ECE 220: Computer Systems & Programming

Spring 2019 – Midterm Exam 2

April 4, 2019

1. This is a closed book exam except for 1 sheet of hand-written notes
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

Select the Lecture Section You Will Attend to Pick Up Your Booklet

[ ] Prof. Moon (BL4, 9:30 am)
[ ] Prof. Hu (BL2, 11:00 am)
[ ] Prof. Chen (BL, 12:30 pm)
[ ] Prof. Bhowmik (BL3, 3:30 pm)

Name:_____

NetID:_____

Room:_____

Question 1 (20 points): _____

Question 2 (20 points): _____

Question 3 (35 points): A_____; B_____; C_____

Question 4 (25 points): A_____; B_____; C_____; D_____

Total Score: _____

Problem 1 (20 points): Spiral Matrix

In this problem, we want to get the elements of an m x n matrix in **clockwise** spiral order. We will start the spiral from [0, 0].

Ex

$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \rightarrow [1 \quad 2 \quad 3 \quad 6 \quad 5 \quad 4]$

Ex

$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \rightarrow [1 \quad 2 \quad 3 \quad 6 \quad 9 \quad 8 \quad 7 \quad 4 \quad 5]$

Ex

$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{bmatrix} \rightarrow [1 \quad 2 \quad 3 \quad 6 \quad 9 \quad 12 \quad 11 \quad 10 \quad 7 \quad 4 \quad 5 \quad 8]$

One approach to solving this problem is to unravel each outer layer of the matrix until the center is reached.
Below is the algorithm for solving this layer-by-layer:

- For each outer layer, iterate through the elements clockwise starting from the **top-left corner**:
    - Traverse the top row of the layer
    - Traverse the rightmost column of the layer
    - If there are 4 sides to the layer
        - Traverse the bottom row of the layer
        - Traverse the leftmost column of the layer
    - Update the outer layer boundaries

Complete the following function with the following function declaration:

```
int* spiralMatrix(int* matrix, int RowSize, int ColSize);
```

- int* matrix → The 1D representation of the input 2D matrix (as seen on the left in the examples) that will be converted
- int RowSize → Number of rows in the input matrix
- int ColSize → Number of columns in the input matrix
- Returns 1D matrix containing the clockwise spiral

```
int* spiralOrder(int* matrix, int RowSize, int ColSize) {
    /* Allocate space for spiral matrix */
    int* ans = (int*) malloc(sizeof(int) * RowSize * ColSize);

    /* Set bounds for the outermost layer such that
       r_low is the first row */
    int r_low  = 0;
    int r_high = RowSize - 1;
    int c_low  = 0;
    int c_high = ColSize - 1;

    int c, r;    /* Column and row indices */
    int idx = 0; /* Index for spiral matrix */

    // Iterate while there are outer layers
    while (r_low<=r_high && c_low<=c_high){
        /* Traverse right/down along row/column and
           add element to spiral matrix */

        for (c = c_low;_____________________; c++, idx++){

            ans[idx] = _____________________________________________;
        }
        for (r = r_low + 1;_____________________; r++, idx++){

            ans[idx] = _____________________________________________;
         }

        /* Determine if it is possible to move left or up in
           this layer */
        if (r_low<=r_high && c_low<=c_high){
            /* Traverse left/up along row/column and
               add element to spiral matrix */

            for (c = c_high - 1;_____________________; c--, idx++){

                ans[idx] = _____________________________________________;
            }
            for (r = r_high;_____________________; r--, idx++){

                ans[idx] = _____________________________________________;
            }
        }

        /* Update bounds for iteration (row_low, row_high,
                                        col_low, col_high) */
        r_low++;
        r_high--;
        c_low++;
        c_high--;
    }
    return ans;
}
```

