ECE220: Computer Systems and Programming

Past Exam

1. This is a closed book closed notes exam
2. You may not use any personal electronic devices, such as cellphone and calculator
3. Absolutely no interaction between students is allowed
4. Illegible handwriting will be graded as incorrect

Name:________________________________

NetID:________________________________

Room:________________________________

Question 1 (36 points): _______________

Question 2 (22 points): A_______; B_______; C_______

Question 3 (42 points): 3.1_______; 3.2_______; 3.3_______; 3.4_______

Total Score: _______________

Problem 1 (30 points): Check for Pythagorean Theorem

In this question, you will complete a program which checks the following condition:

$C^2 = A^2 + B^2$, where A, B and C are three input numbers

More details:

1. The inputs (A, B, C) are already stored in a **stack** as shown below and they are all positive numbers and small enough that there will be no arithmetic overflow even when squared.
2. R6 is the stack pointer, pointing at the next available location on the stack. You should implement stack operations directly using R6 rather than calling the PUSH and POP subroutines as in MP2.
3. The subroutine SQUARE takes one input from R4 and sets the output in R5 (R5 = R4*R4).
4. If $C^2 = A^2 + B^2$, R0 is set to 1. Otherwise, R0 is set to 0 (this part has been done for you).
5. Push the value in R0 onto the stack.
6. Assume no stack overflow nor underflow.

x3FFA		
x3FFB		
x3FFC		
x3FFD		← **R6**
x3FFE	A	
x3FFF	B	
x4000	C	

Each line of code is worth 2 points.

```
.ORIG x3000
; Pop A into R1 in next two lines

______________________________ ; line 1

______________________________ ; line 2
; Pop B into R2 in next two lines

______________________________ ; line 3

______________________________ ; line 4
; Pop C into R3 in next two lines

______________________________ ; line 5

______________________________ ; line 6
AND R0, R0, #0          ; Set R0 = 0

; Set R1 = R1 * R1 in following three lines
; Must call SQUARE subroutine

______________________________ ; line 7

______________________________ ; line 8

______________________________ ; line 9

; Set R2 = R2 * R2 in following three lines
; Must call SQUARE subroutine

______________________________ ; line 10

______________________________ ; line 11

______________________________ ; line 12

; Set R3 = R3 * R3 in following 3 lines
; Must call SQUARE subroutine

______________________________ ; line 13

______________________________ ; line 14

______________________________ ; line 15
```

```
NOT R3, R3        ; R3 = - C^2
ADD R3, R3, #1

ADD R1, R1, R2    ; R1 = A^2 + B^2
ADD R1, R1, R3    ; R1 = A^2 + B^2 - C^2
BRnp CHECK_DONE
ADD R0, R0, #1

CHECK_DONE
; Push the value in R0 onto the stack in following
two lines

______________________________ ; line 16

______________________________ ; line 17

; Stop the program

______________________________ ; line 18


; SQUARE subroutine
; Input   : R4
; Output : R5 = R4*R4
SQUARE
  ST R3, Save_R3
  AND R5, R5, #0
  ADD R3, R4, #0
LOOP
  ADD R5, R5, R3
  ADD R4, R4, #-1
  BRp LOOP

  LD R3, Save_R3
RET

Save_R3 .BLKW #1
```

Problem 2: Debug a Program

Danny implemented a program that reverses the contents of a sequence of memory locations. That is, contents of x4000 is swapped with that of x4009, x4001 with x4008, x4002 with x4007, and so on. The Main program calls the REVERSE subroutine which in turn uses a SWAP subroutine to swap the contents of two memory locations. REVERSE is supposed to return to the Main program once the value of the starting address (in R0) becomes larger than that of the ending address (in R1).

