Middle East Technical University Department of Computer Engineering

**CENG 331** 

Section 3 Fall '2019-2020 Midterm I

- Duration: 100 minutes.
- Exam:
  - This is a **closed book**, **closed notes** exam.
  - No attempts of cheating will be tolerated. In case such attempts are observed, the students who took part in the act will be prosecuted. The legal code states that students who are found guilty of cheating shall be expelled from the university for a minimum of one semester!
  - Data sheet for some aspects of x86-64 assembly is available on the last page.
  - This booklet consists of 8 pages including this page. Check that you have them all!

Question 1	
Question 2	
Question 3	
Question 4	
Question 5	
$Total \Rightarrow$	

Name, SURNAME and ID  $\Rightarrow$ 

1 (15 pts)

### Warm-up.

- (2 pts) Write down the **gdb** (Gnu debugger) commands:
  - to display the 8-byte value at the top of the stack in hexadecimal format.



- to display the assembly code for a function foo.
- (4 pts) Write down a C function that would return 1 if the machine is Big Endian, and 0 if it is Little Endian. You are not allowed to use any library functions. Hint: Think of using char \*.

```
int isBigEndian(void){

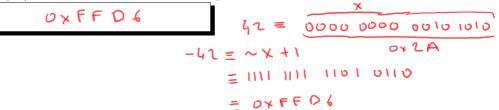
union { int i; charc; }e;

e.i=1;

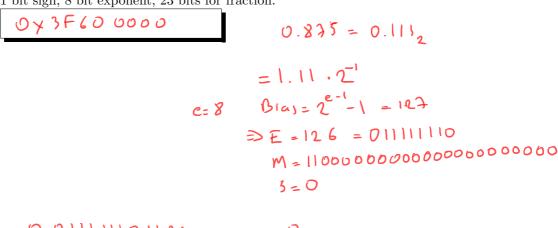
return(!e.cdoxi);
```

}

• (3 pts) Write the 16-bit short int representation of -42 in hexadecimal.



• (6 pts) Write the float representation of 0.875 in **hexadecimal**. Hint: Encoding: 1 bit sign, 8 bit exponent, 23 bits for fraction.



Name, SURNAME and ID  $\Rightarrow$ 

 $|\mathbf{2}|$ (20 pts)

> Consider the following m-bit floating-point representation based on the IEEE floatingpoint format:

- There is a sign bit-field in the most significant bit s.
- The next k bit-fields are the exponent exp. Bias=  $2^{k-1}$
- The last n bit-fields are the significand frac.  $\max = 2^{k-1} 2^{k-1} = 2^{k-1} 1$

In this format, a given numeric value V is encoded in the form  $V = (-1)^s \times M \times 2^E$ , where s is the sign bit, E is exponent after biasing, and M is the significand.

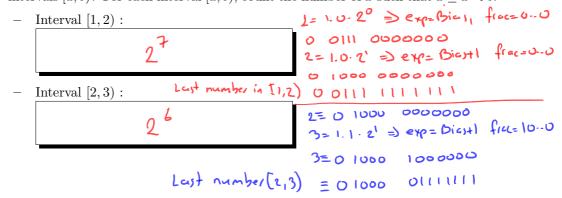
(5 pts) Give a formula for the largest even integer that can be represented exactly.

(2-2-n) 
$$\cdot 2^{k-1}$$
 if  $2^{k-1} > 0$ 

If  $2^{k-1} > 0$ 
 $1 + 2^{k-1} > 0$ 
 $1$ 

(5 pts) Give a formula for the smallest negative normalized value 
$$\begin{cases} 1 & 2^{k-1} \\ 2^{k-1} & 2^{k-1} \\ 1 & 0 & 0 \end{cases}$$

(10 pts) For k = 4 and n = 7, how many floating point numbers are in the following intervals [a, b)? For each interval [a, b), count the number of x such that  $a \le x < b$ .



Name, SURNAME and ID $\Rightarrow$	
3 (10 pts)	

Draw the memory layout of the structure  $\tt r2d2$  defined as below in a x86-84 Linux system:

```
typedef struct {
   char a[2];
  long b;
  float c;
  short d;
  long *e;
  short *f;
} r2d2;
```

(5 pts) Label the bytes with the names of the various fields and clearly mark the end of the struct. Use an X to denote space that is allocated in the struct as padding.

a	a	×	X	X	×	×	X	6	6	٩	<b>b</b>	b	b	6	7
C	C	C	<b>C</b>	d	9	X	$\times$	e	2	و	e	e	e	e	7
1	f	Ţ	f	₹	f	f	I I								

(5 pts) Redefine the struct to minimize its memory layout, and show the new layout.

```
typedef struct {
    _leng_b_;
    _leng_*L_;
    _shert_*f_;
    _fleat_C_;
    _shert_d_;
    _char_a(1);
} r2d2_new;
```

4	4	م	7	٨	b	ط	6	e	e	e	e	e	e	e	0
f	f	f	f	f	f	F	f	<b>C</b>	C	C	C	4	9	a	a

Name, SURNAME and ID  $\Rightarrow$ 4 (25 pts) You have the following assembly code get\_element: pushq %rbp %rsp, %rbp movq %rdi, -8(%rbp) movq-8(%rbp), %rax movq \$5, %rax salq - array +16+32i addq \$array+16, %rax 8(%rax), %rax movq -8(%rbp), %rdxmovq rdy + i -array (i): index \$2, %rdx salq rdx = 4i %rdx, %rax addq array(,%rax,8), %rax movq Rturn M[array + 32/1+ 8+]
array (i). c (array (i). index) popq %rbp ret

```
struct{
} array[_____
long get_element(long i){
  return(array [i]. c [array [i]. index
}
```

```
Name, SURNAME and ID \Rightarrow
```