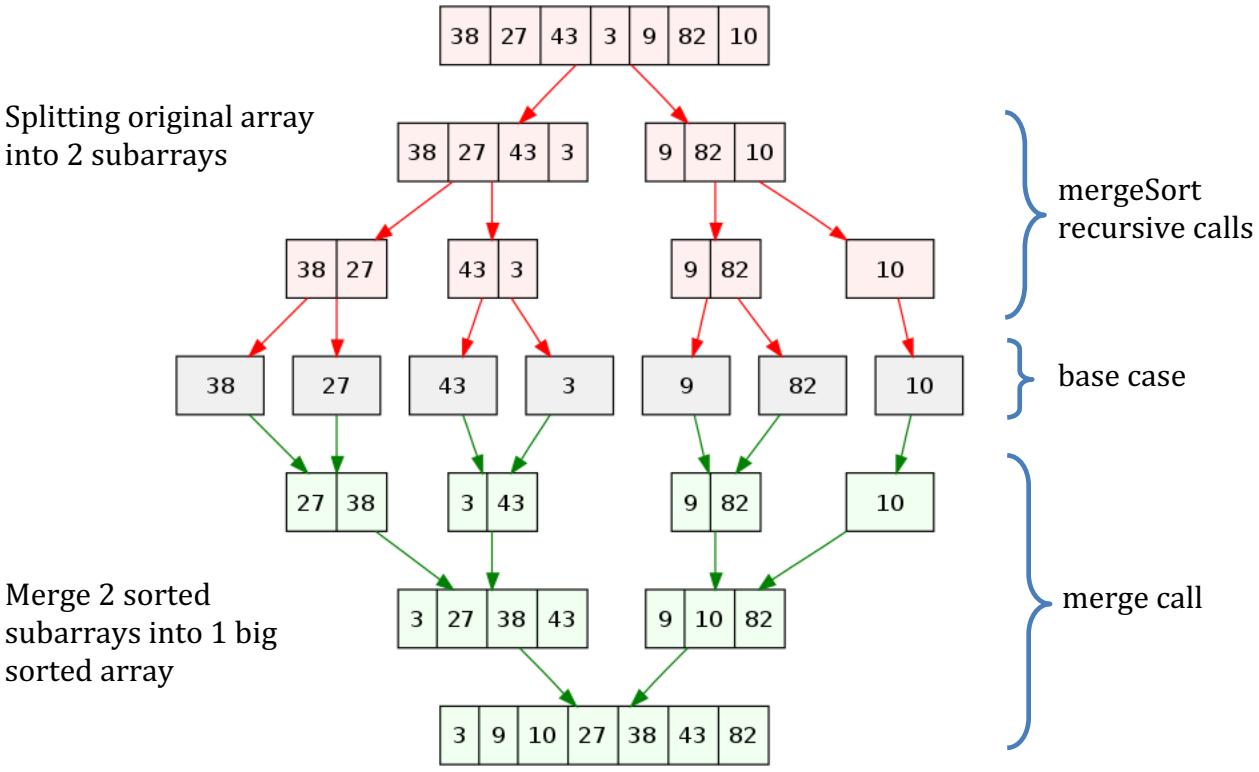
Problem 2 (20 points): Recursion – Merge Sort

In this problem, you need to implement an algorithm called '**Merge Sort**' to sort the integer elements stored in an array in ascending order.

Input : 38, 27, 43, 3, 9, 82, 10

Output : 3, 9, 10, 27, 38, 43, 82

A schematic diagram of how merge sort works can be found below.



We will be using two functions – **merge** and **mergeSort** to implement this algorithm which are declared as following.

```
void merge(int arr[], int l, int m, int r);
void mergeSort(int arr[], int l, int r);
```

