

Lab 11: Functions in RISC-V Assembly SOLUTIONS

Digital Design and Computer Architecture: RISC-V Edition (Harris & Harris, Elsevier © 2021)

1. Please indicate how many hours you spent on this lab. This will be helpful for calibrating the workload for next time the course is taught.
2. Your code: div9Function.s, bubblesortFunction.s, gcd.s, greet.c, and greet.s.

div9Function.s

```
# test cases
test:    addi a0, zero, 15      # div9(15)
        jal  div9
        addi a0, zero, 81      # div9(81)
        jal  div9
        nop
div9:    beq  a0, zero, isdiv9
        blt  a0, zero, notdiv9
        addi a0, a0, -9         # a0 = a0 - 9
        j    div9
notdiv9: addi a0, zero, 0        # a0 = 0 (not divisible by 9)
        j    done
isdiv9:  addi a0, zero, 1        # a0 = 1 (is divisible by 9)
done:    jr   ra                # return
```

bubblesortFunction.s

```
# Test case: sortarray = {-15,42,73,19,-8,24,16,-2,99,-78,-21,23,-88,49,-101}
# When the code completes, sortarray is:
# {-101,-88,-78,-21,-15,-8,-2,16,19,23,24,42,49,73,99}
```

```
initarray:
    addi a0, zero, 0x400
    addi t0, zero, -15
    sw    t0, 0(a0)
    addi t0, zero, 42
    sw    t0, 4(a0)
    addi t0, zero, 73
    sw    t0, 8(a0)
    addi t0, zero, 19
    sw    t0, 0xC(a0)
    addi t0, zero, -8
    sw    t0, 0x10(a0)
    addi t0, zero, 24
    sw    t0, 0x14(a0)
    addi t0, zero, 16
    sw    t0, 0x18(a0)
    addi t0, zero, -2
    sw    t0, 0x1C(a0)
    addi t0, zero, 99
    sw    t0, 0x20(a0)
    addi t0, zero, -78
    sw    t0, 0x24(a0)
    addi t0, zero, -21
```

```

sw    t0, 0x28(a0)
addi  t0, zero, 23
sw    t0, 0x2C(a0)
addi  t0, zero, -88
sw    t0, 0x30(a0)
addi  t0, zero, 49
sw    t0, 0x34(a0)
addi  t0, zero, -101
sw    t0, 0x38(a0)
addi  a1, zero, 15
jal   bubblesort
nop

```

```

# s0 = tmp, s1 = swapped
bubblesort:
    addi a1, a1, -1          # a1 = size-1W
    slli t0, a1, 2          # t0 = (size-1) * 4
    add  t0, a0, t0          # just past second to last element of array
    addi s1, zero, 1        # swapped = 1
L1:
    beq  s1, zero, done     # if (swapped == 0), done
    addi s1, zero, 0        # swapped = 0
    addi t3, a0, 0          # t3 = base address of sortarray[]
L2:
    bge  t3, t0, L1         # if i >= (size-1), end of for loop
    lw   s0, 0(t3)          # tmp = sortarray[i]
    lw   t1, 4(t3)          # t1 = sortarray[i+1]
    bge  t1, s0, L3         # if sa[i+1] >= sa[i], continue
    sw   t1, 0(t3)          # sortarray[i] = sortarray[i+1]
    sw   s0, 4(t3)          # sortarray[i+1] = tmp
    addi s1, zero, 1        # swapped = 1
L3:
    addi t3, t3, 4          # t3 = address of next element in array
    j    L2
done:
    jrr  ra                 # return

```

gcd.s

```

# Algorithm:
# int tmp;
# if (n1 > n2) {
#     tmp = n2;
#     n2 = n1;
#     n1 = tmp;
# }
# while (n2 != 0 & n1 != 0) {
#     tmp = n2 % n1;
#     n2 = n1;
#     n1 = tmp;
# }
# if (n2 == 0) return n1;
# else return n2;

```

```

test:
    addi a0, zero, 25      # gcd(25,15)
    addi a1, zero, 15
    jal  gcd
    addi a0, zero, 64      # gcd(64,96)
    addi a1, zero, 96
    jal  gcd
    addi a0, zero, 71      # gcd(71,9)
    addi a1, zero, 9
    jal  gcd
    nop

# a1 should hold larger value
gcd: bge a1, a0, L1        # if a1 < a0, swap a0 and a1
    addi t0, a1, 0         # a1 holds max
    addi a1, a0, 0         # a0 holds min
    addi a0, t0, 0
L1:  beq a1, zero, L2      # if a1 or a0 is 0, done
    beq a0, zero, L2
    rem t0, a1, a0
    addi a1, a0, 0         # a1 = a0 (new max)
    addi a0, t0, 0         # a0 = remainder (new min)
    j    L1
L2:  beq a1, zero, L3      # if a1 != 0, return a1
    addi a0, a1, 0         # return a1
L3:  jr   ra               # return

# greet.s
    addi a0, zero, 7       # greet(7) = 21
    jal  greet
    addi a0, zero, 10      # greet(10) = 45
    jal  greet
    nop

greet:
    addi sp, sp, -8        # make room for two words on stack
    sw   ra, 4(sp)         # save ra on stack
    sw   a0, 0(sp)         # save n on stack
    addi t0, zero, 2       # t0 = 1
    bne  a0, t0, else      # if n != 1, do else
    addi a0, zero, 0       # return 0
    jr   ra

else:
    addi a0, a0, -1        # n = n-1
    jal  greet
    lw   ra, 4(sp)         # restore registers from stack
    lw   t0, 0(sp)         # t0 = n (formerly a0)
    addi sp, sp, 8         # restore sp
    addi t0, t0, -1        # t0 = n-1
    add  a0, a0, t0         # return (n-1) + greet(n-1)
    jr   ra