(30 pts)

# Switch. Consider the following assembly code:

```
0x4004ed <starwars>
                                   %rbp
                           push
0x4004ee <starwars+1>
                          mov
                                  %rsp,%rbp
                                  %rdi,-0x18(%rbp)
0x4004f1 < starwars+4>
                          mov
0x4004f5 <starwars+8>
                                  %rsi,-0x20(%rbp)
                          mov
0x4004f9 <starwars+12>
                          mov
                                  %rdx,-0x28(%rbp)
0x4004fd <starwars+16>
                          movq
                                  $0x0,-0x8(%rbp)
0x400505 <starwars+24>
                                  $0x7,-0x28(%rbp)
                          cmpq
0x40050a <starwars+29>
                                  0x400557 <starwars+106>
                          ja
0x40050c <starwars+31>
                                  -0x28(%rbp),%rax
                          mov
                                  $0x3, %rax
0x400510 <starwars+35>
                          shl
0x400514 <starwars+39>
                          add
                                  $0x400608, %rax
0x40051a <starwars+45>
                                  (%rax),%rax
                          mov
0x40051d <starwars+48>
                          jmpq
                                  *%rax
0x40051f <starwars+50>
                                  -0x18(%rbp), %rax
                          mov
0x400523 <starwars+54>
                          mov
                                  -0x20(%rbp), %rdx
0x400527 <starwars+58>
                          add
                                  %rdx,%rax
0x40052a <starwars+61>
                                  %rax,-0x8(%rbp)
                          mov
0x40052e <starwars+65>
                          jmp
                                  0x40055f <starwars+114>
0x400530 <starwars+67>
                                  -0x20(%rbp), %rax
                          mov
0x400534 <starwars+71>
                                  $0x2a,%rax
                          add
0x400538 <starwars+75>
                                  %rax,-0x8(%rbp)
                          mov
0x40053c <starwars+79>
                          jmp
                                  0x40055f <starwars+114>
0x40053e <starwars+81>
                                  -0x20(%rbp),%rax
                          mov
0x400542 <starwars+85>
                          sub
                                  \frac{1}{2} x_1 - 0x18(\frac{1}{2} x^2)
0x400546 <starwars+89>
                                  -0x20(%rbp), %rax
                          mov
0x40054a <starwars+93>
                                  -0x18(\%rbp),\%rdx
                          mov
0x40054e <starwars+97>
                                  %rdx,%rax
                          xor
0x400551 <starwars+100>
                          mov
                                  %rax, -0x8(%rbp)
0x400555 <starwars+104>
                                  0x40055f <starwars+114>
                          jmp
0x400557 <starwars+106>
                                  0x14b,-0x8(%rbp)
                          movq
0x40055f <starwars+114>
                                  -0x8(%rbp), %rax
                          mov
0x400563 <starwars+118>
                                  %rbp
                          pop
0x400564 <starwars+119>
                          retq
```

## jump table:

0x400608: 0x40053e 0x400610: 0x400557 0x400618: 0x400557 0x400620: 0x400530 0x400628: 0x40051f 0x400630: 0x400557 0x400638: 0x400546 0x400640: 0x400530 Fill in the C template below based on the compiled code above:

```
long starwars(long a, long b, long c)
 long answer = -2;
 switch(c)
   case _4__:
answer = _b+ a__;
     break;
   case ____ :
    case __<del>_</del>_:
      answer = b+ 42;
      break;
    case _____:
      a = \underline{a - b};
      /* Fall through */
    case ___6_:
      answer = a^{\prime}b;
      break;
   default:
      answer = __331__;
 return answer;
}
```

#### THIS PAGE WILL NOT GRADED!

# Data sheet for x86-64 Assembly

• Arithmetic operations

```
addq Src,Dest Dest = Dest + Src
subq Src,Dest Dest = Dest - Src
imulq Src,Dest Dest = Dest * Src
salq Src,Dest Dest = Dest << Src Also called shll
sarq Src,Dest Dest = Dest >> Src Arithmetic
shrq Src,Dest Dest = Dest >> Src Logical
xorq Src,Dest Dest = Dest ^ Src
andq Src,Dest Dest = Dest & Src
orq Src,Dest Dest = Dest | Src
incq Dest Dest = Dest + 1
decq Dest Dest = Dest - 1
negq Dest Dest = - Dest
notq Dest Dest = ^ Dest
```

- cmpq Src2,Src1
  - cmpq b,a like computing a-b without setting destination
- testq Src2,Src1
  - testq b,a like computing a&b without setting destination
- Condition codes:
  - CF set if carry out from most significant bit
  - ZF set if t == 0
  - SF set if t < 0
  - OF set if two's complement overflow
- Jump operations

jmp	1	Unconditional
je	${ m ZF}$	Equal / Zero
jne	$\sim\!\!\mathrm{ZF}$	Not Equal / Not Zero
js	SF	Negative
jns	$\sim$ SF	Nonnegative
jg	$\sim$ (SF $^{\circ}$ OF)& $\sim$ ZF	Greater (Signed)
jge	$\sim$ (SF $^{\circ}$ OF)	Greater or Equal (Signed)
jl	(SF ^ OF)	Less (Signed)
jle	$(SF \cap OF) \mid ZF$	Less or Equal (Signed)
ja	$\sim$ CF& $\sim$ ZF	Above (unsigned)
jb	CF	Below (unsigned)