Question 6 (30 points):

Dec	Char	Dec	Char	Dec	Char	Dec	Char
0	NUL (null)	32	SPACE	64	@	96	`
1	SOH (start of heading)	33	!	65	A	97	a
2	STX (start of text)	34	"	66	B	98	b
3	ETX (end of text)	35	#	67	C	99	c
4	EOT (end of transmission)	36	$	68	D	100	d
5	ENQ (enquiry)	37	%	69	E	101	e
6	ACK (acknowledge)	38	&	70	F	102	f
7	BEL (bell)	39	'	71	G	103	g
8	BS (backspace)	40	(	72	H	104	h
9	TAB (horizontal tab)	41	)	73	I	105	i
10	LF (NL line feed, new line)	42	*	74	J	106	j
11	VT (vertical tab)	43	+	75	K	107	k
12	FF (NP form feed, new page)	44	,	76	L	108	l
13	CR (carriage return)	45	-	77	M	109	m
14	SO (shift out)	46	.	78	N	110	n
15	SI (shift in)	47	/	79	O	111	o
16	DLE (data link escape)	48	0	80	P	112	p
17	DC1 (device control 1)	49	1	81	Q	113	q
18	DC2 (device control 2)	50	2	82	R	114	r
19	DC3 (device control 3)	51	3	83	S	115	s
20	DC4 (device control 4)	52	4	84	T	116	t
21	NAK (negative acknowledge)	53	5	85	U	117	u
22	SYN (synchronous idle)	54	6	86	V	118	v
23	ETB (end of trans. block)	55	7	87	W	119	w
24	CAN (cancel)	56	8	88	X	120	x
25	EM (end of medium)	57	9	89	Y	121	y
26	SUB (substitute)	58	:	90	Z	122	z
27	ESC (escape)	59	;	91	[	123	{
28	FS (file separator)	60	<	92	\	124	|
29	GS (group separator)	61	=	93	]	125	}
30	RS (record separator)	62	>	94	^	126	~
31	US (unit separator)	63	?	95	_	127	DEL

Figure 1: ASCII Table

In this question you will create two functions to print a string that may include a variable number of integers. The `PrintString` function receives three parameters: the address of a null-terminated string `S`; the address of the first position of a vector of integer values `V`; and the address of an output buffer `B`. Whenever the sequence of characters `%d` appears in the string, these characters must be replaced by a substring that represents the value of one of the integers in the vector `V`. Here are some examples (`S` is the input string, `V` is the vector of integer values, `B` is the output string:

```
S = Sift %d pounds and %d ounces of flour.
V = {2, 4}
B = Sift 2 pounds and 4 ounces of flour.
```

```
S = She got almost %d million more votes than him.  She got %d (%d%) and he got
%d (%d%).  Still, he was elected.
V = {3, 65844954, 48, 62979879, 46}
```

```
B = She got almost 3 million more votes than him.  She got 65844954 (48%) and he
got 62979879 (46%).  Still, he was elected.
```

The relevant portion of the ASCII table is shown in Figure 1.

The solution must work for any null-terminated strings, including the empty string.

All functions must follow all the RISC-V register saving/restoring conventions.

1. (**10 points**) The first function that you will create is called `intToString`. It has two parameters: an integer value and the memory address to the byte in memory that will contain the first character of the string representation of the integer value. `intToString` will create a null-terminated string starting at that address and will return the address of the null character at the end of the created string.

   parameters:

   - `a0`: integer value
   - `a1`: memory address where string should start

   return value:

   - `a0`: memory address of the `NULL` byte at the end of the created string

2. (**20 points**) Now you will write `PrintString`, which has three parameters. The address of the first character of a null-terminated string `S` that may contain `%d` sequences in it. The address to the first position of a vector of integer values `V`. And the address to the first position of a buffer `B` that will contain the output string.

   parameters:

   - `a0`: address of null-terminated string `S`
   - `a1`: address of vector of integers `V`
   - `a2`: output string buffer `B`

   return value: None

```
# intToString
#
# Parameters:
#   a0: integer value
#   a1: pointer to a buffer
# Return Value:
#   a0: address of the null character in the buffer
#
# Register Usage:
# t0: tester
# t1: constant 10
intToString:
    addi    t0, zero, 1     # tester <- 1
    addi    t1, zero, 10    # ten <- 10
bigger:
    muli    t0, t0, t1     # tester <- tester*10
    bgt     a0, t0 bigger  # if a0 > tester go to bigg
smaller:
    div     t0, t0, t1      # tester <- tester/10
    div     t2, a0, t0      # t2 <- a0/tester
    addi    t3, t0, 0x30    # t3 <- ASCII for digit
    sb      t3, 0(a1)       # print digit
    mul     t4, t2, t0      # t4 <- (a0/tester)*tester
    sub     a0, a0, t4      # a0 <- a0 - (a0/tester)*t
    addi    a1, a1, 1       # a1 <- a1+1
    bgtz    a0, smaller     # if a0 > 0 goto smaller
    sb      zero, 0(a1)     # print '\0'
    mv      a0, a1          # a0 <- a1
    jalr    zero, ra, 0     # return
```

