CSE 315: Computer Organization

Sheet 5

1. Assume that the following MIPS assembly code segment is placed starting at location $F4E0D2C0_{16}$ in memory and that register $s2 holds the base of A (an array of integers).

```
Loop:   jal     Calc
        addi    $s0, $s0, 1
        slt     $t1, $s0, $s1
        bne     $t1, $zero, Loop
        sw      $t0, 16 ($s2)
        j       Exit
```

For each of the following questions choose the *one* best answer. Copy the table below to your answer sheet and put your answers in it. Do *NOT* copy the statements themselves.

I) What is the address of the procedure Calc in hexadecimal, if the value stored in the target address field of the machine code of the jal instruction above is 2_{16}?

a) 00000002	b) 00000008	c) F0000002	d) F0000008	e) F4E0D2C8

II) What is the hexadecimal value stored in the address field of the machine code of the bne instruction above?

a) -4	b) FFFC	c) -16	d) FFF0	e) D2C0	f) 34B0

III) Who is responsible for calculating the value to be stored in the address field of the machine code of the bne instruction above?

a) operating system	b) compiler	c) assembler	d) programmer

IV) If register $s2 contains number 4_{10} before executing the sw instruction above, what will be the decimal store-to address?

a) 20	b) 8	c) 68

V) What is the decimal value stored in the immediate field of the machine code of the sw instruction above?

a) 16	b) 4	c) 64

VI) If the store-to address of the sw instruction above is 40_{10} and register $t0 contains number $A0B0C0D0_{16}$ before executing the instruction, what will be the hexadecimal byte stored at memory address 40_{10} after executing the instruction?

a) A0	b) B0	c) C0	d) D0

VII) What is the addressing mode used in the j instruction above?

a) Pseudodirect	b) PC-relative	c) Immediate	d) Register	e) Displacement

VIII) Which of the followings could be the hexadecimal address of the label Exit referenced in the operand field of the j instruction above?

a) F4E0D2C0	b) E4E0D6FA	c) E4E0D6FC	d) F4E0D6FA	e) F4E0D6FC

Statement	**I**	**II**	**III**	**IV**	**V**	**VI**	**VII**	**VIII**
Answer	**d**	**b**	**c**	**a**	**a**	**a**	**a**	**e**

2. The C function on the left is assembled into the MIPS code on the right.

```
int compare (int x, int y) {
   if ((y - x) < 0)
      return 1;
   else
      return 0;
}
```

```
compare:  sub  $t0, $a1, $a0
          slt  $v0, $t0, $zero
          jr   $ra
```

a. The above function compare is modified, as shown below, to have the subtraction operation done via a functional call to the subtract function. Assemble the two functions compare2 and subtract into MIPS assembly code.

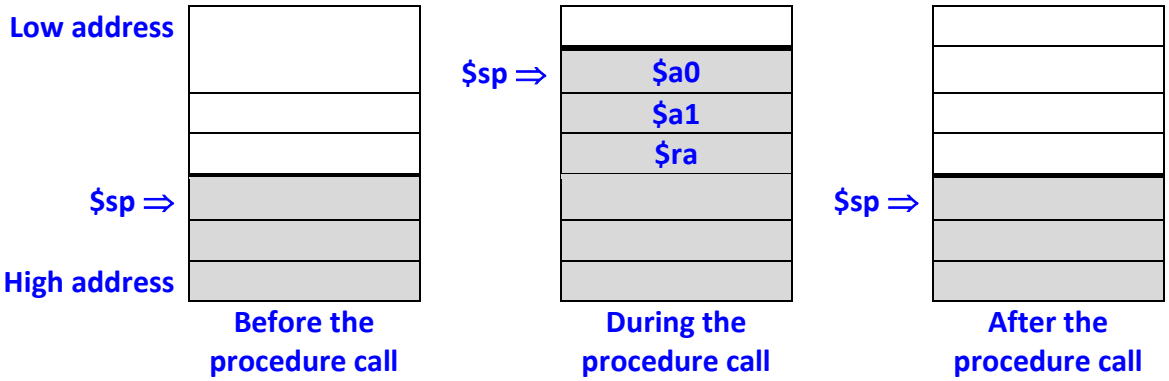
```
int compare2 (int x, int y) {
   if ((subtract(y,x) < 0)
      return 1;
   else
      return 0;
}

int subtract (int a, int b) {
   return a - b;
}
```

