QUIZ NAVIGATION

1 2

Show one page at a time

Finish review

Started on	Thursday, 9 September 2021, 3:30 PM
State	Finished
Completed on	Thursday, 9 September 2021, 4:30 PM
Time taken	1 hour
Grade	**30.00** out of 30.00 (**100**%)

Question 1

Complete

Mark 14.00 out of 14.00

Flag question

Write comment in every line of the following assembly language snippet. Clearly explain what it does. The following snippet is generated from a C function that is part of a large C program.

Note: First column indicates line number. You can use the line number (no need to copy assembly code if you use line number correctly). For .cfi directives, just mention it and may skip the explanation.

.LFB3:

01 .cfi_startproc
02 pushq %rbp
03 .cfi_def_cfa_offset 16
04 .cfi_offset 6, -16
05 movq %rsp, %rbp
06 .cfi_def_cfa_register 6

```
07  movl %edi, -20(%rbp)
08  movq %rsi, -32(%rbp)
09  movq %rdx, -40(%rbp)
10  movl $0, -4(%rbp)
11  movl $0, -8(%rbp)
12  jmp .L26

.L27:

13  movl -8(%rbp), %eax
14  cltq
15  leaq 0(,%rax,4), %rdx
16  movq -32(%rbp), %rax
17  addq %rdx, %rax
18  movl (%rax), %edx
19  movl -8(%rbp), %eax
20  cltq
21  leaq 0(,%rax,4), %rcx
22  movq -40(%rbp), %rax
23  addq %rcx, %rax
24  movl (%rax), %eax
25  imull %edx, %eax
26  addl %eax, -4(%rbp)
27  addl $1, -8(%rbp)

.L26:

28  movl -8(%rbp), %eax
29  cmpl -20(%rbp), %eax
30  jl .L27
31  movl -4(%rbp), %eax
32  popq %rbp
33  .cfi_def_cfa 7, 8
34  ret
```

```
35  .cfi_endproc

## This code has two arrays
## It finds a[0]*b[0]+a[1]*b[1]+a[2]*b[2] and so on
.LFB3:
01  .cfi_startproc              # CFI Directive
02  pushq %rbp                  # save old base pointer
03  .cfi_def_cfa_offset 16      # CFI Directive
04  .cfi_offset 6, -16          # CFI Directive
05  movq %rsp, %rbp             # rbp <--rsp setting new stack base pointer
06  .cfi_def_cfa_register 6     # CFI Directive
07  movl %edi, -20(%rbp)        # Move value of edi to rbp-20 ([rbp-20]=edi).
08  movq %rsi, -32(%rbp)        # Move value of rsi to rbp-32 ([rbp-32]=rsi).  First Array
09  movq %rdx, -40(%rbp)        # Move value of rdx to rbp-40 ([rbp-40]=rdz) Second array
10  movl $0, -4(%rbp)           # Set value of rbp-4 to 0 ([rbp-4]=0) The sum
11  movl $0, -8(%rbp)           # Set value of rbp-8 to 0 ([rbp-8]=0) The index
12  jmp .L26                    # Unconditional jump to .L26
.L27:
13  movl -8(%rbp), %eax         # Move value of rbp-8 to eax   (eax=[rbp-8] )
14  cltq                        # make eax 64 bit and store loc in rax
15  leaq 0(,%rax,4), %rdx       # Move value of 4*rax into rdx (rdx = 4*rax)
16  movq -32(%rbp), %rax        # Move value of rbp-32 to rax (rax =[rbp-32] )
17  addq %rdx, %rax             # Add rdx and rax and store in rax (rax=rdx+rax)
18  movl (%rax), %edx           # Move memory address of rax to edx (edx=rax)
19  movl -8(%rbp), %eax         # Move value of rbp-8 to eax (eax=[rbp-28])
20  cltq                        # make eax 64 bit and store loc in rax
21  leaq 0(,%rax,4), %rcx       # Move value of 4*rax into rcx (rcx=4*rax)
22  movq -40(%rbp), %rax        # Move value of rbp-32 to rax (rax=[rbp-40])
```

```
23  addq %rcx, %rax          # Add rcx and rax and store in rax (rax=rax+rcx)
24  movl (%rax), %eax        # Move memory address of rax to eax (eax=rax)
25  imull %edx, %eax         # Multiply vale of eax and edx and store in eax (eax=eax*edx)
26  addl %eax, -4(%rbp)      # Add eax and rbp-4 and store in rbp-4 ([rbp-4]=[rbp-4]+eax)
27  addl $1, -8(%rbp)        # Add 1 to rbp-8 and store in rbp-8 ([rbp-8]=[rbp-8]+1)
.L26:
28  movl -8(%rbp), %eax      # Move value of rbp-8 to eax (eax=[rbp-8])
29  cmpl -20(%rbp), %eax     # Compare eax and rbp-20
30  jl .L27                  # Jump to L27 to if eax > [rbp-20]
31  movl -4(%rbp), %eax      # Move value of rbp-4 to eax
32  popq %rbp                # pop the register rbp
33  .cfi_def_cfa 7, 8        # CFI Directive
34  ret                      # pop return address from stack and jump there
35  .cfi_endproc             # CFI Directive
```

Comment:

Question 2

Complete

Mark 16.00 out of 16.00

Flag question

For the given input:

76+81/3

| 7 – 8 |

7 4 * 6y

A scanner generates following token streams:

====START of TOKENS==========

Number=76

ADD

Number=81

DIVIDE

Number=3

Newline

ABS

Number=7

SUBTRACT

Number=8

ABS

Newline

Number=7

Number=4

MULTIPLY

Number=6

Illegal input y

====END of TOKENS==========

Write down the corresponding Flex specifications of the scanner. No need to write auxiliary functions.

%{ /*

C Declarations and Definitions */

%}

/* Regular Expression Definitions */

NUM [0-9]+

NL [\n]

```
WS          [ \t]
%%
{NUM} { printf("Number=%s\n",yytext);  /* Keyword Rule */}
 "+"          { printf("ADD\n"); /* Operator Rule */ }
 "-"          { printf("SUBTRACT\n"); /* Operator Rule */ }
"/"          { printf("DIVIDE\n"); /* Operator Rule */ }
 "*"          { printf("MULTIPLY\n"); /* Operator Rule */ }
 "|"          { printf("ABS\n"); /* Operator Rule */ }
 {NL}         { printf("Newline\n"); /* Newline Rule */}
{WS}          {/* Ignore Whitespace */}
.             {printf("Illegal input %s\n", yytext); }
%%

/* C functions */
main() {
   printf("=======START of TOKENS ============\n");
   yylex(); /* Flex Engine */
   printf("=======END of TOKENS ============\n");
 }
```

Comment:

Finish review