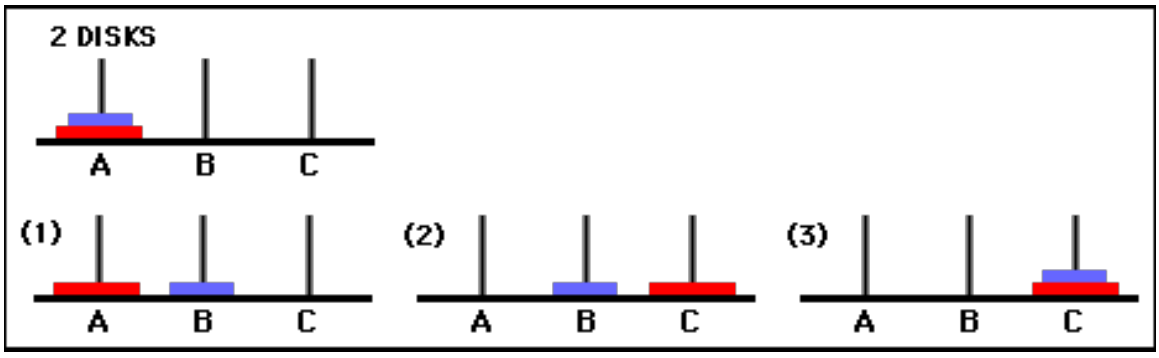
Now in this question, you need to implement the mergeSort function. Assume that the merge function has been implemented for you. **Hint: you need to call merge in mergeSort. Think about what should be the base case and the recursive case(s) in mergeSort.**

```
void merge(int arr[], int l, int m, int r){
     /* Implementation omitted for simplicity */
     /* It will merge two subarrays of arr[].
        First subarray is arr[l..m]
        Second subarray is arr[m+1..r] */
}

/* Use no more than 6 semicolons (;) in mergeSort, or you will
receive 0 for this question */
void mergeSort(int arr[], int l, int r){
    /* l is the lowest index and r is the highest index
       of the arr to be sorted */
    /* base case */

      _____

      _____

      _____

    /* recursive case(s) */

      _____

      _____

      _____

      _____

      _____

      _____

      _____

}

int main(){
     int arr[] = {38, 27, 43, 3, 9, 82, 10};
     int arr_size = 7;
     printf("Printing unsorted array\n");
     printArray(arr, arr_size); /* assume printArray given */
     mergeSort(arr, 0, arr_size - 1);
     printf("Printing sorted array\n");
     printArray(arr, arr_size); /* assume printArray given */
     return 0;
}
```

Problem 3 (35 points): C to LC-3 Conversion

Tower of Hanoi is a mathematical puzzle where we have three rods and n disks. The objective of the puzzle is to move the entire stack to another rod, obeying the following simple rules:
1) Only one disk can be moved at a time.
2) Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack.
3) No disk may be placed on top of a smaller disk.

```
Take an example for 2 disks: Let from_rod = 'A', mid_rod = 'B',
to_rod = 'C'.
Step 1 : Shift first disk from 'A' to 'B'.
Step 2 : Shift second disk from 'A' to 'C'.
Step 3 : Shift first disk from 'B' to 'C'.
The pattern here is :
Shift 'n-1' disks from 'A' to 'B'.
Shift last disk from 'A' to 'C'.
Shift 'n-1' disks from 'B' to 'C'.
```




```
// C recursive function to solve tower of hanoi puzzle
void towerOfHanoi(int n, char from_rod, char to_rod, char mid_rod) {
    if (n == 1) {
        printf("\n Move disk 1 from rod %c to rod %c", from_rod,
                 to_rod);
        return;
    }
    towerOfHanoi(n-1, from_rod, mid_rod, to_rod);

    printf("\n Move disk %d from rod %c to rod %c", n, from_rod,
           to_rod);

    towerOfHanoi(n-1, mid_rod, to_rod, from_rod);
}


int main() {
    int n = 3; // Number of disks
    towerOfHanoi(n, 'A', 'C', 'B');  // A, B and C are names of rods
    return 0;
}
```

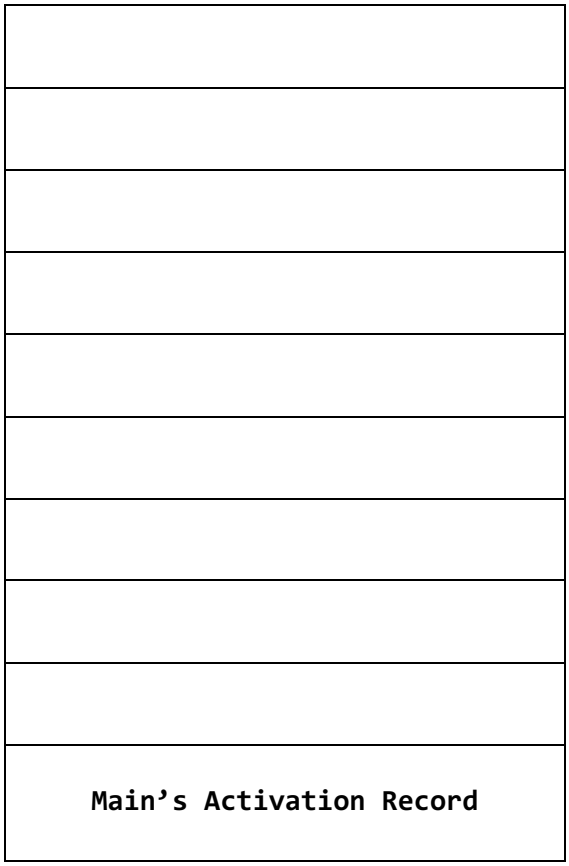
Part A (3 points):
For n = 3, what is the first three output? Please fill in the blanks:

Move disk _____ from rod _____ to rod _____

Move disk _____ from rod _____ to rod _____

Move disk _____ from rod _____ to rod _____

Part B (5 points)
You **MUST** use and conform to the **LC-3 calling conventions** we have described **in class**. Let R6 be the stack pointer and R5 the frame pointer. Please show the status of stack when the first line of code (e.g. if(n==1)) inside towerofHanoi is executed. Use arrows to mark where R5 and R6 are pointing to.



Part C (17 points): C to LC-3 Conversion

You **MUST** use and conform to the **LC-3 calling conventions** we have described **in class**. You do not have to convert the section denoted as "Omitted for simplicity". To receive full credit, you **MUST** complete each task within the number of lines provided in each section. A section is defined as a set of blank lines without any intervening provided code. A line is defined as a single instruction in LC-3.
If you use a register that was set in a previous task without re-loading that register from the stack, you **MUST** document (with a comment) your usage of that register on first use within each section. Semicolons setting off space for comments have been provided on the given lines.

Please finish the LC-3 subroutine for the towerOfHanoi. **You must use R2 to represent n, R3 for from_rod, R4 for mid_rod, R5 for to_rod.**

```
towerOfHanoi
    ; Complete callee stack build up within 7 lines of code
    ; You MUST comment on each line of your code

    _____________________________________;

    _____________________________________;

    _____________________________________;

    _____________________________________;

    _____________________________________;

    _____________________________________;

    _____________________________________;

    ; preparation for function logic

    _____________________________________; get the number of disks

    _____________________________________; get the from_rod

    _____________________________________; get the to_rod

    _____________________________________; get the mid_rod

    ; Test for the termination condition (n == 1)
    AND R1, R1, #0
    ADD R1, R1, #-1
    ADD R1, R1, R2
    BRz DisplayAndReturn
    ADD R2, R2, #-1

    _____________________________________; Update R2 on the stack;
```

```
    ; caller built-up for first recursive call
    ; omitted for simplicity
    ; H(n-1, source, dest, helper)
    JSR towerOfHanoi

    ; Pop arguments off the stack

    _____________________________________;

    JSR Display
    ; caller built-up for second recursive call
    ; omitted for simplicity
    JSR towerOfHanoi
    BRnzp RETURN

DisplayAndReturn
     ; Omitted for simplicity
DisplayAndReturn
     ; Omitted for simplicity

RETURN
     ; Complete callee stack tear-down within 6 lines of code
     ; You MUST comment on each line of your code

    _____________________________________;

    _____________________________________;

    _____________________________________;

    _____________________________________;

    _____________________________________;

    _____________________________________;

    RET
```

Problem 4 (25 points): Concepts

Part A (10 points):

```
#include<stdio.h>

int sub1(int arg);
int t = 1;

int main (){
        int z = 3;
        z = sub1(z);
        printf("variable z = %d\n", z);
        {
                int t = 4;
                z = t^2;
                printf("variable z = %d\n", z);
        }
        return 0;
}

int sub1(int fluff){
        int i = t;
        return (fluff + i);
}
```

What is the output of the program?

First printf statement: _____

Second printf statement: _____

Part B (5 points):

```
#include<stdio.h>
typedef struct recordStruct{
  char name[100];
  char phone_number[10];
}record;

int main(){
        record new_record={“Jane Doe”, “123456789”};
        record *ptr = &new_record;
        ptr++;
        printf(“%c”, &ptr);
}
```

What is the output of the program?

Your answer: _____

Part C: True or False (5 points)

In C, if an array is passed to another function, the callee function will have a local copy of all the elements of the array.

Your answer: _____

Part D (5 points):

We have used the ‘->’ operator in MP - 2048. What does it do?

