Name: _____

- Instructions:
  - Show your work to receive partial credit.
  - Keep your eyes on your own paper and do your best to prevent anyone else from seeing your work.
  - Do NOT communicate with anyone other than the professor/proctor for ANY reason in ANY language in ANY manner.
  - This exam is closed notes, closed books, no calculator.
  - Turn all mobile devices off and put them away now. You cannot have them on your desk.
  - Write neatly and clearly indicate your answers. What I cannot read, I will assume to be incorrect.
  - Stop writing when told to do so at the end of the exam. I will take 5 points off your exam if I have to tell you multiple times.
  - Academic misconduct will not be tolerated. Suspected academic misconduct will be immediately referred to the Rollins Honor Council. Penalties for misconduct will be a zero on this exam, an F grade in the course, and/or other disciplinary action that may be applied by the Rollins Honor Council.
- Time: This exam has 7 questions on 13 pages including the title page. Please check to make sure all pages are included. You will have 120 minutes to complete this exam.

On my honor, I have not given, nor received, nor witnessed any unauthorized assistance on this work. Also, I have read and understand the above policies for this exam.

Signature: _____

Question:	1	2	3	4	5	6	7	Total
Points:	22	8	18	7	25	9	11	100
Score:								

1. Base Conversions: Convert the following numbers.
   (a) (2 points) 52_{10} to 8 bit binary (base 2)

   Solution: 00110100

   (b) (2 points) -52_{10} to 8 bit sign-magnitude binary.

   Solution: 10110100
   Use positive value in part (a), and change sign bit to 1 to represent negative number.

   (c) (2 points) -52_{10} to 8 bit 2's complement binary.

   Solution: 11001011 + 1 = 11001100

   Use positive value in part a, flip all bits, add 1.

   (d) (2 points) 011010110100_2 to octal (base 8)

   Solution:
   bin:011 010 110 100
   oct: 3 2 6 4
   Common error: grouping by 4 bits instead of 3 bits. You group by 4 to transform to hex, group by 3 for octal.

   (e) (3 points) $E7D_{16}$ to binary (base 2)

   Solution:
   E 7 D
   1100 0111 1101

   (f) (3 points) Encode "Zu@" as a C-style string. Give your answer as either hex or binary.

   Solution: binary: 0101 1010 0111 0101 0100 0000 0000 0000
   hex: 5A754000

   (g) (4 points) -31.75_{10} to IEEE single precision (32 bit) floating point decimal number.

   Solution: negative so sign bit is 1
   1 10000011 11111100000000000000000

(h) (4 points) 108_{10} to base 4.

Solution: 1230_4

```
108/4 => 27 R0
27/4 => 6 R3
6/4 => 1 R2
1/4 => 0 R1
0 => stop
```

Common error: taking the remainder on the next to last division and carrying that into the last division to calculate 2/4 and ending up with the final answer 2230 instead of 1230.

2. Calculate the following values if `int x = 11;` and `int y = 12;`. Give you answers as 8-bit, 2's complement values.

(a) (2 points) `x | y`

Solution: `x = 00001011`
`y = 00001100`
`answer = 00001111`

(b) (2 points) `x & y`

Solution: `answer = 00000000`

(c) (2 points) `~x`

Solution: `11110100`
All 8 bits are important. An answer of `0100` is only half credit.

(d) (2 points) `x >> 3`

Solution: x is shifted by 3 bits to the right: `00000001`

3. Explain things to me.

(a) (3 points) What value is stored in the PC, and what role does it play in your program's execution?

Solution: During the Fetch-Decode-Execute cycle, the PC (program counter) stores in the address of the next instruction to execute. This means that it is particularly important to track/store when calling functions. We must store the value of the PC through the function call so that when the function ends, we have the address to return to and continue with execution.

