inmatning och en för utmatning. Var och en av dessa buffertar behöver också någon variabel som håller reda på aktuell position i respektive buffert. Eftersom ett bibliotek ska implementeras måste nedanstående specifikation följas. Biblioteket ska ligga i en separat fil som kompileras och länkas tillsammans med testprogrammet `Mprov64.s` när sluttestet sker.

4.1 Inmatning

De rutiner som ska implementeras för *inmatning* är följande:

4.1.1 `inImage`

Rutinen ska läsa in en ny textrad från tangentbordet till er inmatningsbuffert för indata och nollställa den aktuella positionen i den. De andra inläsningsrutinerna kommer sedan att jobba mot den här bufferten. Om inmatningsbufferten är tom eller den aktuella positionen är vid buffertens slut när någon av de andra inläsningsrutinerna nedan anropas ska `inImage` anropas av den rutinen, så att det alltid finns ny data att arbeta med.

4.1.2 `getInt`

Rutinen ska tolka en sträng som börjar på aktuell buffertposition i inbufferten och fortsätta tills ett tecken som inte kan ingå i ett heltal påträffas. Den lästa substrängen översätts till heltalsformat och returneras. Positionen i bufferten ska vara det första tecken som inte ingick i det lästa talet när rutinen lämnas. Inledande blanktecken i talet ska vara tillåtna.

Ett plustecken eller ett minustecken ska kunna inleda talet och vara direkt följt av en eller flera heltalssiffror. Ett tal utan inledande plus eller minus ska alltid tolkas som positivt. Om inmatningsbufferten är tom eller om den aktuella positionen i inmatningsbufferten är vid dess slut vid anrop av `getInt` ska `getInt` kalla på `inImage`, så att `getInt` alltid returnerar värdet av ett inmatat tal.
Returvärde: inläst heltal

4.1.3 `getText`

Rutinen ska överföra maximalt `n` tecken från aktuell position i inbufferten och framåt till minnesplats med början vid `buf`. När rutinen lämnas ska aktuell position i inbufferten vara första tecknet efter den överförda strängen. Om det inte finns `n` st. tecken kvar i inbufferten avbryts överföringen vid slutet av bufferten. Returnera antalet verkligt överförda tecken. Om inmatningsbufferten är tom eller aktuell position i den är vid buffertens slut vid anrop av `getText` ska `getText` kalla på `inImage`, så att `getText` alltid läser över någon sträng till minnesutrymmet som`buf` pekar till. Kom ihåg att en sträng per definition är NULL-terminerad.
Parameter 1: adress till minnesutrymme att kopiera sträng till från inmatningsbufferten (`buf` i texten)
Parameter 2: maximalt antal tecken att läsa från inmatningsbufferten (`n` i texten)
Returvärde: antal överförda tecken

4.1.4 `getChar`

Rutinen ska returnera ett tecken från inmatningsbuffertens aktuella position och flytta fram aktuell position ett steg i inmatningsbufferten ett steg. Om inmatningsbufferten är tom eller aktuell position i den är vid buffertens slut vid anrop av `getChar` ska `getChar` kalla på `inImage`, så att `getChar` alltid returnerar ett tecken ur inmatningsbufferten.
Returvärde: inläst tecken

4.1.5 `getInPos`

Rutinen ska returnera aktuell buffertposition för inbufferten.
Returvärde: aktuell buffertposition (index)

4.1.6 `setInPos`

Rutinen ska sätta aktuell buffertposition för inbufferten till `n`. `n` måste dock ligga i intervallet `[0,MAXPOS]`, där `MAXPOS` beror av buffertens faktiska storlek. Om `n`<0, sätt positionen till 0, om `n`>`MAXPOS`, sätt den till `MAXPOS`.
Parameter: önskad aktuell buffertposition (index), `n` i texten.

4.2 Utmatning

De rutiner som ska implementeras för *utmatning* är följande:

4.2.1 `outImage`

Rutinen ska skriva ut strängen som ligger i utbufferten i terminalen. Om någon av de övriga utdatarutinerna når buffertens slut, så ska ett anrop till `outImage` göras i dem, så att man får en tömd utbuffert att jobba mot.