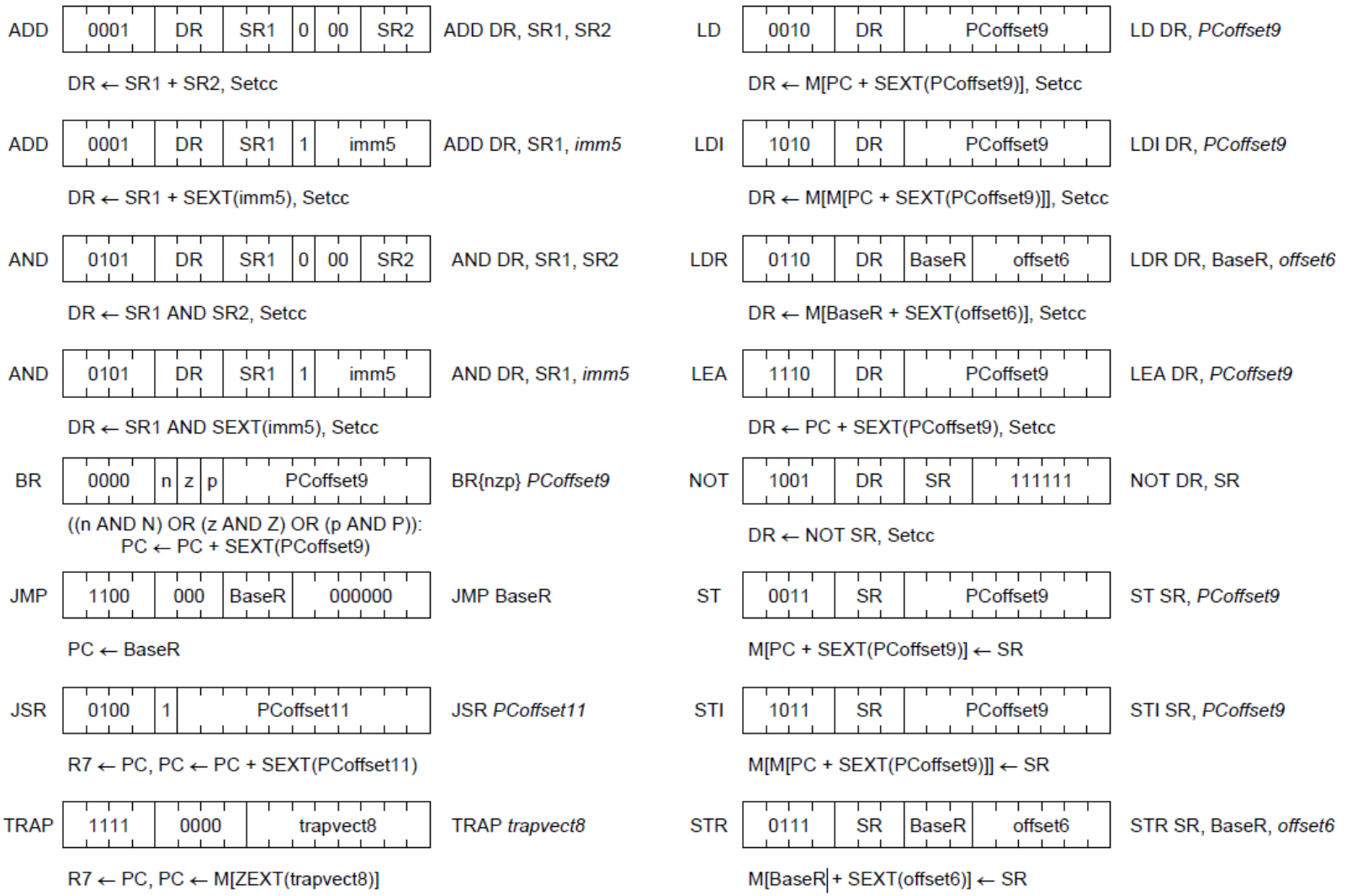
A. Access the member of a structure directly through a variable
B. Have the same functionality as the ‘*‘ operator
C. Dereference a structure pointer and access the member
D. Add more members to structure

Your answer: _____ (select all that apply)

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



End of ECE 220 Midterm Exam 2