99

CSCI-21 Midterm #2, fall 2019 100 Name Ismael Cortez

Closed books, closed notes except for the note sheet I gave you, no electronic devices of any type. Please use pencil and erase mistakes. If you need more room, use the back of the PRECEEDING sheet (so you and I can see both at the same time). For the preceding sheet for this page, use the back of the last sheet. 100 points.

8/

Short answer – three points each except as otherwise specified

1. What is a *code block*?

A code block is a list of instructions that has a single entry point and a single exit point.

3

it is not a convention, it is a block of ? RAM on the stack?

2. What is a stack frame?

A stack frame (activation record) is a convention used in languages like C and C++ that defines where data can be stored and accessed by the caller and callee when calling subroutines.

2

3. Why do we use stack frames?

We use stackframes to free up register space, pass in parameters, and store local variables for each activation of a subroutine.

3

ok close

$fp lets $sp change?

8/

4. What is a *calling convention*?

3

A calling convention is an agreement about how subroutines are called and how control is returned to the caller.

OK

register use?
argument/parameter passing?
return values?
register use?

5. What does it mean to say that two computers have the same *computing power*?

Two computers have the same computing power if they can run the same programs (after being translated to their respective machine code) and produce the same results.

3

6. What do we call an operation that puts an item onto a stack? (1 point)

1

Push

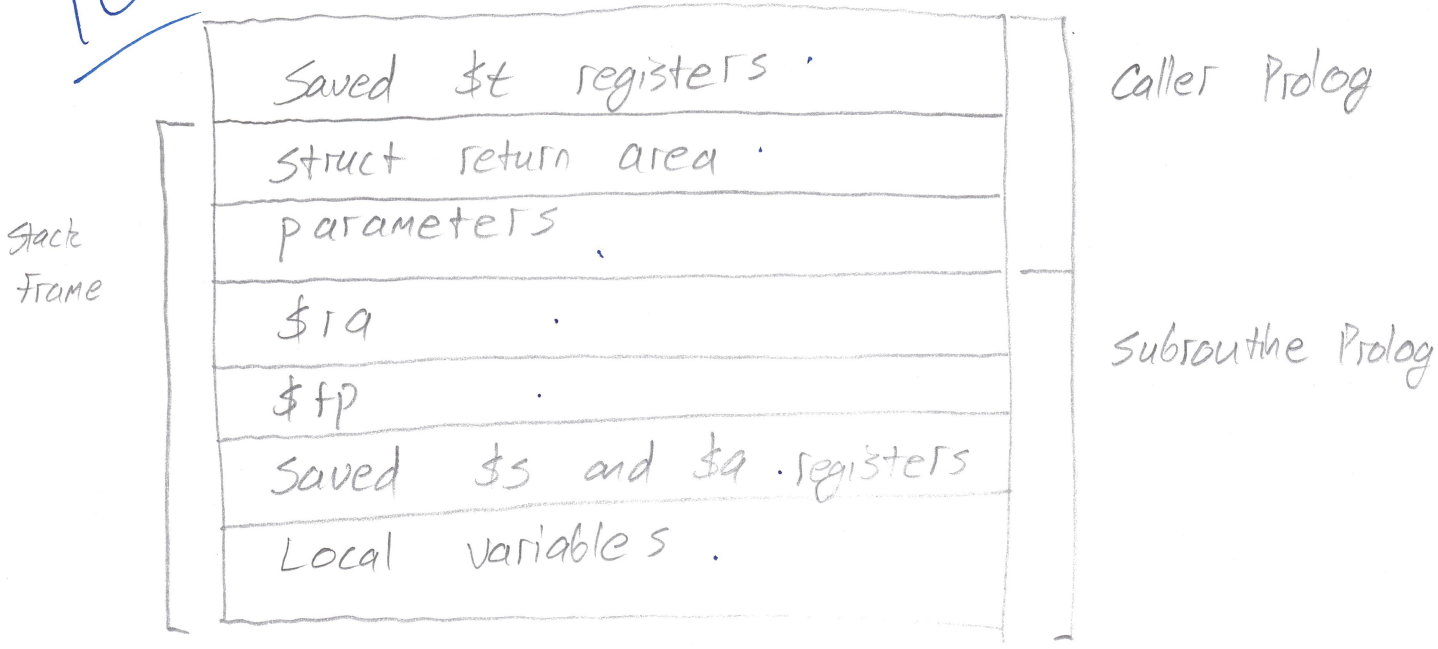
7. What do we call an operation that removes an item from a stack? (1 point)

1

Pop

8. Draw a diagram of a stack frame (as would be used in a compiled programming language like C or C++), showing the all of the things put onto the stack, the limits of the actual stack frame and which code builds which parts. (10 points)

10



$n! = n \times (n-1)!$

9. Write a recursive subroutine to compute the value n factorial (n!), with integer n passed in as a parameter on a real-world style stack frame. Write the subroutine only, NOT a complete program. (20 points)