Solution:

```
compare2:  addi $sp, $sp, -12
           sw   $ra, 8 ($sp)
           sw   $a1, 4 ($sp)
           sw   $a0, 0 ($sp)

           add  $t0, $a0,
$zero
           add  $a0, $a1,
$zero
           add  $a1, $t0,
$zero

           jal  subtract

           slt  $v0, $v0,
$zero

           lw   $a0, 0 ($sp)
           lw   $a1, 4 ($sp)
           lw   $ra, 8 ($sp)
           addi $sp, $sp, 12

           jr   $ra

subtract:  sub  $v0, $a0, $a1
           jr   $ra
```

b. Trace the stack contents (stack image) before, during, and after calling the function compare2 in Part (a). Indicate the names of the registers stored on the stack and mark the locations on the stack pointed to by the registers $sp.

Solution:



c. Compilers often implement the subtract function in Part (a) using *in-lining*. In-lining a function means copying its body into the caller space, allowing the overhead of the function call to be eliminated. In-lining the subtract function into the compare2 function will produce the original MIPS code of the compare function given above. Find the reduction in the total number of executed MIPS assembly instructions gained from inlining the subtract function.

Solution:

Dynamic size of the original MIPS code = 3.

Dynamic size of the modified MIPS code = 14 + 2 = 16.

Reduction = 16 – 3 = 13.

3. The following assembly code computes the factorial of a number. The integer input is passed through $a0 and the result is returned in register $v0. The code fragment contains some errors. Correct those MIPS errors and show the content of the stack after each function call, assuming that the input is 5.

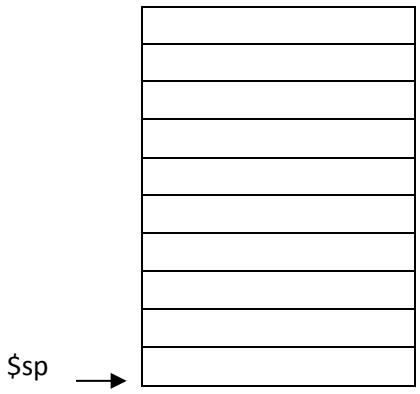
```
FACT:   addi  $sp,  $sp,  -8
        sw    $ra,  4($sp)
        sw    $a0,  0($sp)
        slti  $t0,  $a0,  -1
        beq   $t0,  $0,   L1
        addi  Sv0,  $0,   1
        addi  $sp,  $sp,  8
        jr    $ra
L1:     addi  $t0,  $t0,  -1
        jal   FACT
        lw    $a0,  4($sp)
        lw    $ra,  0($sp)
        addi  $sp,  $sp,  8
        mul   $v0,  $a0,  $v0
        jr    $ra
```

Solution:

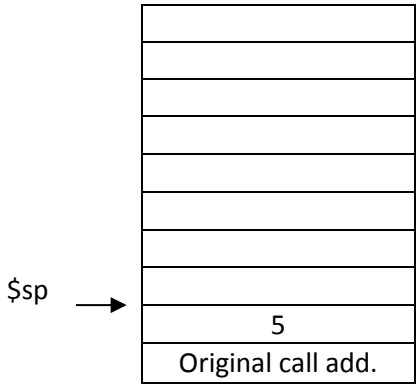
```
FACT:   addi   $sp,   $sp,   -8
        sw     $ra,   4($sp)
        sw     $a0,   0($sp)
        slti   $t0,   $a0,   1
        beq    $t0,   $0,    L1
        addi   Sv0,   $0,    1
        addi   $sp,   $sp,   8
        jr     $ra
L1:     addi   $a0,   $a0,   -1
        jal    FACT
        lw     $ra,   4($sp)
        lw     $a0,   0($sp)
        addi   $sp,   $sp,   8
        mul    $v0,   $a0,   $v0
        jr     $ra
```

The stack content:

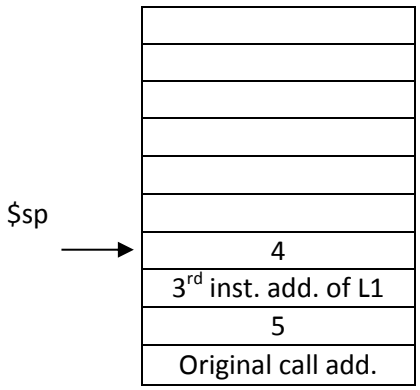
1. Originally:



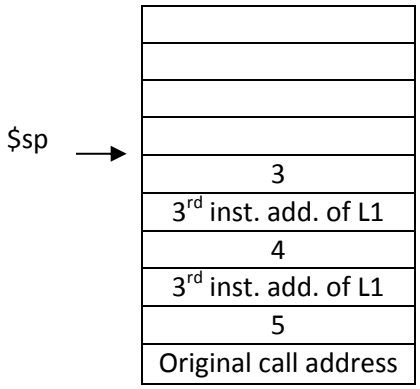
2. After the original call, i.e. calling Factorial(5), $a0 = 5 and $ra = original call address



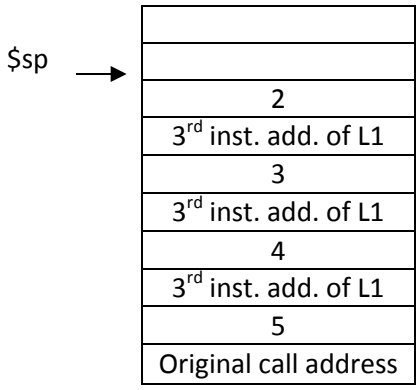
3. After the original call, i.e. calling Factorial(4), $a0 = 4 and $ra = address of third inst. In L1



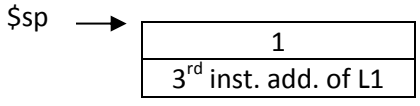
4. After the original call, i.e. calling Factorial(3), $a0 = 3 and $ra = address of third inst. In L1



5. After the original call, i.e. calling Factorial(2), $a0 = 2 and $ra = address of third inst. In L1



6. After the original call, i.e. calling Factorial(1), $a0 = 1 and $ra = address of third inst. In L1



2
3rd inst. add. of L1
3
3rd inst. add. of L1
4
3rd inst. add. of L1
5
Original call address

4. For the code below:

```
main()
{
        leaf_function(1)
}

int leaf_function (int f)
{
        int result;
        result = f + 1;
        if (f > 5)
                return result;
        return leaf_function(result);
}
```

   a. Write the corresponding MIPS code
   b. Assume that the stack is initially empty and that the stack pointer value is 0x7fff fffc. Functions inputs are passed using register $a0 and result returned in $v0. Assume that the functions may only use saved registers. Show the content of the stack after each function call.

Solution:

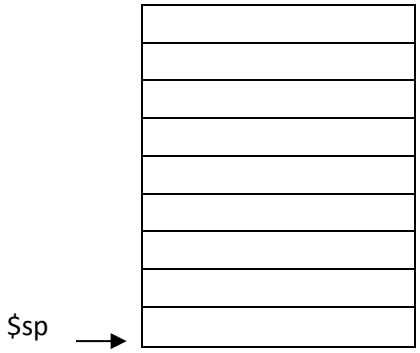
a.
```
Main:           addi $a0, $0, 1
                leaf_funco

Leaf_funco:     addi $sp, $sp, -4
                sw $ra, 0($sp)
                addi $v0, $a0, 1
                addi $t1, $0, 5
                slt $t2, $t1, $a0
                bne $t2, $0, end
                addi $a0, $v0, $0
                jal Leaf_funco
                j Exit

Exit:           lw $ra, 0($sp)
                addi $sp, $sp, 4
                jr $ra
```

b.
```
Main:           addi $a0, $0, 1
                leaf_funco

Leaf_funco:     addi $sp, $sp, -4
                sw $ra, 0($sp)
                addi $v0, $a0, 1
                addi $sp, $sp, -8
```

