CS 334 : COMPILER DESIGN LABORATORY **February 29, 2024**
Premises : Lab 1 [VIA] & Lab 3, 6 [VIB]

Lab Assignment 7 :
Objective : To write a lexical analyzer for X86-64 bit assembly language programs.

General Instructions :
1. Save all your work in this laboratory session in a separate directory, Lab7.
2. Test your scanner design with the test cases given to you. Once satisfied test with inputs designed by you.

Practice Problems

P1. The problem is to acquire skills to process assembly code of a contemporary real architecture, namely the X86-64 bit architecture, which most likely is the same as that of the server used for this course. You need some background information about x86-64 bit architecture which are being provided to you in the table below. Read the accompanied document, "x86-64_overview.pdf" for more details.

X86-64 bit Architecture	Clarification
16 registers of 64 bit	%rip (instruction pointer), %rsp (stack pointer), %rbp (base pointer on stack), "%rax, %rdi, %rsi, %rdx, %rbx, %r8, %r9, %r10, %r11, %r12, %r13, %r14, %r15
32 bit Registers	%esp (stack pointer), %ebp (base pointer on stack), "%eax, %edi, %esi, %edx, %ebx
Common instruction op codes	sub, mov, xor, lea, add, ret, cmp, imul, sal, shr, and, or ,not
Other assembly instructions	endbr64, call, ...
Assembly instructions vary in length from 1 to 8 bytes	We add a suffix {'b', 'w', 'l' or 'q'} to denote the length {1, 2, 4, 8} in bytes. For instance, movb, movw, movl, movq are all instances of the move instructions.
Jump instructions	jmp, je, jne, js, jns, jg, jge, jl, jle
Stack instructions	pushq, popq
Assembly code starting with a dot (.) followed with a "file" or a "string", or LX, where X is a alphanumeric character are of interest to us. All other lines starting with dot(.) are to be recognized and reported as Assembler Directives and ignored.	.file gives the name of the source C file .string is followed by a format string .LX where X ≥ 1 alphanumeric characters denotes a Label
Operand formats	$number specifies an immediate operand registers are used as operands other operand formats are given in the handout.

Your task is to

- examine the assembly code generated by gcc,
- identify patterns in the assembly code of interest,
- write a lex script that identifies the patterns, and
- finally produces some statistical data about the various elements in the assembly code.

The problem specification is illustrated through an example. The C program, "prog.c", is compiled using "$ gcc prog.c -O2 -S" to generate assembly code using level 2 optimization. Let the generated assembly code be named

as "prog-opt.s". **Check the assembly code on your server as it may be slightly different than that given below.**

```
// C program named as prog.c
int main()
{  int a[1000], i, j;
   int sum = 10000;
   for (i = 0; i < 100; i++) a[i] = i;
   for (i = 0; i < 10; i++)
   for (j = 0; j < i*i; j++)  a[i] = a[i]*a[j];
printf(" sum : %d \n", sum);
return 0;
}
```

The generated assembly code and the desired output after processing the assembly code are given in the following table.

X86-64 bit assembly program "prog-opt.s"	Desired Output after Processing "prog-opt.s"

