ZJU-UIUC Institute
First Midterm Exam, ECE 220

Tuesday 25 October 2022

Name (Pinyin and Hanzi):
SOLUTION IN RED
ZJU ID:

- **Make sure that your exam booklet has exactly ELEVEN pages.**
- **Write your ZJU ID at the top of each page.**
- **Do not tear the exam apart other than to remove the reference sheet that includes RTL for LC-3 instructions (except JSRR) and a table of ASCII characters.**
- **Copies of Patt & Patel's Appendix A are also available during the exam. Raise your hand if you need that.**
- **This is a closed book exam. You may <u>not</u> use a calculator.**
- **You are allowed ONE A4 sheet of notes (both sides).**
- **YOU MAY NOT USE EXTRA PAPER! WRITE ON THE EXAM!**
- **Absolutely no interaction between students is allowed.**
- **Show all work, write neatly, and clearly indicate any assumptions that you make.**
- **Challenge problems are marked with ***.**
- **Don't panic, and good luck!**

Correct Room	1 point	______________________
Problem 1	24 points	______________________
Problem 2	25 points	______________________
Problem 3	25 points	______________________
Problem 4	25 points	______________________

Total	100 points	______________________

Problem 1 (24 points): Short Answer

Part A. (8 points) For each statement below, circle "**T**" if the statement is true and circle "**F**" if false. If you don't know the answer, circle "**IDK**". Each correct choice is worth +2 points; each incorrect choice is worth −1 point; each "**IDK**" is worth +0.5 point.

IDK **T** **F** Unlike many modern ISAs, LC-3 has no instruction opcodes for stack instructions.

IDK **T** **F** Like many modern ISAs, LC-3 uses memory-mapped I/O.

IDK **T** **F** The LC-3 memory has an address space of 2^{16} (65,536) locations, and **not** all addresses are used for memory locations. **Some of them are used as I/O ports.**

IDK **T** **F** ~~All~~ **Most** interactions between an LC-3 processor and I/O devices are **a**synchronous.

Part B. (5 points) While Keyi was doing MP2, he wrote the following snippet of code to print the five events from one time slot using **PRINT_CENTERED** (recall: a string address is passed to the subroutine in R1).

```
            ; R3 points to the schedule
            AND    R6,R6,#0     ; R6 counts days
            ADD    R6,R6,#-5
PRINT_ONE_LINE
            LD     R0,VLINE
            OUT
            LDR    R1,R3,#0     ; read the name of the event
            JSR    PRINT_CENTERED
            ADD    R3,R3,#1     ; point to next schedule day
            ADD    R6,R6,#1
            BRn    PRINT_ONE_LINE

VLINE .FILL x7C                 ; ASCII vertical line character
```

Keyi did not get full points from printing schedule, but he did pass some tests. **USING NO MORE THAN TWENTY WORDS** to describe what those tests look like. Assume that **R3** is set correctly and that Keyi's **PRINT_CENTERED** has no bugs.

Schedules with no empty slots.

Problem 1, continued:

Part C. (11 points) The **`MYINPUT`** subroutine below was written partially by Tianyu. **`MYINPUT`** waits for a key to be pressed, then stores the key pressed in ASCII to **`R0`**.

```
01   MYINPUT  LDI    R0,REG1______________ ; Test for keyboard input
02            LD     R1,MAGIC
03            AND    R0,R0,R1
04            BRzp   MYINPUT______________
05            LDI    R0,REG2______________ ; Store the input to R0
06            RET
07     REG1   .FILL  xFE00     ; KBSR
08     REG2   .FILL  xFE02     ; KBDR
09     MAGIC  .FILL  x8000
```

C.1 **(4 points)** Fill in the blanks to complete the code. Registers **`R2`**, **`R3`**, **`R4`**, **`R5`**, and **`R6`** must be callee-saved. **Do not use TRAPs nor subroutine calls.**

C.2 **(2 points)** xFE00 is the (address / content / memory) of KBSR. **CIRCLE EXACTLY ONE ANSWER.**

C.3 **(5 points)** After reviewing Tianyu's code, Kaiyuan asserts that lines #02 and #03 are unnecessary—that the code works even if both are deleted. **USING NO MORE THAN FORTY WORDS**, tell us if Kaiyuan is correct and explain your reason.

