Midterm 1

Hello everyone,

Midterm 1 grades are now being released from Crowdmark and the solution has been posted.
Check the solution. If you have any questions related to the marking of your Midterm 1, you can see me during my office hours.
Please, no email, no posting on Discussion forum.
Thank you! :)

Class scores distribution Show

My score
72.2% (32.5/45)

Q1 4

1. [6 marks] Consider the following syntactically correct C main function:

```
int main(int argc, char *argv[]) {

  int x = 0xDECAF000;
  short y = 0xCAFE;
  char z = y;
  // z = 0xFE

  if ( x > y ) printf("Cafe\n");
  // Your modified condition is:
  // if ( (unsigned short) x > y )


  if ( x < z ) printf("Decaf\n");
  // Your modified condition is:
  // if ( (unsigned int) x < z )


  return 0;
}
```

2.uncorrect second cast −2

Using only two casts (data type casts), and these two casts must be different, modify the conditions of the two conditional statements in the main function above such that only "Cafe" is printed on the computer monitor screen when the program executes on our *target machine*.

Q2 3

2. [3 marks] Consider the following syntactically correct C code fragment:

```
float aFloat = 13.75;
int sum = (int) aFloat + 0xFFFFFFFE;
```

Which value does the variable sum contain when the above C code fragment has executed on our *target machine*? Represent your answer as a hexadecimal number.

0x 0000 000B

Correct 3

Q3 3

3. [3 marks] Consider the following C code fragment:

```
char char1 = 101;
char char2 = 27 ;
char sumOfChar = char1 + char2;
```

Which value must be assigned to char2 in order for the sum of char1 and char2 to create a positive overflow? 27

Q4 3

4. [3 marks] Consider the disassembled function below. It was obtained by disassembling an executable containing this function on our *target machine*.

```
7002 :  f3 0f 1e fa +4          endbr64
7006 :  48 83 ec 08 +4          sub    $0x8,%rsp
700A:   48 8b 05 d9 2f 00 00 +7 mov    0x2fd9(%rip),%rax
7011:   48 85 c0 +3             test   %rax,%rax
7014:   74 02 +2                je     7016 <_init+0x16>
7016:   ff d0 +2                callq  *%rax
7018:   48 83 c4 08 +4          add    $0x8,%rsp
701C:   c3 +1                   retq
```

As you can see, six (6) of the memory addresses are missing. Complete the left column of the above disassembled function by adding these six (6) missing memory addresses.

7002 7003 7004 7005
7006 7007 7008 7009
700A 700B 700C 700D 700E 700F 7010
7011 7012 7013
7014 7015
7016 7017
7018 7019 701A 701B

701C

5. [Total marks: 9] Consider the following function `mystery` written in x86-64 assembly code, where the parameter `x` is in the register `%edi` and the parameter `y` is in the register `%esi`:

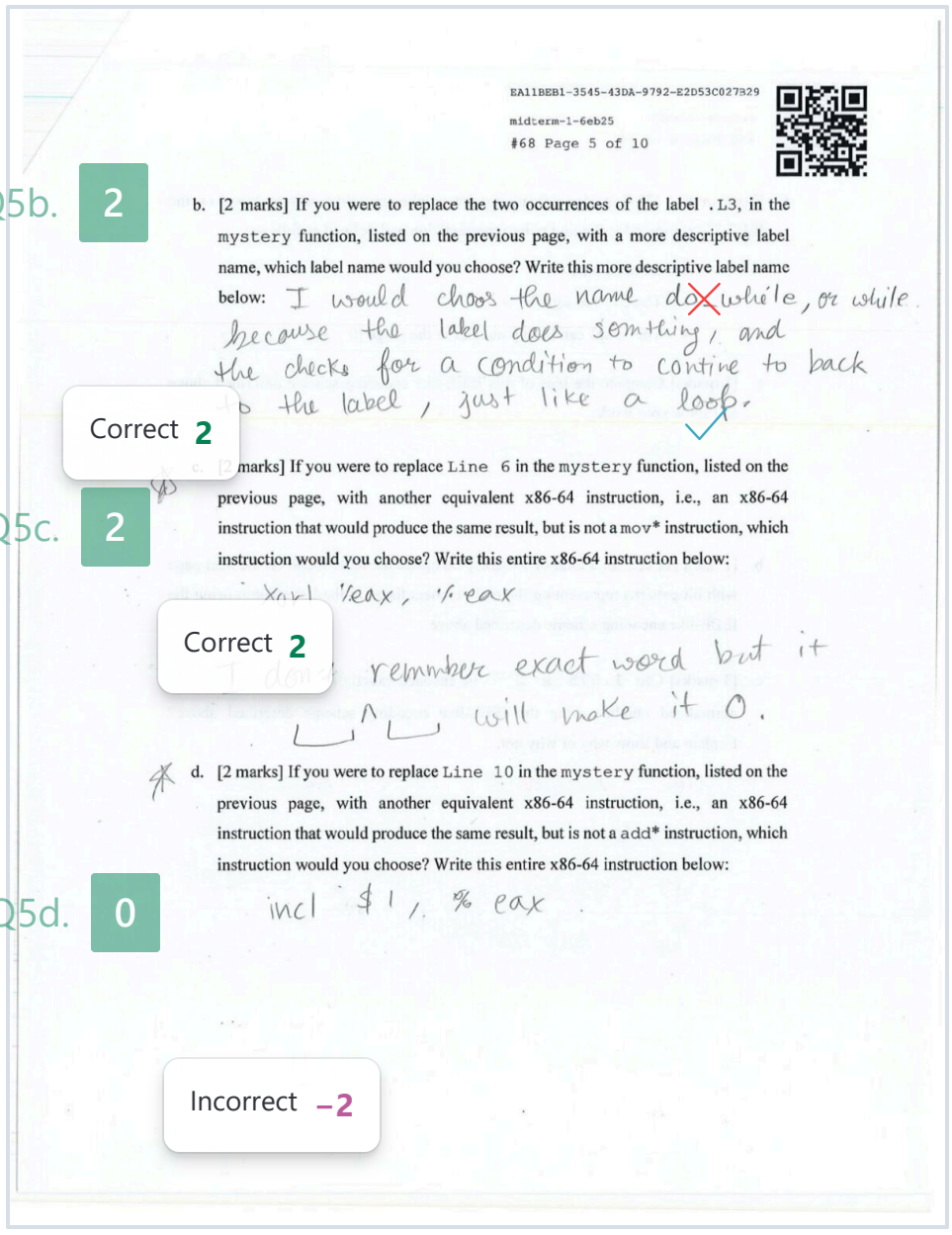
```
        .globl  mystery                                  Line 1
mystery:                                                 Line 2
        endbr64                                          Line 3
        testl   %esi,   %esi                             Line 4
        jle     .L4                                      Line 5
        movl    $0,     %eax                             Line 6
        movl    $1,     %edx                             Line 7
.L3:                                                     Line 8
        imull   %edi,   %edx                             Line 9
        addl    $1,     %eax                             Line 10
        cmpl    %esi,   %eax                             Line 11
        jne     .L3                                      Line 12
.L1:                                                     Line 13
        movl    %edx,   %eax                             Line 14
        ret                                              Line 15
.L4:                                                     Line 16
        movl    $1,     %edx                             Line 17
        jmp     .L1                                      Line 18
```

a. [3 marks] If we call this function as follows: `mystery( x, y )` where `x = 4` and `y = 3`, which value will it return?

it will return 64

Q5a. 3

Correct 3



EA11BEB1-3545-43DA-9792-E2D53C027B29
midterm-1-6eb25
#68 Page 5 of 10

Q5b. 2

b. [2 marks] If you were to replace the two occurrences of the label .L3, in the mystery function, listed on the previous page, with a more descriptive label name, which label name would you choose? Write this more descriptive label name below: I would choos the name do while, or while because the label does somting, and the checks for a condition to contine to back to the label, just like a loop.

Correct 2

Q5c. 2

c. [2 marks] If you were to replace Line 6 in the mystery function, listed on the previous page, with another equivalent x86-64 instruction, i.e., an x86-64 instruction that would produce the same result, but is not a mov* instruction, which instruction would you choose? Write this entire x86-64 instruction below:

xorl %eax, %eax

Correct 2

I don't remmber exact word but it will make it 0.

d. [2 marks] If you were to replace Line 10 in the mystery function, listed on the previous page, with another equivalent x86-64 instruction, i.e., an x86-64 instruction that would produce the same result, but is not a add* instruction, which instruction would you choose? Write this entire x86-64 instruction below:

Q5d. 0

incl $1, %eax

Incorrect −2

B8972FFA-582A-4454-B5FB-FDA85643890E

16 8 4 2 0
4 3 2 1 0

6. [Total marks: 12] Consider a floating point number encoding scheme based on the IEEE 754 standard for floating point representation and defined as follows:

- It uses 8 bits.
- There is one sign bit `s`.
- The `exp` can be any number in the range [0 .. 31]. 5 bits

Q6a. 2

a. [2 marks] Compute the bias of this IEEE-like encoding scheme described above and show your work:

bias = $2^{K-1} - 1 = 2^{5-1} - 1 = 2^4 - 1 = 16 - 1 = 15$

b. [1 mark per cell for a total of 7 marks] Complete the table found on the next page with bit patterns representing the real numbers listed in the left column using the IEEE-like encoding scheme described above.