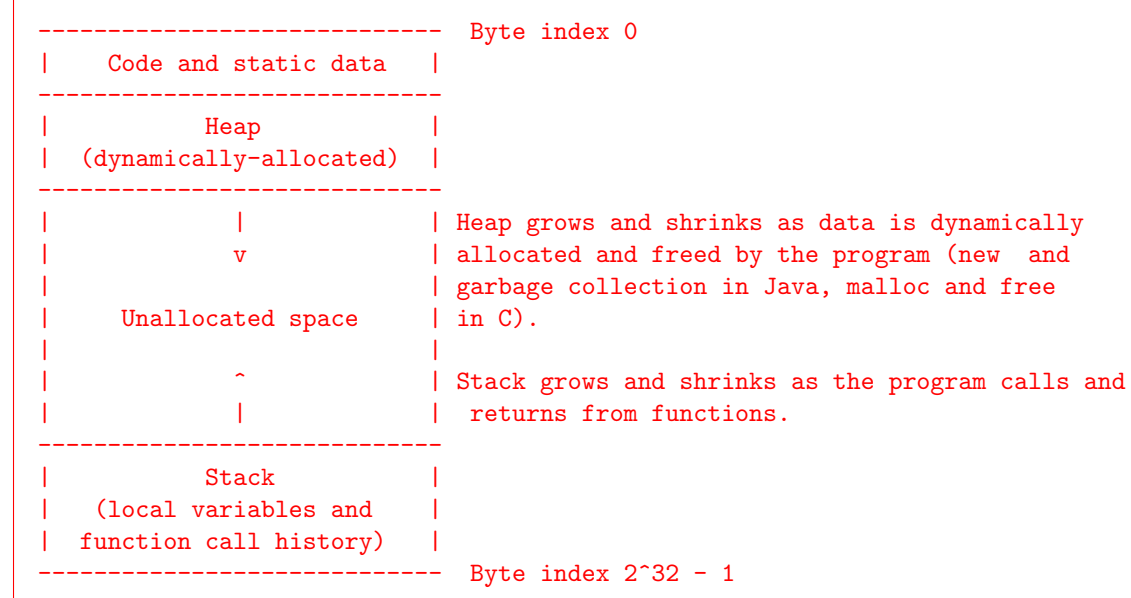
(b) (3 points) What is an ISA, and what role did it play in this course?

Solution: An ISA is an Instruction Set Architecture. An ISA can be thought of as an interface or an API for a particular chip. It defines what operations we can ask a chip to do/calculate, and how those operations work (e.g. what effects the instruction has on the HW). In this course, we used an ARM chip, and thus had to adhere to the ARM ISA when we wrote our assembly programs.

(c) (3 points) Recall the canonical model of a program's address space, which is conceptually a big array of bytes divided into three regions: code and static data, heap, and stack. Draw a picture of the canonical address space. Label each of the three regions.

Solution:

A canonical 32-bit address space (taken directly from the course notes)



Common error: Swapping the "unallocated space" and "static data and code" sections of memory.

(d) Summarize the ARM calling conventions:

i. (2 points) How are arguments passed?

Solution: The first 4 arguments are passed in r0-r3. Others are passed on the stack. Arguments must be passed IN ORDER!

ii. (1 point) Which registers can be freely modified by the callee?

Solution: r0-r3

iii. (1 point) Which registers is the callee responsible for preserving?

Solution: The callee must save r4-r12 before overwriting them.

iv. (1 point) How are values returned?

Solution: The returned value is stored in r0

(e) (4 points) Give an overview of under what circumstances a linked hash table (as discussed in class) would perform well (i.e. fast) and under what circumstances it would perform poorly. Be sure to include the importance and characteristics of a good hashing function in your discussion.

Solution: 1. The table needs adequate number of buckets to ensure good performance. If there are too few buckets, the buckets will become very full, and the LL chain of nodes will grow to be unreasonably long. Which leads to point No. 2.

2. Each chain must be kept to a reasonable (short) length. If chains are too long, the performance of the HT will degrade to that of a linked list for searching, removing, (and possibly inserting if you are inserting in sorted order or at the end of the LL).

3. A good hashing function will ensure several things which will ensure we don't encounter the two problems above. It will have a low collision probability (things are hashed to different buckets). It has a more or less random/even distribution of output values, and it will avoid clustering (close inputs will have very different hash values).

4. Professor Summet is trying to write a method which will increment a variable's value by a specified value. Unfortunately, her program isn't doing what she wants:

```
//** Prof. Summet's version **//
void incrementVariable1(int x, int y) {
  x = x + y;
}

int main() {
  int var = 5;
  incrementVariable1(var, 2);
  printf("result of incrementVariable1 is %d\n", var);

  return 0;
}
```

(a) (1 point) What value for `var` will the program display when run as currently written? 5

(b) (4 points) Time for you to take over. Complete the version below so that it will function as described.

```
void incrementVariable2(                                ) {
```

Solution:

```
  void incrementVariable2(int* x, int y) {
    *x = *x + y;
  }
```

```
}
```

(c) (2 points) Add a call in `main` (after Prof. Summet's call to `incrementVariable1`) which would correctly use your function to increase `var`'s value by 4.