20

```
fact:
        # subroutine Prolog
        addiu   $sp, $sp, -4        # Push callers $ra
        sw      $ra, ($sp)
        addiu   $sp, $sp, -4        # Push callers $fp
        sw      $fp, ($sp)
        move    $fp, $sp            # No variables, $fp = $sp
        # Subroutine Body
        lw      $t0, 8($fp)         # Get N
        li      $t1, 1              # Constant of 1 use $v0 and save some code?
        bgt     $t0, $t1, recur     # if (N > 1)
        nop
        li      $v0, 1              # else N<=1, return 1
        j       epilog
        nop
recur:
        # caller Prolog
        addiu   $t0, $t0, -1        # Compute N-1
        addiu   $sp, $sp, -4        # Allocate space for N-1
        sw      $t0, ($sp)
        jal     fact                # Recursive call
        nop
        # caller Epilog
        addiu   $sp, $sp, 4         # Deallocate space for N-1
        lw      $t0, 8($fp)         # Restore N
        nop
        mult    $t0, $v0            # $v0 = N x fact(N-1)
        mflo    $v0
epilog:
        move    $sp, $fp            # why?
        lw      $fp, ($sp)          # Pop callers $fp
        addiu   $sp, $sp, 4
        lw      $ra, ($sp)          # Pop callers $ra
        addiu   $sp, $sp, 4
        jr      $ra                 # Return to caller
        nop
```

10. Write a complete MIPS program that prompts the user for an integer with reasonable text, calls the factorial subroutine you wrote in question 9, and then (afterward!) prints the entered integer and the result with reasonable descriptive text. Do not rewrite the subroutine. Do not print the entered integer before calling the subroutine. (20 points)

20

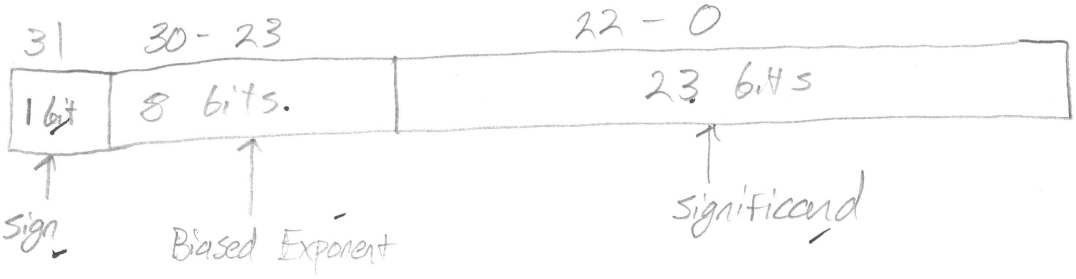
```
        .data
prompt: .asciiz "Enter N to calculate N!: "
msg:    .asciiz "! = "
n:      .word   0
        .text
        .globl  main

main:
        la      $a0, prompt         # Output prompt to user
        li      $v0, 4
        syscall
        li      $v0, 5              # Get N
        syscall
        sw      $v0, n              # store N for later use
        # Caller Prolog
        addiu   $sp, $sp, -4        # Allocate space for N
        sw      $v0, ($sp)          # Call with JAL
        jal     fact
        nop
        # Caller Epilog
        addiu   $sp, $sp, 4         # Deallocate space for N
        move    $s0, $v0            # Hold result
        lw      $a0, n              # Output N to user
        li      $v0, 1
        syscall
        la      $a0, msg            # output descriptive text
        li      $v0, 4
        syscall
        move    $a0, $s0            # output result
        li      $v0, 1
        syscall
        li      $v0, 10             # End the program
        syscall
```

18

11. Listing in sequence from high-order to low-order bits, name and give the size of all the fields of the IEEE 754 standard for single-precision floating-point numbers. (4 points)

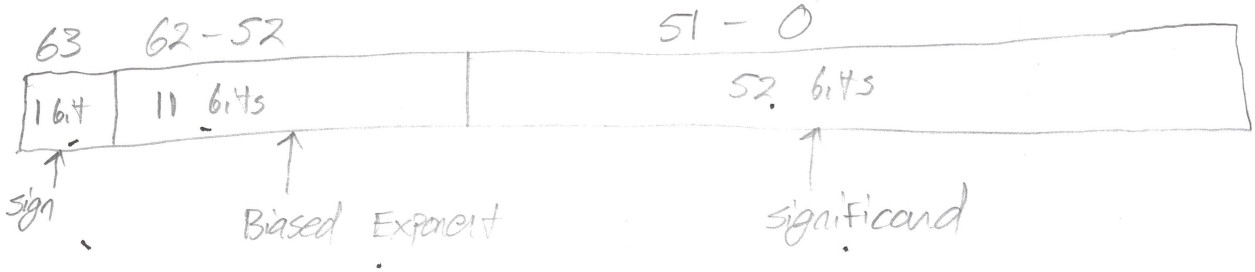
4



list = ?