The program compiled fine but it falls into an infinite loop. Help Danny find and fix the bug.

```
Line
0        .ORIG x3000
         ;  Main program

1        LD R0, START
2        LD R1, END
3        JSR REVERSE
4        HALT

       ; Reverse Subroutine
       ; Input R0, R1; Output - NONE
5       REVERSE
6                ST R0, SAVER0_REVERSE
7                ST R1, SAVER1_REVERSE
8                ST R2, SAVER2_REVERSE
9                ST R3, SAVER3_REVERSE
10        RLOOP
11               JSR SWAP
12               ADD R0, R0, #1
13               ADD R1, R1, #-1
14               NOT R2, R0
15               ADD R2, R2, #1
16               ADD R3, R2, R1
17               BRp RLOOP
18               LD R0, SAVER0_REVERSE
19               LD R1, SAVER1_REVERSE
20               LD R2, SAVER2_REVERSE
21               LD R3, SAVER3_REVERSE
22               RET

       ; Swap Subroutine
       ; Input R0, R1; Output - NONE
23      SWAP
24               ST R2, SAVER2_SWAP
25               ST R3, SAVER3_SWAP
```

```
26            LDR R2, R0, #0
27            LDR R3, R1, #0
28            STR R2, R1, #0
29            STR R3, R0, #0
30            LD R2, SAVER2_SWAP
31            LD R3, SAVER3_SWAP
32            RET

33      START .FILL    x4000
34      END   .FILL    x4009
35      SAVER0_REVERSE      .BLKW  #1
36      SAVER1_REVERSE      .BLKW  #1
37      SAVER2_REVERSE      .BLKW  #1
38      SAVER3_REVERSE      .BLKW  #1
39      SAVER2_SWAP.BLKW  #1
40      SAVER3_SWAP.BLKW  #1
41      .END
```

Part A (7 points). Danny suspects that the RLOOP on lines 10–17 never finishes; i.e., line 18 is never reached. Is he correct in his suspicion or not? Justify your answer (use no more than 30 words).

Your Answer:

Part B (7 points). Why does the program get into an infinite loop and between which line(s)? (use no more than 30 words)

Your Answer:

____

____

____

____

Part C (8 points). Please provide a solution for the problem by modifying or inserting instructions. Maximum 4 lines are allowed. Please be very specific about the lines you are modifying/inserting.

e.g., between lines 30 and 31, insert ADD R5, R5, R5
or
change line 30 to ADD R5, R5, R5

Your Answer:

____

____

____

____

Problem 3: Concepts

3.1 (2 points) LC-3 is a 16-bit system. Now suppose you have a 32-bit system, what is its address space (number of unique memory locations)?

Your Answer:

____

3.2 (5 points) What are some benefits of using a subroutine in LC-3? (Choose all that apply)

a) Hide program details from others
b) Separate program from underlying hardware
c) Make code more organized
d) Create libraries for others to use
e) Make debugging easier

Your Answer:

____

3.3 The following piece of code is given to you to handle I/O. Does it describe interrupt-driven I/O or polling I/O? Justify your choice using no more than 30 words.

```
.ORIG x3000

KCHECK        LDI R0, KBSR
              BRzp KCHECK
              LDI R0, KBDR
DCHECK        LDI R1, DSR
              BRzp DCHECK
              STI R0, DDR
HALT

KBSR   .FILL  xFE00
KBDR   .FILL  xFE02
DSR    .FILL  xFE04
DDR    .FILL  xFE06
.END
```

Choose one (2 points):

Interrupt-driven I/O   Polling I/O

Justify your choice using no more than 30 words (3 points):

____

____

____

____

3.4 Consider the evaluation of postfix expression using a stack in MP2. Assume that all operands in the expression are single-digit. You need to apply the same method used in MP2 to evaluate the postfix expression given below. When an operand is encountered, it should be pushed onto the stack. When an operator is encountered, two numbers should be popped off the stack to perform the arithmetic operation, and the result should be pushed onto the stack.

Postfix expression: 487-3+/2*

Part A: find the result of this postfix expression

Your Answer: 487-3+/2* equals ______________________________ (5 points)

Part B: Given that the stack starts at memory location x4000. Fill out the stack at the following instances. Assume that the stack is empty at the beginning, be sure to include all the values at each instance.

(1) Right before the first arithmetic operation (i.e., before its operands are popped off the stack). (5 points)

x3FFA	
x3FFB	
x3FFC	
x3FFD	
x3FFE	
x3FFF	
x4000	

(2) Right after the first arithmetic operation (i.e., after its result has been pushed onto the stack). (5 points)

x3FFA	
x3FFB	
x3FFC	
x3FFD	
x3FFE	
x3FFF	
x4000	

Postfix expression: 487-3+/2*

(3) Right before the second arithmetic operation (i.e., before its operands are popped off the stack). (5 points)

x3FFA	
x3FFB	
x3FFC	
x3FFD	
x3FFE	
x3FFF	
x4000	

(4) Right after the second arithmetic operation (i.e., after its result has been pushed onto the stack). (5 points)

x3FFA	
x3FFB	
x3FFC	
x3FFD	
x3FFE	
x3FFF	
x4000	

(5) Right after the third arithmetic operation (i.e., after its result has been pushed onto the stack). (5 points)