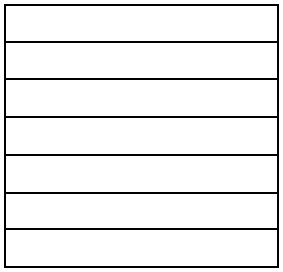
```
                sw $s0, 0($sp)
                sw $s1,4($sp)
                addi $s0, $0, 5
                slt $s1, $s0, $a0
                bne $s1, $0, end1
                addi $a0, $v0, $0
                lw $s0, 0($sp)
                lw $s1, 4($sp)
                addi $sp, $sp, 8
                jal Leaf_funco
                j End2

End1:           lw $s0, 0($sp)
                lw $s1, 4($sp)
                addi $sp, $sp, 8
                j End2

End2:           lw $ra, 0($sp)
                addi $sp, $sp, 4
                jr $ra
```

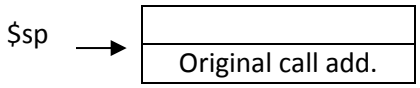
The stack content:

1. Originally:

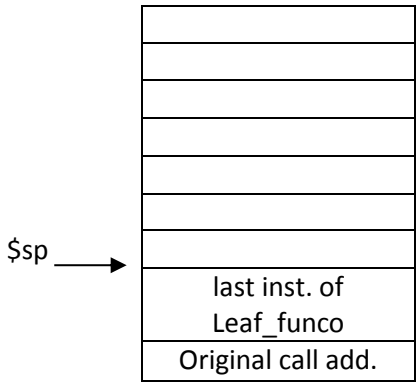


2. After the original call, i.e. calling Leaf_funco(1), $ra = original call address of Main

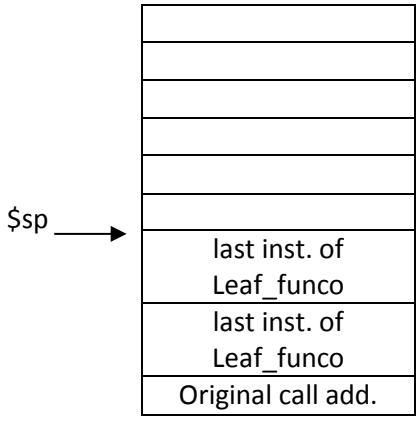




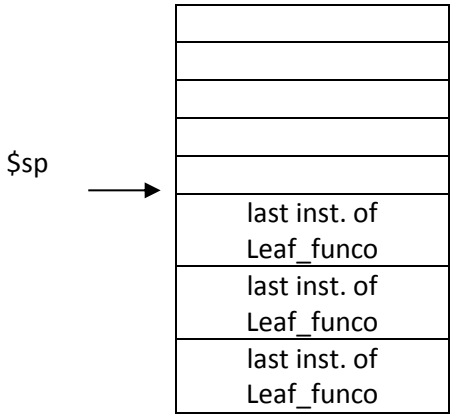
3. After the original call, i.e. calling Leaf_funco(2), $ra = address of last instruction of Leaf_funco

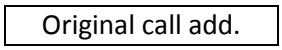


4. After the original call, i.e. calling Leaf_funco(3), $ra = address of last instruction of Leaf_funco

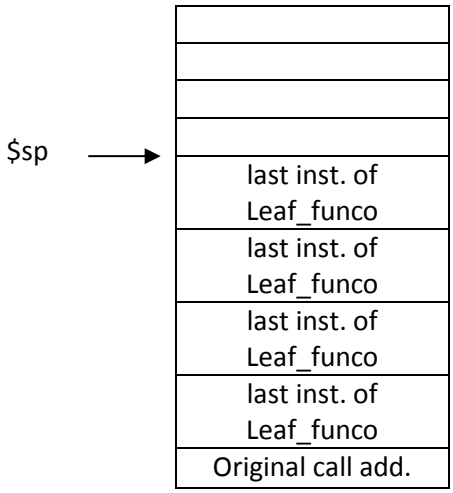


5. After the original call, i.e. calling Leaf_funco(4), $ra = address of last instruction of Leaf_funco





6. After the original call, i.e. calling Leaf_funco(5), $ra = address of last instruction of Leaf_funco



7. After the original call, i.e. calling Leaf_funco(6), $ra = address of last instruction of Leaf_funco