Figure 2: A solution for `intToString`.

```
#
# PrintString
# Parameters:
#   a0 <- address of null-terminated string S
#   a1 <- address of vector of integers V
#   a2 <- output string buffer B
#
PrintString:
    addi sp, sp, -16
    sw  s0, 0(sp)
    sw  s1, 4(sp)
    sw  s2, 8(sp)
    sw  ra, 12(sp)
    mv  s0, a0          # s0 <- a0 = pointer to string S
    mv  s1, a1          # s1 <- a1 = pointer to V
    mv  s2, a2          # s2 <- a2 = pointer to output buffer B
nextchar:
    lb  t0, 0(s0)       # t0 <- character
    beq t0, zero, done  # if character == '\0' goto done
    li  t1, 0x25        # t1 <- '%'
    bne t0, t1, notPerc # if character != "%" got notInt
    lb  t2, 1(s0)       # t2 <- nextCharacter
    li  t3, 0x64        # t3 <- 'd'
    bne t2, t3, notd    # if T2 != 'd' goto notd
    lw  a0, 0(s1)       # a0 <-- *V
    mv  a1, s2          # a1 <-- current output buffer pointer
    jal intToString
    mv  s2, a0          # B <- new output buffer pointer
    addi s1, s1, 4      # V++
    addi s0, s0, 2      # S <- S+2 (skipping over "%d")
    j nextchar
notPerc:
    sb  t0, 0(s2)       # *B <- character
    addi s0, s0, 1      # S++
    addi s2, s2, 1      # B++
    j nextchar
notd:
    sb t0, 0(s2)        # *B <- '%'
    sb t2, 1(s2)        # *B <- character after '%'
    addi s2, s2, 2      # B <- B+2
    addi s0, s0, 2      # S <- S+2
    j nextchar
done:
    sb  t0, 0(s2)       # *B <- '/0'
    lw  s0, 0(sp)
    lw  s1, 4(sp)
    lw  s2, 8(sp)
    lw  ra, 12(sp)
    addi sp, sp, 16
    jalr    zero, ra, 0     # return
```

Figure 3: A solution to the `PrintString` function.

```
# PrintString  -- a simpler solution, after students
# Parameters:
#   a0 <- address of null-terminated string S
#   a1 <- address of vector of integers V
#   a2 <- output string buffer B
#
PrintString:
    addi sp, sp, -24
    sw  s0, 0(sp)
    sw  s1, 4(sp)
    sw  s2, 8(sp)
    sw  s3, 12(sp)
    sw  s4, 16(sp)
    sw  ra, 20(sp)
    mv  s0, a0          # s0 <- a0 = pointer to string S
    mv  s1, a1          # s1 <- a1 = pointer to V
    mv  s2, a2          # s2 <- a2 = pointer to output buffer B
    li  s3, 0x25        # t1 <- '%'
    li  s4, 0x64        # t3 <- 'd'
nextchar:
    lb  t0, 0(s0)       # t0 <- character
    beq t0, s3, perc    # if character == "%" goto perc
putchar:
    sb  t0, 0(s2)       # *B <- character
    addi s0, s0, 1      # S++
    addi s2, s2, 1      # B++
    bne t0, zero, nextchar # if character != '\0' goto nextchar
    lw  s0, 0(sp)
    lw  s1, 4(sp)
    lw  s2, 8(sp)
    lw  s3, 12(sp)
    lw  s4, 16(sp)
    lw  ra, 20(sp)
    addi sp, sp, 24
    jalr    zero, ra, 0     # return
perc:
    lb  t2, 1(s0)       # t2 <- nextCharacter
    bne t2, s4, putchar # if t2 != 'd' goto putchar
    lw  a0, 0(s1)       # a0 <-- *V
    mv  a1, s2          # a1 <-- current output buffer pointer
    jal intToString
    mv  s2, a0          # B <- new output buffer pointer
    addi s1, s1, 4      # V++
    addi s0, s0, 2      # S <- S+2 (skipping over "%d")
    j nextchar
```

Figure 4: A solution to the `PrintString` function with simpler control flow — after many student's solutions.