.file "prog.c"
.text
.section .rodata.str1.1,"aMS",@progbits,1
.LC0:
.string " sum : %d \n"
.section .text.startup,"ax",@progbits
.p2align 4
.globl main
.type main, @function
main:
.LFB23:
.cfi_startproc
endbr64
subq $8, %rsp
.cfi_def_cfa_offset 16
movl $10000, %edx
movl $1, %edi
xorl %eax, %eax
leaq .LC0(%rip), %rsi
call __printf_chk@PLT
xorl %eax, %eax
addq $8, %rsp
.cfi_def_cfa_offset 8
ret
.cfi_endproc
.LFE23:
.size main, .-main
.ident "GCC: (Ubuntu 9.4.0-1ubuntu1~20.04.2) 9.4.0"
.section .note.GNU-stack,"",@progbits
.section .note.gnu.property,"a"
.align 8
.long 1f - 0f
.long 4f - 1f
.long 5
0:
	C Source File : .file "prog.c"
	Assembler Directive : .text
	Assembler Directive : .section .rodata.str1.1,"aMS",@progbits,1
	Label : .LC0
	DELIMITER :
	Format String : .string " sum : %d \n"
	Assembler Directive : .section .text.startup,"ax",@progbits
	Assembler Directive : .p2align 4
	Assembler Directive : .globl main
	Assembler Directive : .type main, @function
	Identifier : main
	DELIMITER :
	Label : .LFB23
	DELIMITER :
	Assembler Directive : .cfi_startproc
	Other Instructions : endbr64
	Quad Word Instructions : subq
	Immediate operand : $8
	DELIMITER ,
	64 bit Register operand : %rsp
	Assembler Directive : .cfi_def_cfa_offset 16
	Long Word Instructions : movl
	Immediate operand : $10000
	DELIMITER ,
	32 bit Register operand : %edx
	Long Word Instructions : movl
	Immediate operand : $1
	DELIMITER ,
	32 bit Register operand : %edi
	Long Word Instructions : xorl
	32 bit Register operand : %eax
	DELIMITER ,
	32 bit Register operand : %eax
	Quad Word Instructions : leaq
	Label : .LC0
	LEFT PAR : (

X86-64 bit assembly program "prog-opt.s"	Desired Output after Processing "prog-opt.s"
.string "GNU" 1: .align 8 .long 0xc0000002 .long 3f - 2f 2: .long 0x3 3: .align 8 4:	64 bit Register operand : %rip RIGHT PAR :) DELIMITER , 64 bit Register operand : %rsi Other Instructions : call Function call : __printf_chk@PLT Long Word Instructions : xorl 32 bit Register operand : %eax DELIMITER , 32 bit Register operand : %eax Quad Word Instructions : addq Immediate operand : $8 DELIMITER , 64 bit Register operand : %rsp Assembler Directive : .cfi_def_cfa_offset 8 Identifier : ret Assembler Directive : .cfi_endproc Label : .LFE23 DELIMITER : Assembler Directive : .size main, .-main Assembler Directive : .ident "GCC: (Ubuntu 9.4.0 1ubuntu1~20.04.2) 9.4.0" Assembler Directive : .section .note.GNU-stack,"",@progbits Assembler Directive : .section .note.gnu.property,"a" Assembler Directive : .align 8 Assembler Directive : .long 1f - 0f Assembler Directive : .long 4f - 1f Assembler Directive : .long 5 Number : 0 DELIMITER : Format String : .string "GNU" Number : 1 DELIMITER : Assembler Directive : .align 8 Assembler Directive : .long 0xc0000002 Assembler Directive : .long 3f - 2f Number : 2 DELIMITER : Assembler Directive : .long 0x3 Number : 3 DELIMITER : Assembler Directive : .align 8 Number : 4 DELIMITER : No of Assembler Directives: 23 No of in-line comment : 0 No of Strings : 0 Count of Immediate Operands Used 4 Count of 64 bit Registers Used 4 Count of 32 bit Registers Used 6 Count of Byte length Instructions Used 0 Count of Word length Instructions Used 0 Count of Long Word length Instructions Used 4 Count of Quad length Instructions Used 3

X86-64 bit assembly program "prog-opt.s"	Desired Output after Processing "prog-opt.s"
	Count of Stack Instructions Used 0

P2. Repeat P1 above with assembly code generated by gcc without the switch -O2. Examine whether the scanner of P1 is able to generate the desired output. In case, there are unrecognized characters, modify the lex script of P1 to make it work for this problem.

End of the Experiment 7

Attachments : 1. X86-64_overview.pdf
2. "prog.c"
3. "prog-opt.s" (to be used as sample only, since you are required to process the assembly code generated by gcc on your system