```

3. Screenshots of a0 from the Venus simulator after making the following function calls:
div9(15);

Run
Step
Prev
Reset
Dump

Machine Code	Basic Code	Original Code
0x00f00513	addi x10 x0 15	test: addi a0, zero, 15
0x010000ef	jal x1 16	jal div9
0x05100513	addi x10 x0 81	addi a0, zero, 81

s0 (x8) 0x00000000
s1 (x9) 0x00000000
a0 (x10) 0x00000000
a1 (x11) 0x00000000
a2 0x00000000

div9(81);

Run
Step
Prev
Reset
Dump

0x05100513	addi x10 x0 81	addi a0, zero, 81
0x008000ef	jal x1 8	jal div9
0x00000013	addi x0 x0 0	nop

s0 (x8) 0x00000000
s1 (x9) 0x00000000
a0 (x10) 0x00000001
a1 (x11) 0x00000000

gcd(25, 15);

0x01900513	addi x10 x0 25	addi a0, zero, 25 # gcd(25,15)
0x00f00593	addi x11 x0 15	addi a1, zero, 15
0x020000ef	jal x1 32	jal gcd
0x04000513	addi x10 x0 64	addi a0, zero, 64 # gcd(64,96)
0x06000593	addi x11 x0 96	addi a1, zero, 96
0x014000ef	jal x1 20	jal gcd
0x04700513	addi x10 x0 71	addi a0, zero, 71 # gcd(71,9)

t2 (x7) 0
s0 (x8) 0
s1 (x9) 0
a0 (x10) 5

gcd(64, 96);

0x04000513	addi x10 x0 64	addi a0, zero, 64 # gcd(64,96)
0x06000593	addi x11 x0 96	addi a1, zero, 96
0x014000ef	jal x1 20	jal gcd
0x04700513	addi x10 x0 71	addi a0, zero, 71 # gcd(71,9)

(x9)
a0 (x10) 32
a1 (x11) 32
a2 0

`gcd(71, 9);`

0x04700513	<code>addi x10 x0 71</code>	<code>addi a0, zero, 71 # gcd(71,9)</code>	a0 (x10)	<input type="text" value="1"/>
0x00900593	<code>addi x11 x0 9</code>	<code>addi a1, zero, 9</code>	a1 (x11)	<input type="text" value="1"/>
0x008000ef	<code>jal x1 8</code>	<code>jal gcd</code>	a2	<input type="text" value="0"/>
0x00000013	<code>addi x0 x0 0</code>	<code>nop</code>		

`greet(7);`

0x00700513	<code>addi x10 x0 7</code>	<code>addi a0, zero, 7 # greet(7) = 21</code>	(x8)	<input type="text" value=""/>
0x010000ef	<code>jal x1 16</code>	<code>jal greet</code>	s1 (x9)	<input type="text" value="0"/>
0x00a00513	<code>addi x10 x0 10</code>	<code>addi a0, zero, 10 # greet(10) = 55</code>	a0 (x10)	<input type="text" value="21"/>

`greet(10);`

0x00a00513	<code>addi x10 x0 10</code>	<code>addi a0, zero, 10 # greet(10) = 55</code>	a0 (x10)	<input type="text" value="45"/>
0x008000ef	<code>jal x1 8</code>	<code>jal greet</code>	a1 (x11)	<input type="text" value="0"/>
0x00000013	<code>addi x0 x0 0</code>	<code>nop</code>	a2	<input type="text" value="0"/>

4. A screenshot of memory addresses 0x400 - 0x438 when `bubblesort()` is called with the following 15-element array, `sortarray`, as an argument, with the array's base address being at 0x400. Be sure to include code that initializes the array before the function call.

```
sortarray[15]={-15,42,73,19,-8,24,16,-2,99,-78,-21,23,-88,49,-101};
```

sw t0, 0x2C(a0)
addi t0, zero, -88
sw t0, 0x30(a0)
addi t0, zero, 49
sw t0, 0x34(a0)
addi t0, zero, -101
sw t0, 0x38(a0)
addi a1, zero, 15
jal bubblesort

Registers	Memory			
Address	+0	+1	+2	+3
0x00000438	99	0	0	0
0x00000434	73	0	0	0
0x00000430	49	0	0	0
0x0000042c	42	0	0	0
0x00000428	24	0	0	0
0x00000424	23	0	0	0
0x00000420	19	0	0	0
0x0000041c	16	0	0	0
0x00000418	-2	-1	-1	-1
0x00000414	-8	-1	-1	-1
0x00000410	-15	-1	-1	-1
0x0000040c	-21	-1	-1	-1
0x00000408	-78	-1	-1	-1
0x00000404	-88	-1	-1	-1
0x00000400	-101	-1	-1	-1

Please indicate any bugs you found in this lab manual, or any suggestions you would have to improve the lab.