Solution: `incrementVariable2(&var, 4);`

5. Translate the following code to ARM assembly language.

   (a) (8 points) You do not need to include varible definitions or program "header" materials, and you can assume all the variables are global variables. Focus just on the code.

```
if (x < a) {
        x = 1;
} else if (x > b) {
        x = 2;
} else {
        x = a;
}
```

Solution:

```
   ldr r0, =a
   ldr r0, [r0]

   ldr r1, =b
   ldr r1, [r1]

   ldr r2, =x
   ldr r2, [r2]

if:
   cmp r2, r0
   bge elseif
   mov r2, #1
   b done
elseif:
   cmp r2, r1
   ble else
   mov r2, #2
   b done
else:
   mov r2, r0
done:
   ldr r0, =x
   str r2, [r0]
```

Scoring:
+2 reads global variables out of memory
+3 if to elseif to else branching correct
+1 updates value of x correctly if branch taken
+2 stores x back to memory

(b) (7 points) You do not need to include varible definitions or program "header" materials, and you can assume all the variables are global variables. Focus just on the code.

```
while (a < x && b >= y) {
        a++;
        b--;
}
```

Solution:

```
ldr r0, =a
ldr r0, [r0]

ldr r1, =b
ldr r1, [r1]

ldr r2, =x
ldr r2, [r2]

ldr r3, =y
ldr r3, [r3]

while:
   cmp r0, r2 //a and x
   bge done
   cmp r1, r3 //b and y
   blt done
   add r0, #1 //a++
   sub r1, #1 //b--
   b while
done:
   //a and b's values are updated by loop!
   //must store these values back to memory!
   ldr r2, =a
   ldr r3, =b
   str r0, [r2]
   str r1, [r3]
```

Scoring:
+2 reads global variables out of memory
+2 while loop structure correct
+2 both conditions checked
+1 stores a/b values back to memory

(c) (10 points) Translate the following entire program into assembly.

```
int var = 1; // Global variable
int sum(int c, int d) {
        return c + d;
}
int main() {
        printf("%d\n", sum(var, 3));
        return;
}
```

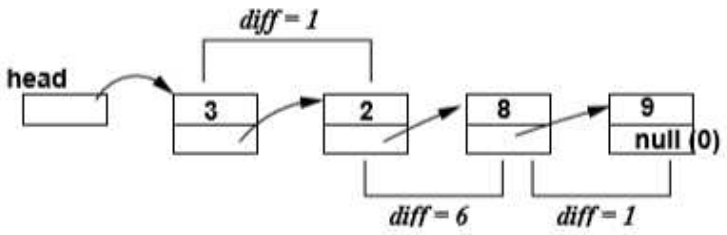
Solution:

```
.global main

.data
var .word 1
format .asciz "%d\n"

.text
sum:
   push{ip, lr}
   add r0, r0, r1
   pop{ip, pc}

main:
   push{ip, lr}
   ldr r0, =var
   ldr r0, [r0]

   mov r1, #3
   bl sum

   mov r1, r0
   ldr r0, =format
   bl printf
   pop{ip, pc}
```

6. (9 points) You are given the following structure which can be used to define a linked list:

```
typedef struct _node_t {
    int value;
    struct _node_t* next;
} Node;
```

We define the *difference* of a pair of consecutive nodes in a linked list to be the absolute difference between their values. For example:



Write a C function which takes a pointer to the start of the list as an argument. The function should return the difference for the linked list as described above. For the above example list, the function would return 8 (1 + 6 + 1). Your function should not modify the linked list in any way. You can assume that the list has at least 1 `Node`.

Solution:

```
int difference(Node* head) {
  if(head == NULL) {
    return;
  }

  //maintain two pointers
  Node* prev = head;
  Node* temp = head->next;

  //running sum
  int diff = 0;

  while(temp != NULL) {
    //difference between two nodes
    int d = temp->value - prev->value;

    //absolute value calculation
    if(d < j0) {
      d = d * -1;
    }
    //add to running sum
    diff += d;

    //advance both pointers
    prev = prev->next;
    temp = temp->next;
  }
}
```

Scoring:
+2 function header
+1 checks if list has no nodes
+2 calculates difference between two nodes
+2 calculates running sum
+4 loop advances and terminates correctly

7. (11 points) Write a function named `printNums` which takes a single int as a parameter. This function should print a pyramid as shown below which is the number of rows high as specified by the parameter.