x3FFA	
x3FFB	
x3FFC	
x3FFD	
x3FFE	
x3FFF	
x4000	

Table E.2 **The Standard ASCII Table**

ASCII			ASCII			ASCII			ASCII		
Character	Dec	Hex	Character	Dec	Hex	Character	Dec	Hex	Character	Dec	Hex
nul	0	00	sp	32	20	@	64	40	`	96	60
soh	1	01	!	33	21	A	65	41	a	97	61
stx	2	02	"	34	22	B	66	42	b	98	62
etx	3	03	#	35	23	C	67	43	c	99	63
eot	4	04	$	36	24	D	68	44	d	100	64
enq	5	05	%	37	25	E	69	45	e	101	65
ack	6	06	&	38	26	F	70	46	f	102	66
bel	7	07	'	39	27	G	71	47	g	103	67
bs	8	08	(	40	28	H	72	48	h	104	68
ht	9	09	)	41	29	I	73	49	i	105	69
lf	10	0A	*	42	2A	J	74	4A	j	106	6A
vt	11	0B	+	43	2B	K	75	4B	k	107	6B
ff	12	0C	,	44	2C	L	76	4C	l	108	6C
cr	13	0D	-	45	2D	M	77	4D	m	109	6D
so	14	0E	.	46	2E	N	78	4E	n	110	6E
si	15	0F	/	47	2F	O	79	4F	o	111	6F
dle	16	10	0	48	30	P	80	50	p	112	70
dc1	17	11	1	49	31	Q	81	51	q	113	71
dc2	18	12	2	50	32	R	82	52	r	114	72
dc3	19	13	3	51	33	S	83	53	s	115	73
dc4	20	14	4	52	34	T	84	54	t	116	74
nak	21	15	5	53	35	U	85	55	u	117	75
syn	22	16	6	54	36	V	86	56	v	118	76
etb	23	17	7	55	37	W	87	57	w	119	77
can	24	18	8	56	38	X	88	58	x	120	78
em	25	19	9	57	39	Y	89	59	y	121	79
sub	26	1A	:	58	3A	Z	90	5A	z	122	7A
esc	27	1B	;	59	3B	[	91	5B	{	123	7B
fs	28	1C	<	60	3C	\	92	5C	|	124	7C
gs	29	1D	=	61	3D	]	93	5D	}	125	7D
rs	30	1E	>	62	3E	^	94	5E	~	126	7E
us	31	1F	?	63	3F	_	95	5F	del	127	7F

NOTES: RTL corresponds to execution (after fetch!); JSRR not shown

ADD	0001 DR SR1 0 00 SR2	ADD DR, SR1, SR2	DR ← SR1 + SR2, Setcc
ADD	0001 DR SR1 1 imm5	ADD DR, SR1, *imm5*	DR ← SR1 + SEXT(imm5), Setcc
AND	0101 DR SR1 0 00 SR2	AND DR, SR1, SR2	DR ← SR1 AND SR2, Setcc
AND	0101 DR SR1 1 imm5	AND DR, SR1, *imm5*	DR ← SR1 AND SEXT(imm5), Setcc
BR	0000 n z p PCoffset9	BR{nzp} *PCoffset9*	((n AND N) OR (z AND Z) OR (p AND P)): PC ← PC + SEXT(PCoffset9)
JMP	1100 000 BaseR 000000	JMP BaseR	PC ← BaseR
JSR	0100 1 PCoffset11	JSR *PCoffset11*	R7 ← PC, PC ← PC + SEXT(PCoffset11)
TRAP	1111 0000 trapvect8	TRAP *trapvect8*	R7 ← PC, PC ← M[ZEXT(trapvect8)]

LD	0010 DR PCoffset9	LD DR, *PCoffset9*	DR ← M[PC + SEXT(PCoffset9)], Setcc
LDI	1010 DR PCoffset9	LDI DR, *PCoffset9*	DR ← M[M[PC + SEXT(PCoffset9)]], Setcc
LDR	0110 DR BaseR offset6	LDR DR, BaseR, *offset6*	DR ← M[BaseR + SEXT(offset6)], Setcc
LEA	1110 DR PCoffset9	LEA DR, *PCoffset9*	DR ← PC + SEXT(PCoffset9), Setcc
NOT	1001 DR SR 111111	NOT DR, SR	DR ← NOT SR, Setcc
ST	0011 SR PCoffset9	ST SR, *PCoffset9*	M[PC + SEXT(PCoffset9)] ← SR
STI	1011 SR PCoffset9	STI SR, *PCoffset9*	M[M[PC + SEXT(PCoffset9)]] ← SR
STR	0111 SR BaseR offset6	STR SR, BaseR, *offset6*	M[BaseR + SEXT(offset6)] ← SR