Yes. The ZP condition code only checks KBSR[15], which is KBSR ready bit.

Problem 2 (25 points): Stack Usage

Part A. (20 points) In lecture, you were introduced to a data structure called a stack. A stack can be described as LIFO (Last-In-First-Out) because the element that is pushed last is popped first. In this problem, you must use two stacks to implement another data structure called a queue. A queue is FIFO (First-In-First-Out), meaning that the element that is pushed first is also popped first. A queue data structure behaves in the same way as queues that you encounter in life. For instance, at a bus stop!

To implement a queue using two stacks, Prof. Loskot wants you to use the following approach:

- Call the two stacks **`S1`** and **`S2`**, and let **`R5`** and **`R6`** always points to the tops of **`S1`** and **`S2`**, respectively.
- To initialize the queue, point **`R5`** to x6000, and point **`R6`** to x7000.
- To push an element into the queue, push the element to the top of **`S1`**.
- To pop an element from the queue, if **`S2`** is not empty, directly pop an element from **`S2`**. Otherwise, pop every element from **`S1`** and push them to **`S2`** (one by one), then pop from **`S2`**.

`QUEUE_INIT`, **`QUEUE_POP`**, and **`QUEUE_PUSH`** must be implemented correctly to enable correct functioning of the queue, while **`QUEUE_INIT`** and **`QUEUE_PUSH`** are provided to you:

```
; QUEUE_INIT - initializes the queue
; Input: None
; Output: None
; All registers other than R5, R6, and R7 are callee-saved.
; R5 is set to x6000, and R6 is set to x7000.
QUEUE_INIT  LD    R5,S1_BTM
            LD    R6,S2_BTM
            RET
S1_BTM      .FILL x6000
S2_BTM      .FILL x7000

; QUEUE_PUSH - pushes a 16-bit 2's complement value into the queue
; Input: R0 - the 16-bit 2's complement value to be pushed
; Output: None
; All registers other than R5 and R7 are callee-saved.
; R5 changes to reflect push.  Ignores possibility of stack overflow.
QUEUE_PUSH  ADD   R5,R5,#-1
            STR   R0,R5,#0
            RET
```

You must write the following subroutine:

```
; QUEUE_POP - pops a 16-bit 2's complement from the head of the queue
; Input: None
; Output: R0 - the 2's complement value from the head of the queue
; All registers other than R0, R5, R6, and R7 are callee-saved.
; R5 and R6 change as necessary.  Ignores possibility of stack overflow
; and queue underflow (empty queue).
```

***** WRITE YOUR CODE ON THE NEXT PAGE *****

Use **NO MORE THAN THIRTY MEMORY LOCATIONS**, including storage for any data needed.

**** Using more memory than THIRTY LOCATIONS will earn NO CREDIT. ****

(Include comments for more partial credit.)