   Examples:

   `printNums(5)` outputs:

   ```
   ....1
   ...22
   ..333
   .4444
   55555
   ```

   `printNums(3)` outputs:

   ```
   ..1
   .22
   333
   ```

   `printNums(7)` outputs:

   ```
   ......1
   .....22
   ....333
   ...4444
   ..55555
   .666666
   7777777
   ```

Solution: Solutions vary. A sample answer:

```
void printNums(int h) {
  int dots = h - 1; //number of dots to print
  int nums = 1;  //number of nums to print

  for(i=0; i < h; i++) {   //each iteration prints a row
    for(j = 0; j < dots; j++) {  //print the dots
      printf(".");
    }

    for(j = 0; j < nums; j++) { //print the numbers
      printf("%d", nums);   // or (i+1)
    }

    printf("\n");
    dots--;
    nums++;
  }
}
```

Scoring:
+2: function header (all parts)
+2: outer loop
+2: inner loop(s) run correct number of times
+2: prints correct number of .
+2: prints correct number of numbers and correct number
+1: ends lines at correct place (newline chars)

Reference Material

Excess-127 Encoding

Bit Pattern	Value Encoded
00000000	-127
00000001	-126
...	...
01111111	0
10000000	1
10000001	2
...	...
11111111	128

Fractions and decimal equivalents

Fraction	Decimal Value
$\frac{1}{2}$	.5
$\frac{1}{4}$	.25
$\frac{1}{8}$	.125
$\frac{1}{16}$	.0625
$\frac{1}{32}$	.03125

`printf` format strings:

Syntax	Datatype
%i, %d	integer
%f	double, float
%c	char
%s	string
%x, %X	hex rep.
%p	pointer

ASCII chart

Dec	Hex	Char
000	00	(nul)
001	01	(soh)
002	02	(stx)
003	03	(etx)
004	04	(eot)
005	05	(enq)
006	06	(ack)
007	07	(bel)
008	08	(bs)
009	09	(tab)
010	0A	(lf)
011	0B	(vt)
012	0C	(np)
013	0D	(cr)
014	0E	(so)
015	0F	(si)
016	10	(dle)
017	11	(dc1)
018	12	(dc2)
019	13	(dc3)
020	14	(dc4)
021	15	(nak)
022	16	(syn)
023	17	(etb)
024	18	(can)
025	19	(em)
026	1A	(eof)
027	1B	(esc)
028	1C	(fs)
029	1D	(gs)
030	1E	(rs)
031	1F	(us)

Dec	Hex	Char
032	20	␣
033	21	!
034	22	"
035	23	#
036	24	$
037	25	%
038	26	&
039	27	'
040	28	(
041	29	)
042	2A	*
043	2B	+
044	2C	’
045	2D	-
046	2E	.
047	2F	/
048	30	0
049	31	1
050	32	2
051	33	3
052	34	4
053	35	5
054	36	6
055	37	7
056	38	8
057	39	9
058	3A	:
059	3B	;
060	3C	<
061	3D	=
062	3E	>
063	3F	?

Dec	Hex	Char
064	40	@
065	41	A
066	42	B
067	43	C
068	44	D
069	45	E
070	46	F
071	47	G
072	48	H
073	49	I
074	4A	J
075	4B	K
076	4C	L
077	4D	M
078	4E	N
079	4F	O
080	50	P
081	51	Q
082	52	R
083	53	S
084	54	T
085	55	U
086	56	V
087	57	W
088	58	X
089	59	Y
090	5A	Z
091	5B	[
092	5C	\
093	5D	]
094	5E	^
095	5F	_

Dec	Hex	Char
096	60	‘
097	61	a
098	62	b
099	63	c
100	64	d
101	65	e
102	66	f
103	67	g
104	68	h
105	69	i
106	6A	j
107	6B	k
108	6C	l
109	6D	m
110	6E	n
111	6F	o
112	70	p
113	71	q
114	72	r
115	73	s
116	74	t
117	75	u
118	76	v
119	77	w
120	78	x
121	79	y
122	7A	z
123	7B	{
124	7C	|
125	7D	}
126	7E	~
127	7F	DEL