12. Listing in sequence from high-order to low-order bits, name and give the size of all the fields of the IEEE 754 standard for double-precision floating-point numbers. (4 points)

4



13. Convert the following values from decimal to IEEE-754 standard single-precision floating point numbers. Show your work (if you need room, use the back of the preceding page), and write the answers in hexadecimal (5 points each)

a. 35.6875

5

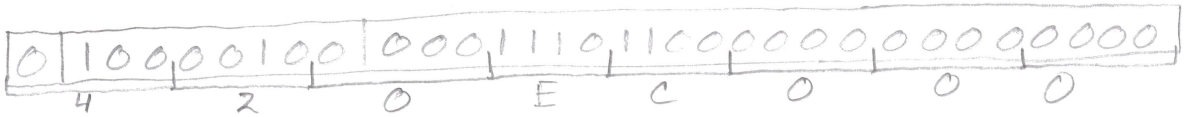
128	64	32	16	8	4	2	1
0	0	1	0	0	0	1	1

	.6875
1	.3750
0	.7500
1	.5000
1	.0000

35.6875 = 00100011.1011 = 1.00011011×2^5

Biased Exponent: 127+5 = 128+4 = 10000100

0x420EC000

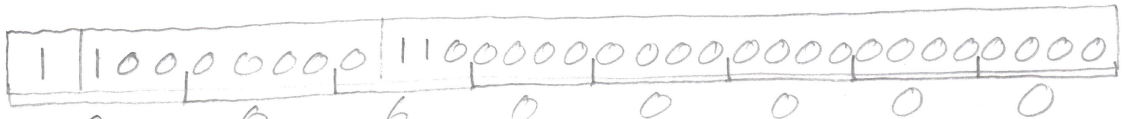


b. -3.5

5

3 = 0011, $\frac{.5}{1.0}$ 3.5 = 11.1 = 1.11×2^1

Biased Exponent = 127+1 = 128 = 10000000



0xC0600000

15

14. Convert the following IEEE-754 standard single-precision floating point numbers from hexadecimal to decimal. Show your work and/or thinking process. If you need room, use the back of the preceding page. (3 points each)

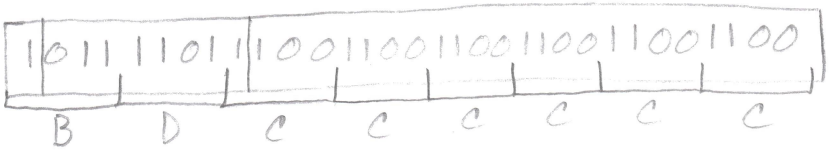
a. 0x00000000

Sign: 0, Biased Exponent: All 0s, Significand: 0s

3

+0

b. 0xBDCCCCCC



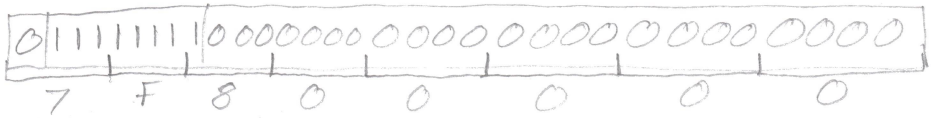
3

Sign: 1, Biased Exponent: 64+32+16+8+3 = 112+8+3 = 123 = 127 + (-4)

$1.0011001\ldots \times 2^{-4} = 0.00001001100\ldots$ special case of −0.1

(not really special - it's just as close as the machine can get!)

c. 0x7F800000

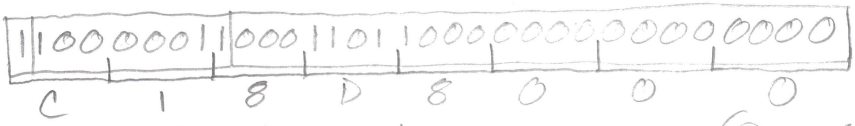


3

Sign: 0, Biased Exponent: All 1s

+∞

d. 0xC18D8000

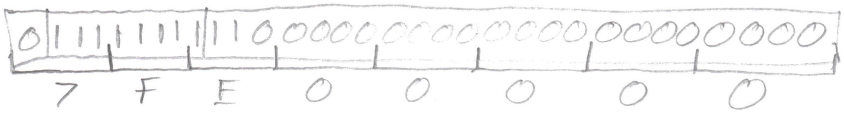


3

Sign: 1, Biased Exponent: 128+3 = 127+(4), Significand: 00011011

$-1.00011011 \times 2^4 = -10001.1011 =$ −17.6875

e. 0x7FE00000



3

Sign: 0, Biased Exponent: All 1s, Significand: Leading 1

special case of Quiet NaN