Problem 2, continued: **(QUEUE_POP specifications duplicated for your convenience)**

```
; QUEUE_POP - pops a 16-bit 2's complement from the head of the queue
; Input: None
; Output: R0 - the 2's complement value from the head of the queue
; All registers other than R0, R5, R6, and R7 are callee-saved.
; R5 and R6 change as necessary.  Ignores possibility of stack overflow
; and queue underflow (empty queue).
```

```
QUEUE_POP   ST  R1,R1_STR      ; save R1
            LD  R1,NEG_X7000   ; is S2 empty?
            ADD R0,R1,R6
            BRn POP_S2         ; S2 is not empty, pop from S2!
            LD  R1,NEG_X6000   ; R1 always holds -x6000
POPNPUSH    ADD R0,R1,R5       ; is S1 empty now?
            BRz POP_S2         ; S1 is empty, pop from S2!
            LDR R0,R5,#0       ; pop from S1 to R0
            ADD R5,R5,#1
            ADD R6,R6,#-1      ; push R0 into S2
            STR R0,R6,#0
            BR  POPNPUSH
POP_S2      LDR R0,R6,#0       ; pop from S2 to R0 and return
            ADD R6,R6,#1
            LD  R1,R1_STR      ; restore R1
            RET

R1_STR      .BLKW #1
NEG_X6000   .FILL xA000        ; -x6000, used to check if S1 is empty
NEG_X7000   .FILL x9000        ; -x7000, used to check if S2 is empty
```

Problem 2, continued:

Part B. (5 points) Jack loves stacks! When you pop a value from a stack, is the value still stored in memory? Explain your answer **USING NO MORE THAN THIRTY WORDS**.

Yes. Pointer moves down, while the bits remain in place but are no longer on the stack.

Problem 3 (25 points): Basics of C Programming

Part A. (15 points) Yingying wants to play "Among Us" with Chuxuan. Chuxuan sends a program to Yingying, whose output indicates a place where they should meet up. Unfortunately, Yingying's laptop is not functioning well. Can you help her?

```
01    #include <stdint.h>
02    #include <stdio.h>
03
04    int8_t X = 6; // Part A.3 comments this line out.
05
06    void foo() {
07        static int8_t Y = 2;
08        if (X > 3) { X = 3; }
09        printf("%c%c", 'A' + (++X), 'A' + (Y--));
0A    }
0B
0C    int main() {
0D        int8_t X = 2;
0E        foo();
0F        ++X;
10        foo();
11        return 0;
12    }
```

A.1 **(8 points)** Where should Yingying go? Write the output of the program.

____**ECEB**____

____

A.2 **(4 points)** Yingying wants to know where variables are stored. **CIRCLE EXACTLY ONE ANSWER** for each variable.

Y on line #07 is stored (in the heap / on the stack / in the global data area).

X on line #0D is stored (in the heap / on the stack / in the global data area).

A.3 **(3 points)** Chuxuan decides to remove line #04. **USING FIVE WORDS OR FEWER,** describe what happens when she does.

______**The code no longer compiles.**____

Problem 3, continued:

Part B. (10 points) As part of his Math 347 homework, Howie must calculate the *i*-th element of the *Fibonacci Sequence* multiple times. After finishing these unremitting calculations, he recalls Prof. Lumetta's advice: *Turn drudge work into opportunities for invention.* Howie decides to write a program to help him to calculate *i*-th element of the sequence.

For your reference, Fibonacci Sequence is formally defined by:

$$F_0 = 1,\ F_1 = 1,\ F_n = F_{n-1} + F_{n-2}\ (n > 1),$$

Howie's program appears as follows:

```
01    #include <stdint.h>
02    #include <stdio.h>
03
04    void fib(int32_t n) {
05        if (n < 1) return 1;
06        return fib(n - 1) + fib(n - 2);
07    }
08
09    int main() {
0A        int32_t n, result;
0B        if (1 != scanf("%d", &n) || n < 0) {
0C            printf("enter a non-negative number!\n");
0D            return 3;
0E        }
0F        result = fib(n);
10        printf("%d\n", &result);
11        return 0;
12    }
```

Sadly, Howie's program does not work as expected, but Howie is too lazy to debug. So, he decides to let the smart ECE220 students help him. Please describe **EACH OF THE THREE BUGS** in Howie's program with their line numbers and indicate how to fix them. Howie really appreciates your help!

Bug 1: line #04. `fib` should return `int32_t` instead of `void`.
Bug 2: line #05. The condition should be `(n <= 1)`.
Bug 3: line #10. Remove the ampersand.