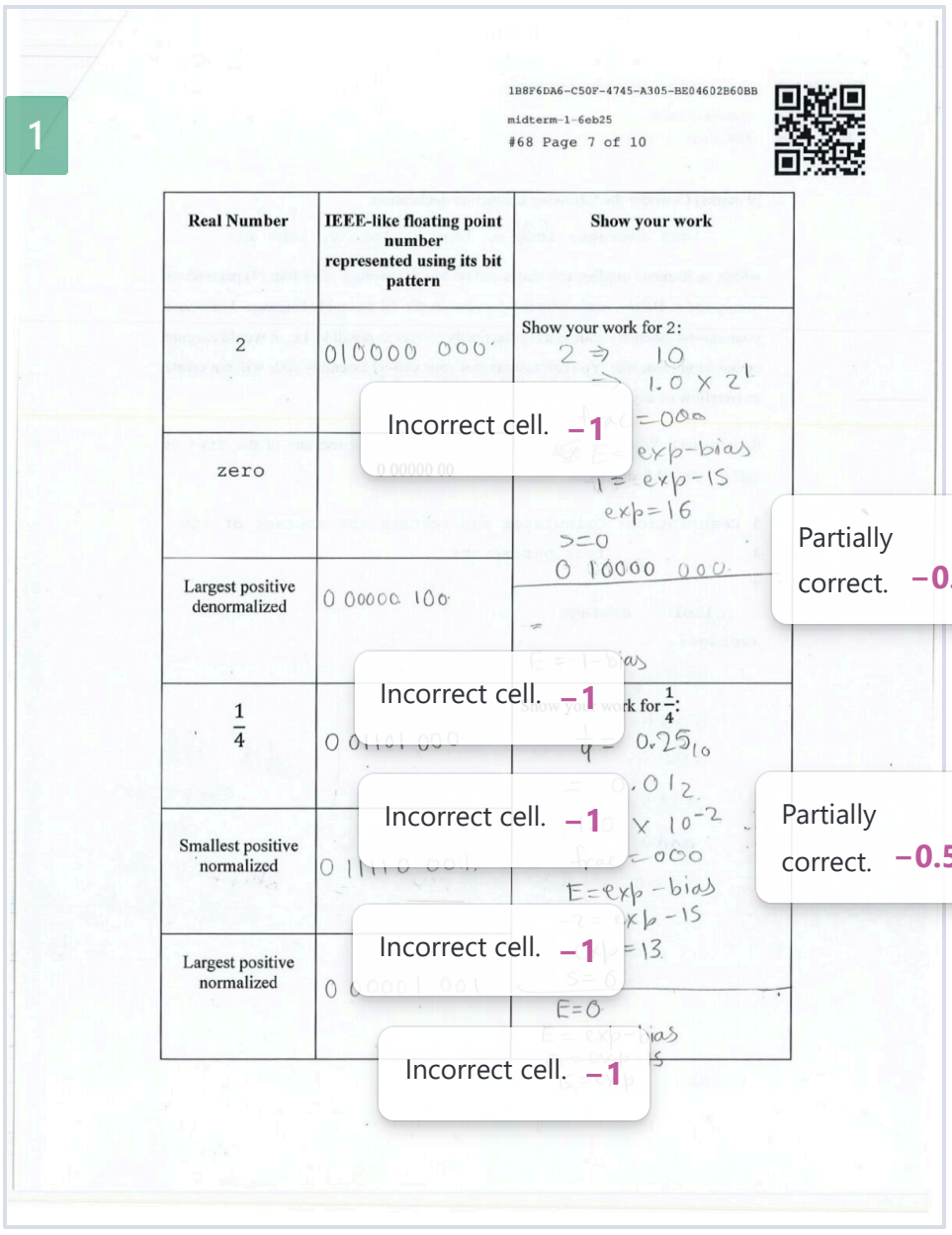
c. [3 marks] Can `1.875 x 2^-15` be encoded exactly (not approximated) as a normalized number using the IEEE-like encoding scheme described above? Explain and show why or why not.

