|  |  |  |
| --- | --- | --- |
| **Course**: | COMP 2131 | **Programming Assignment-3** |
| **Weight:** | **100 marks = 10**% **of final grade** | **Working with Assembly loops** |

Step 1. Write a program in GNU assembly language that uses macros to print the following messages on the screen [20 marks]:

Hello, programmers! Welcome to the world of, Linux assembly programming!

Ans1)

A macro with two parameters  
; Implements the write system call  
   %macro write\_string 2   
      mov   eax, 4  
      mov   ebx, 1  
      mov   ecx, %1  
      mov   edx, %2  
      int   80h  
   %endmacro  
   
section    .text  
   global \_start             
      
\_start:                      
   write\_string msg1, len1                
   write\_string msg2, len2      
   write\_string msg3, len3    
      
   mov eax,1                 
   int 0x80

section    .data  
msg1 db    'Hello, programmers!',0xA,0xD      
len1 equ $ - msg1

msg2 db 'Welcome to the world of,', 0xA,0xD   
len2 equ $- msg2

msg3 db 'Linux assembly programming! '  
len3 equ $- msg3

Step 2. Using the C Programming language, write four version of a function that contain a loop. Each function should accept two numbers and calculate the sum of all numbers between the first number and last number (inclusive of the first and last number). Tip: Try to use the same number of variables and almost the same logic when writing the C code for all four functions**.** Once each function is written and tested for correct output, generate an assembly language version of the function using the command: *$gcc -O1 -S filename.c.* (where *filename.c* is the C program containing the function). Next, analyze and compare the assembly language version of each function. [50 marks]

1. Write a version of the function using a *for* loop
2. Write a version of the function using a *while* loop
3. Write a version of the function using a *do..while* loop
4. Write a version of the function using a *goto* loop
5. Is the assembly language version of each loop function the same or different? If different, identify the differences. Your comparison should be based on:
   * Number of registers used
   * Number of jumps (iterations)
   * Total number of operations

#include <stdio.h>

void main()

{

int n1,n2,s;

printf("Please enter the two nbers:");

scanf("%d%d",&n1,&n2);

s = for\_Loop(n1,n2); // Qustion 2(a)

printf("\nS With For Loop is %d",s);

s = while\_Loop(n1,n2); // Qustion 2(b)

printf("\nS With While loop is %d",s);

s = do\_WhileLoop(n1,n2); // Qustion 2(c)

printf("\nS With DoWhile loop is %d",s);

s = go\_toLoop(n1,n2); // Qustion 2(d)

printf("\nS With GoTo is %d",s);

}

int getMinimum(int n1, int n2)

{

int minimum;

if(n1 < n2 )

{

minimum = n1;

}

else

{

minimum = n2;

}

}

int getMaximum(int n1, int n2)

{

int maximum;

if(n1 > n2 )

{

maximum = n1;

}

else

{

maximum = n2;

}

}

int for\_Loop(int n1,int n2){

int i,minimum,maximum,s;

minimum = getMinimum(n1,n2);

maximum = getMaximum(n1,n2);

s = 0;

for(i = minimum; i <= maximum ; i++)

{

s = s + i;

}

return s;

}

int while\_Loop(int n1,int n2){

int i,minimum,maximum,s;

minimum = getMinimum(n1,n2);

maximum = getMaximum(n1,n2);

s = 0;

i = minimum;

while(i <= maximum)

{

s = s + i;

i++;

}

return s;

}

int do\_WhileLoop(int n1,int n2){

int i,minimum,maximum,s;

minimum = getMinimum(n1,n2);

maximum = getMaximum(n1,n2);

s = 0;

i = minimum;

do

{

s = s + i;

i++;

}while(i <= maximum);

return s;

}

int go\_toLoop(int n1,int n2){

int i,minimum,maximum,s;

minimum = getMinimum(n1,n2);

maximum = getMaximum(n1,n2);

s = 0;

i = minimum;

doS: s = s + i;

i++;

if(i <= maximum)