Problem 4 (25 points): Understanding Compiled C Code

Part A. (18 points) The LC-3 code below corresponds to the output of a non-optimizing compiler for the C function **foo**.

```
FOO      ADD R6,R6,#-6
         STR R5,R6,#3
         ADD R5,R6,#2
         STR R7,R5,#2
         AND R0,R0,#0
         ADD R0,R0,#1
         STR R0,R5,#-2
         LDR R0,R5,#4
         STR R0,R5,#-1
LOOP     LDR R0,R5,#5
         BRz DONE
         LDR R0,R5,#5
         AND R0,R0,#1
         BRz TEST
         LDR R0,R5,#-1
         ADD R6,R6,#-1
         STR R0,R6,#0
         LDR R0,R5,#-2
         ADD R6,R6,#-1
         STR R0,R6,#0
         ; call C function mul
         JSR MUL
         LDR R0,R6,#0
         ADD R6,R6,#3
         STR R0,R5,#-2
TEST     LDR R0,R5,#-1
         ADD R6,R6,#-1
         STR R0,R6,#0
```

```
         LDR R0,R5,#-1
         ADD R6,R6,#-1
         STR R0,R6,#0
         ; call C function mul
         JSR MUL
         LDR R0,R6,#0
         ADD R6,R6,#3
         STR R0,R5,#-1
         AND R0,R0,#0
         ADD R0,R0,#1
         ADD R6,R6,#-1
         STR R0,R6,#0
         LDR R0,R5,#5
         ADD R6,R6,#-1
         STR R0,R6,#0
         ; call C function rshift
         JSR RSHIFT
         LDR R0,R6,#0
         ADD R6,R6,#3
         STR R0,R5,#5
         BRnzp LOOP
DONE     LDR R0,R5,#-2
         STR R0,R5,#3
         LDR R7,R5,#2
         LDR R5,R5,#1
         ADD R6,R6,#5
         RET
```

In order to help you better understand **foo**, we provide you with the C functions **mul** and **rshift**:

```
int16_t mul(int16_t M, int16_t N) { return M * N; }
int16_t rshift(int16_t M, int16_t N) { return M >> N; }
```

Write C code below for the function **foo** from which a **non-optimizing compiler** could have produced the LC-3 code above, following the LC-3 calling convention explained in lecture for all subroutines (**foo**, **mul**, and **rshift**). For local variables, choose names from **A**, **B**, and **C**. All data types are **int16_t**. To receive full credit, make sure that your C code would generate no warnings or errors when compiled. (For more space, write on back of page **AND TELL US HERE**!). Prototype is already provided:

```
int16_t foo(int16_t M, int16_t N) {
    int16_t A = 1, B = M, C;
    while (N != 0) {            // "(N)" also works.
        if ((N & 1) != 0) {     // "(N & 1)" also works
            A = mul(A, B);      // "&" has lower precedence than "!="
        }                       // "(N & 1 != 0)" has a warning
        B = mul(B, B);
        N = rshift(N, 1);
    }
    return A;
}
```

Problem 4, continued:

Part B. *(5 points) USING NO MORE THAN TWENTY WORDS**, describe the functionality of **`foo`** in English (writing code or pseudo-code earns no credit). Hint: You may assume both **`M`** and **`N`** are positive.

Compute M^N and return.

Part C. (2 points) Like LC-3, the x86 ISA also has a small number of registers. One x86 register is **`ESP`**, which points to the top of the stack. Which LC-3 register is used by convention in the way that the x86 **`ESP`** register is used? **CIRCLE EXACTLY ONE ANSWER.**

`ESP` in x86 is similar to (**`R0`** / **`R1`** / **`R2`** / **`R3`** / **`R4`** / **`R5`** / **`R6`** / **`R7`**) in the LC-3 register conventions.

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

Instruction	Encoding	Assembly	RTL
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

Table of ASCII Characters

Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	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