Q6c. 0.5

no because -15 would make exp = 111111 which is +/- ∞, meaning cannot be represented.

Your conclusion is correct but your work is faulty. **0.5**

Q6b. 1


1B8F6DA6-C50F-4745-A305-BE04602B60BB
midterm-1-6eb25
#68 Page 7 of 10
Real Number
IEEE-like floating point number represented using its bit pattern
Show your work
2
zero
Largest positive denormalized
$\frac{1}{4}$
Smallest positive normalized
Largest positive normalized
Show your work for 2:
Incorrect cell. −1
Partially correct. −0.5
Incorrect cell. −1
Incorrect cell. −1
Partially correct. −0.5
Incorrect cell. −1
Incorrect cell. −1

Q7 9

7. [9 marks] Consider the following C function declaration:

```
long average( long w, long x, long y, long z);
```

(rdi, rsi, rdx, rcx)

which, as its name implies, calculates and returns the average of its four (4) parameters w, x, y and z. Below, implement this function in x86-64 assembly language. Make sure your x86-64 assembly code is as syntactically correct as possible, i.e., it would execute on our *target machine*. You can assume that your x86-64 assembly code will not create an overflow of any kinds.

Requirement: When answering this question, you cannot use any of the `div*` or `idiv*` x86-64 instructions.

```
# Description: Calculates and returns the average of its
#              four parameters.
#
   .globl    average
average:
    movq   $0, %rax
    addq   %rdi, %rax
    addq   %rsi, %rax
    addq   %rdx, %rax
    addq   %rcx, %rax
    shrq   $2, %rax

    ret
```

add all
Store in rax

shift right 2 = divide by 2^2 = 4

Table of x86-64 Registers

64-bit (quad)	32-bit (double)	16-bit (word)	8-bit (byte)		
63..0	31..0	15..0	15..8	7..0	
rax	eax	ax	ah	al	Return value
rbx	ebx	bx	bh	bl	Callee saved
rcx	ecx	cx	ch	cl	4th arg
rdx	edx	dx	dh	dl	3rd arg
rsi	esi	si		sil	2nd arg
rdi	edi	di		dil	1st arg
rbp	ebp	bp		bpl	Callee saved
rsp	esp	sp		spl	Stack pointer
r8	r8d	r8w		r8b	5th arg
r9	r9d	r9w		r9b	6th arg
r10	r10d	r10w		r10b	Caller saved
r11	r11d	r11w		r11b	Caller saved
r12	r12d	r12w		r12b	Callee saved
r13	r13d	r13w		r13b	Callee saved
r14	r14d	r14w		r14b	Callee saved
r15	r15d	r15w		r15b	Callee saved

Table of Powers of 2

Power of 2^x	Value	Power of 2^x	Value	
0	1			
1	2	-1	1/2	0.5
2	4	-2	1/4	0.25
3	8	-3	1/8	0.125
4	16	-4	1/16	0.0625
5	32	-5	1/32	0.03125
6	64	-6	1/64	0.015625
7	128	-7	1/128	0.0078125
8	256	-8	1/256	0.00390625
9	512	-9	1/512	0.001953125
10	1024	-10	1/1024	0.0009765625
11	2048			
12	4096			
13	8192			
14	16384			
15	32768			
16	65536			

Table of x86-64 Jumps

Instruction	Condition	Description
`jmp`	always	Unconditional jump
`je / jz`	`ZF`	Jump if equal / zero
`jne / jnz`	`~ZF`	Jump if not equal / not zero
`js`	`SF`	Jump if negative
`jns`	`~SF`	Jump if nonnegative
`jo`	`OF`	Jump if overflow
`jno`	`~OF`	Jump if not overflow
`jg / jnle`	`~(SF ^ OF) & ~ZF`	Jump if greater (signed >)
`jge / jnl`	`~(SF ^ OF)`	Jump if greater or equal (signed ≥)
`jl / jnge`	`SF ^ OF`	Jump if less (signed <)
`jle / jng`	`(SF ^ OF) | ZF`	Jump if less or equal (signed ≤)
`ja / jnbe`	`~CF & ~ZF`	Jump if greater (unsigned >)
`jae / jnb`	`~CF`	Jump if greater or equal (unsigned ≥)
`jb / jnae / jc`	`CF`	Jump if less (unsigned <)
`jbe / jna / jnc`	`CF | ZF`	Jump if less or equal (unsigned ≤)