Name, SURNAME and ID ⇒ KEY

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 ⇒	

Name, SURNAME and ID ⇒

$\boxed{1}$ **(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. x/g $rsp
  - to display the assembly code for a function `foo`. disas 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 *`.

/* many possible answers */

```
int isBigEndian(void){
    union{ int i; char c;}e;
    e.i=1;
    return(!e.c&0x1);

}
```

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

0xFFD6

$42 \equiv \overbrace{0000\ 0000\ 0010\ 1010}^{x}$ 0x2A

$-42 \equiv \sim x + 1$

$\equiv 1111\ 1111\ 1101\ 0110$

$\equiv$ 0xFFD6

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

0x3F60 0000

$0.875 = 0.111_2$

$= 1.11 \cdot 2^{-1}$

$e=8$ $\quad Bias = 2^{e-1}-1 = 127$

$\Rightarrow E = 126 = 01111110$

$M = 1100000000000000000000000$

$s = 0$

$\underbrace{0\ 011}_{3}\ \underbrace{1\ 111}_{F}\ \underbrace{0\ 110}_{6}\ \underbrace{00\ldots 0}_{00000}$

0x 3 F 6 00000

Name, SURNAME and ID ⇒

$\boxed{2}$ **(20 pts)**

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

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

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.

$\underbrace{111\cdots10}_{\substack{exp \\ k}}$ $\underbrace{111\cdots1}_{\substack{frac \\ n}}$

$E = 2^{k-1}$ $M = 2 - 2^{-n}$

$V = (-1)^s \cdot (2 - 2^{-n}) \cdot 2^{2^{k-1}-1}$

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

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

  If $2^{k-1} - 1 < n$ frac: $\underbrace{111..1}_{2^{k-1}-1 \text{ bits}}000$

  $V = (2 - 2^{-2^{k-1}+1}) \cdot 2^{2^{k-1}-1} = 2^{2^{k-1}}$

- (5 pts) Give a formula for the smallest negative normalized value

  $-2^{-2^{k-1}+1}$

  $\overbrace{100\cdots01}^{exp}$ $\overbrace{00\cdots00}^{frac}$

  $-1.1.0 \cdot 2^{1-Bias}$

  $-2^{-2^{k-1}+1}$

- (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 $a \leq x < b$.

  – Interval $[1, 2)$:

  2^7

  $1 = 1.0 \cdot 2^0 \Rightarrow exp = Bias,\ frac = 0..0$

  0 0111 0000000

  $2 = 1.0 \cdot 2^1 \Rightarrow exp = Bias + 1\ \ frac = 0..0$

  0 1000 0000000

  Last number in $[1,2)$ 0 0111 1111111

  – Interval $[2, 3)$:

  2^6

  $2 \equiv$ 0 1000 0000000

  $3 = 1.1 \cdot 2^1 \Rightarrow exp = Bias + 1\ \ frac = 10..0$

  $3 \equiv$ 0 1000 1000000

  Last number$[2,3)$ $\equiv$ 0 1000 0111111

Name, SURNAME and ID ⇒

3 **(10 pts)**

Draw the memory layout of the structure `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	X	X	X	b	b	b	b	b	b	b	b
c	c	c	c	d	d	X	X	e	e	e	e	e	e	e	e
f	f	f	f	f	f	f	f								

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

```
typedef struct {
   long b;
   long *e;
   short *f;
   float c;
   short d;
   char a[2];
} r2d2_new;
```

b	b	b	b	b	b	b	b	e	e	e	e	e	e	e	e
f	f	f	f	f	f	f	f	c	c	c	c	d	d	a	a

Name, SURNAME and ID ⇒

4 **(25 pts)**

You have the following assembly code

```
get_element:
        pushq   %rbp
        movq    %rsp, %rbp
        movq    %rdi, -8(%rbp)       i
        movq    -8(%rbp), %rax       rax ← i
        salq    $5, %rax             rax ← 32 i
        addq    $array+16, %rax      rax ← array+16+32i
        movq    8(%rax), %rax        rax ← M[array+32i+24]
        movq    -8(%rbp), %rdx       rdx ← i
        salq    $2, %rdx             rdx ← 4i
        addq    %rdx, %rax           rax ← t+4i
        movq    array(,%rax,8), %rax
        popq    %rbp                 return M[array+32i+8t]
        ret
```

offset of the field

struct size 32 byte

array[i].index

array[i].c[array[i].index]

Fill in corresponding structure and code in the template shown below:

```
struct{
    ____long____c[3]______;
    ____long_____index____;
} array[__—__];   doesn't matter

long get_element(long i){
  return(array_[i].c[array[i].index]_________________);
}
```

Name, SURNAME and ID ⇒

$\boxed{5}$ **(30 pts)**

Switch. Consider the following assembly code:

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

jump table:
0x400608: 0x40053e
0x400610: 0x400557
0x400618: 0x400557
0x400620: 0x400530
0x400628: 0x40051f
0x400630: 0x400557
0x400638: 0x400546
0x400640: 0x400530
```

Name, SURNAME and ID ⇒

Fill in the C template below based on the compiled code above:

```
long starwars(long a, long b, long c)
{
  long answer = ___0____;
  switch(c)
  {
    case __4____:
      answer = __b+a______;
      break;
    case __3__ :
    case __7__:
      answer = __b+42_____;
      break;
    case __0__:
      a = __a-b______;
      /* Fall through */
    case __6__:
      answer = __a^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	ZF	Equal / Zero
jne	~ZF	Not Equal / Not Zero
js	SF	Negative
jns	~SF	Nonnegative
jg	~(SF ^ OF)&~ZF	Greater (Signed)
jge	~(SF ^ OF)	Greater or Equal (Signed)
jl	(SF ^ OF)	Less (Signed)
jle	(SF ^ OF) | ZF	Less or Equal (Signed)
ja	~CF&~ZF	Above (unsigned)
jb	CF	Below (unsigned)