4.2.2 `putInt`

Rutinen ska lägga ut talet `n` som sträng i utbufferten från och med buffertens aktuella position. Glöm inte att uppdatera aktuell position innan rutinen lämnas.
Parameter: tal som ska läggas in i bufferten (`n` i texten)

4.2.3 `putText`

Rutinen ska lägga textsträngen som finns i `buf` från och med den aktuella positionen i utbufferten. Glöm inte att uppdatera utbuffertens aktuella position innan rutinen lämnas. Om bufferten blir full så ska ett anrop till `outImage` göras, så att man får en tömd utbuffert att jobba vidare mot.
Parameter: adress som strängen ska hämtas till utbufferten ifrån (`buf` i texten)

4.2.4 `putChar`

Rutinen ska lägga tecknet `c` i utbufferten och flytta fram aktuell position i den ett steg. Om bufferten blir full när `getChar` anropas ska ett anrop till `outImage` göras, så att man får en tömd utbuffert att jobba vidare mot.
Parameter: tecknet som ska läggas i utbufferten (`c` i texten)

4.2.5 `getOutPos`

Rutinen ska returnera aktuell buffertposition för utbufferten.
Returvärde: aktuell buffertposition (index)

4.2.6 `setOutPos`

Rutinen ska sätta aktuell buffertposition för utbufferten till `n`. `n` måste dock ligga i intervallet `[0,MAXPOS]`, där `MAXPOS` beror av utbuffertens storlek. Om `n`<0 sätt den till 0, om `n`>`MAXPOS` sätt den till `MAXPOS`.
Parameter: önskad aktuell buffertposition (index), `n` i texten

4.3 Körexempel med testprogrammet

När programmet testas kan en lyckad körning med biblioteket kompilerat och länkat ihop med testprogrammet se ut som något i stil med det här (användarinmatningar är återgivna i kursiv stil):

Körexempel

```
Start av testprogram. Skriv in 5 tal!
1 2 -47
5 7 Kalle
1+2-47+5+7=-32
Kalle
125
Testprogram slut
```

5 Litet exempelprogram

Här följer ett litet kodexempel i Intel x64 assembler som kan testas som uppvärmning. Funktionen borde vara känd sedan Laboration 1, men då skrev ni den i ARMv6 assembler.

```
1   .data
2   resMsg: .asciz   "fak=%d\n"
3   buf:    .asciz   "xxxxxxxxx"
4   endMsg: .asciz   "slut\n"
5
6   .text
7   .global main
8   main:
9       pushq   $0           #Stacken ska vara 16 bytes "aligned"
10      movq    $5, %rdi     # Beräkna 5!
11      call    fac
12      movq    %rax, %rsi  #Flytta returvärdet till argumentregistret (arg2)
13      movq    $resMsg, %rdi  # skriv ut Fak= "resultat", adr. till formatsträng i arg1
14      call    printf
15  # läs med fgets(buf,5,stdin)
16      movq    $buf, %rdi  # lägg i buf, adr. arg1
17      movq    $5,%rsi     # högst 5-1=4 tecken, arg2
18      movq    stdin, %rdx # från standard input, arg3
19      call    fgets
20      movq    $buf, %rdi  # adress till sträng i arg1
21      call    printf      # skriv ut buffert
22      movq    $endMsg, %rdi   # följd av slut
23      call    printf
24      popq    %rax
25      ret     # avsluta programmet
26
27
28
29
30      # Här finns funktionen fac = n! (rekursiv)
31  fac:
32      cmpq    $1,%rdi     # if n>1
33      jle     lBase
34      pushq   %rdi        # lägg anropsvärde på stacken
35      decq    %rdi        # räkna ned värdet med 1
36      call    fac        # temp = fakultet av (n-1)
37      popq    %rdi        # hämta från stack
38      imul    %rdi,%rax    # return n*temp
39      ret                 # Återvänd
40
41  lBase:
42      movq    $1,%rax     # else return 1
43      ret                  # Återvänd
```