{

goto doS;

}

return s;

}

Assembly version of the code:

       .file "Test1.c"

       .text

       .def   \_\_main;       .scl   2;     .type 32;    .endef

       .section .rdata,"dr"

.LC0:

       .ascii "Please enter the two nbers:\0"

.LC1:

       .ascii "%d%d\0"

.LC2:

       .ascii "\12S With For Loop is %d\0"

.LC3:

       .ascii "\12S With While loop is %d\0"

.LC4:

       .ascii "\12S With DoWhile loop is %d\0"

.LC5:

       .ascii "\12S With GoTo is %d\0"

       .text

       .globl main

       .def   main; .scl   2;     .type 32;    .endef

       .seh\_proc     main

main:

       pushq %rsi

       .seh\_pushreg %rsi

       pushq %rbx

       .seh\_pushreg %rbx

       subq   $56, %rsp

       .seh\_stackalloc     56

       .seh\_endprologue

       call   \_\_main

       leaq   .LC0(%rip), %rcx

       call   printf

       leaq   44(%rsp), %rdx

       leaq   40(%rsp), %r8

       leaq   .LC1(%rip), %rcx

       call   scanf

       movl   $0, %eax

       movl   $1, %edx

.L3:

       addl   $1, %eax

       cmpl   %edx, %eax

       jne    .L3

       movl   $0, %edx

       leaq   .LC2(%rip), %rcx

       call   printf

       movl   $0, %eax

       movl   $1, %edx

.L5:

       addl   $1, %eax

       cmpl   %eax, %edx

       jne    .L5

       movl   $0, %edx

       leaq   .LC3(%rip), %rcx

       call   printf

.L6:

       addl   $1, %esi

       testl %esi, %esi

       jle    .L6

       movl   %esi, %edx

       leaq   .LC4(%rip), %rcx

       call   printf

.L7:

       addl   $1, %ebx

       testl %ebx, %ebx

       jle    .L7

       movl   %ebx, %edx

       leaq   .LC5(%rip), %rcx

       call   printf

       nop

       addq   $56, %rsp

       popq   %rbx

       popq   %rsi

       ret

       .seh\_endproc

       .globl getMinimum

       .def   getMinimum;   .scl   2;     .type 32;    .endef

       .seh\_proc     getMinimum

getMinimum:

       .seh\_endprologue

       ret

       .seh\_endproc

       .globl getMaximum

       .def   getMaximum;   .scl   2;     .type 32;    .endef

       .seh\_proc     getMaximum

getMaximum:

       .seh\_endprologue

       ret

       .seh\_endproc

       .globl for\_Loop

       .def   for\_Loop;     .scl   2;     .type 32;    .endef

       .seh\_proc     for\_Loop

for\_Loop:

       .seh\_endprologue

       movl   $0, %eax

       movl   $1, %edx

.L16:

       addl   $1, %eax

       cmpl   %eax, %edx

       jne    .L16

       movl   $0, %eax

       ret

       .seh\_endproc

       .globl while\_Loop

       .def   while\_Loop;   .scl   2;     .type 32;    .endef

       .seh\_proc     while\_Loop

while\_Loop:

       .seh\_endprologue

       movl   $0, %eax

       movl   $1, %edx

.L20:

       addl   $1, %eax

       cmpl   %eax, %edx

       jne    .L20

       movl   $0, %eax

       ret

       .seh\_endproc

       .globl do\_WhileLoop

       .def   do\_WhileLoop; .scl   2;     .type 32;    .endef

       .seh\_proc     do\_WhileLoop

do\_WhileLoop:

       .seh\_endprologue

.L23:

       addl   $1, %eax

       testl %eax, %eax

       jle    .L23

       ret

       .seh\_endproc

       .globl go\_toLoop

       .def   go\_toLoop;    .scl   2;     .type 32;    .endef

       .seh\_proc     go\_toLoop

go\_toLoop:

       .seh\_endprologue

.L26:

       addl   $1, %eax

       testl %eax, %eax

       jle    .L26

       ret

       .seh\_endproc

       .ident "GCC: (tdm64-1) 9.2.0"

       .def   printf;       .scl   2;     .type 32;    .endef

       .def   scanf; .scl   2;     .type 32;    .endef

here is both assembly version and C code

Step 3. Using the C Programming language, write a program that sums an array of 50 elements. Next, optimize the code using loop unrolling. Loop unrolling is a program transformation that reduces the number of iterations for a loop by increasing the number of elements computed on each iteration.

Generate a graph of performance improvement. Tip: Figure 5.17 in the textbook provides an example of a graph depicting performance improvements associated with loop unrolling. [30 marks]

Ans3)

c program without loop unrollong:

|  |
| --- |
| #include<stdio.h> #include<stdlib.h>  int main() { int n,sum=0; n=50; int a[50]; for(int i=0;i<n;i++) { scanf("%d",&a[i]); } for(int i=0;i<n;i++) { sum=sum+a[i]; } printf("sum is %d",sum); return 0;  } |

with loop unrolling:

|  |
| --- |
| #include<stdio.h> #include<stdlib.h>  int main() { int n,sum=0; n=50; int a[50]; for(int i=0;i<n;i++) { scanf("%d",&a[i]); } sum=sum+a[0]; sum=sum+a[1]; sum=sum+a[2]; sum=sum+a[3]; sum=sum+a[4]; sum=sum+a[5]; sum=sum+a[6]; sum=sum+a[7]; sum=sum+a[8]; sum=sum+a[9]; sum=sum+a[10]; sum=sum+a[11]; sum=sum+a[12]; sum=sum+a[13]; sum=sum+a[14]; sum=sum+a[15]; sum=sum+a[16]; sum=sum+a[17]; sum=sum+a[18]; sum=sum+a[19]; sum=sum+a[20]; sum=sum+a[21]; sum=sum+a[22]; sum=sum+a[23]; sum=sum+a[24]; sum=sum+a[25]; sum=sum+a[26]; sum=sum+a[27]; sum=sum+a[28]; sum=sum+a[29]; sum=sum+a[30]; sum=sum+a[31]; sum=sum+a[32]; sum=sum+a[33]; sum=sum+a[34]; sum=sum+a[35]; sum=sum+a[36]; sum=sum+a[37]; sum=sum+a[38]; sum=sum+a[39]; sum=sum+a[40]; sum=sum+a[41]; sum=sum+a[42]; sum=sum+a[43]; sum=sum+a[44]; sum=sum+a[45]; sum=sum+a[46]; sum=sum+a[47]; sum=sum+a[48]; sum=sum+a[49];  printf("sum is %d",sum); return 0;  } |

Submission:

Make a Word document that contains all the programs that you wrote for Step 1, Step 2 and Step 3.

* For Step 1, include only the assembly code.
* For Step 2, include the C code for the 4 functions and the assembly code that you generated for each function. Be sure to include your analysis of the assembly code generated from each function.
* For Step 3, include the C code and the assembly code that you generated for each step of the loop unrolling process.

Submit the word document to the Open Learning Faculty Member for grading.

|  |  |  |
| --- | --- | --- |
| Step | Jobs Done | Marks |
| I | Write a program in GNU assembly language using macros. | 20 |
| II(a) | Write a function in C containing a *for* loop | 5 |
| II(b) | Write a function in C containing a *while* loop | 5 |
| II(c) | Write a function in C containing a *do ..while* loop | 5 |
| II(d) | Write a function in C containing a *goto* loop | 5 |
| II(e) | Generate the assembly language version for each of the four functions;  analyze and compare the assembly language code for the four functions. | 30 |
| III | Optimize the code for an array of 50 elements using loop unrolling.  Prepare a graph that show the performance improvements | 30 |
| Total | | 100 |