Midterm Retrospective and Recursion

CS230 Tutorial 07
Midterm 4a - Floating Point

For this question, assume the standard IEEE single-precision 32-bit floating point representation (one sign bit, 8 exponent bits, 23 fraction bits, bias 127).

Show the (1) normalized binary floating point representation (1.xxxx.. * 2^y), (2) binary number with sign and binary point (1xxxx.xxx), and (3) the decimal value of the following value (Steps not necessary):
1 10000110 11001011100000000000000

(1) First check the exponent: it’s not all ones, and it’s not all zeros, so we know this is a normal number.
Convert the exponent (10000110) to decimal: 128 + 4 + 2 = 134
Now we can use the formula (-1)^(1) x 1.110010111 x 2^{134-127} = -1.110010111 x 2^7

(2) Shift the decimal 7 places to the right: -11100101.11

(3) Convert from binary to (negative) decimal: 0 - (128 + 64 + 32 + 4 + 1 + 0.5 + 0.25) = -229.75
Midterm 4b - Floating Point

For this question, assume the standard IEEE single-precision 32-bit floating point representation (one sign bit, 8 exponent bits, 23 fraction bits, bias 127).

Convert the decimal number 0.5 to its raw floating-point binary representation (Raw implies that the final answers should consist only of 0s and 1s and no signs or points). Show all steps.

First convert 0.5 from decimal to binary: 0.1
Now normalize: 1.0 x 2^{-1}
Solve for the exponent bits: E - 127 = -1 therefore E = 126
Now we have the equation: (-1)^0 x 1.0 x 2^{126-127}
Break out the bits: S = 0   E = 01111110   F = 000000000000000000000000
Raw bits: 00111111000000000000000000000000
Write a MIPS program for the C code shown below. The code performs factorial of a given number ‘n’ using a loop. (The program should not use other methods for doing factorial, such as recursion.) Assume $4$ stores the function argument. Use $2$ as the register for storing the return value, $8$ for $x$, $9$ for $j$.

```c
int fact_o(int n) {
    x=1;
    j=n;
    while(j !=0) {
        x = x*j;
        j=j -1;
    }
    return x;
}
```
addi $8, $0, 1 ; x = 1
add $9, $4, $0 ; j = n
L1: beq $9, $0, L2 ; if (j == 0), goto L2
mult $8, $9 ; x * j
mflo $8 ; x = x * j
addi $9, $9, -1 ; j = j - 1
j L1 ; goto L1 (could be beq $0, $0, L1)
L2: add $2, $8, $0 ; return val in $2
jr $31 ; exit
The MIPS code for the following C code has some missing instructions. Please fill out the missing MIPS instructions. Each blank pertains to exactly one MIPS instruction. $17$ corresponds to variable ‘$i$’, and $8$ is used as temporary register. Use $4$ as the register for the function argument, $2$ as the register for return value, $30$ as the stack pointer, $31$ as the return address register.

```c
int A() {
    int i;
    i = 3;
    i = B(i);
    return 0;
}

int B(int k) {
    return k+131;
}
```
int A() {
    int i;
    i = 3;
    i = B(i);
    return 0;
}

int B(int k) {
    return k + 131;
}

addi $17, $0, 3
; pass the argument
addi $8, $4, 131
; return value
add $17, $2, $0
; rest. ret. addr 1
add $17, $0, $0
; rest. ret. addr 2
jr $31

jr $31
int A() {
    int i;
    i = 3;
    i = B(i);
    return 0;
}

int B(int k) {
    return k + 131;
}

addi $17, $0, 3
add $4, $17, $0
addi $30, $30, -4
sw $31, 0($30)
jal B
add $17, $2, $0
lw $31, 0($30)
addi $30, $30, 4
jr $31

B: addi $8, $4, 131
add $2, $8, $0
jr $31
Recursion

A recurrent expression is an expression that contains itself as a member. Recurrence relations have (at minimum) one base case, and one recursive case. This is the same concept as recursion in Racket.

Factorial:
- Base case: $\text{Fac}(0) = 1$
- Recursive case: $\text{Fac}(n) = n \times \text{Fac}(n - 1)$

Fibonacci:
- Base cases: $\text{Fib}(0) = 1 \quad \text{Fib}(1) = 1$
- Recursive case: $\text{Fib}(n) = \text{Fib}(n - 1) + \text{Fib}(n - 2)$
Recursion Implementation

To implement recursion we make a function that calls itself. If the function gets a base case input, then we return the base case. If the function gets a recursive case, it calls itself again.

Follow along with the following example:

Write a function $\text{Fib}(n)$ that calculates the $n^{\text{th}}$ digit of fibonacci (starting from zero). Assume $n$ is in register $\$1$ at the beginning of the program. Place the result in register $\$3$ at the end of the program. Remember that standard convention has the first function argument in register $\$4$ and return value in register $\$2$. 
Recursive Fibonacci Solution

```assembly
addi $4, $1, 0 ; move n to $4
addi $30, $30, -4
sw $31, 0($30) ; save ret. addr
jal Fib ; call function
lw $31, 0($30) ; rest. ret. addr
addi $30, $30, 4 ; move result $3
jr $31 ; exit

Fib:
beq $4, $0, base ; check for Fib(0)
addi $8, $0, 1
beq $4, $8, base ; check for Fib(1)
addi $30, $30, -12 ; reserve 3 stack spots
sw $31, 4($30) ; save ret. addr
sw $4, 0($30) ; save argument
addi $4, $4, -1 ; call Fib(n-1)
jal Fib
sw $2, 8($30) ; save result
addi $4, $4, -1 ; call Fib(n-2)
jal Fib
lw $8, 8($30) ; load saved result
add $2, $8, $2 ; Fib(n-1) + Fib(n-2)
lw $4, 0($30)
lw $31, 4($30)
addi $30, $30, 12
jr $31

base:
addi $2, $0, 1 ; Fib(0) and Fib(1)
jr $31 ; are both 1
```
Recursion Exercise

Try coding the following exercise:

Write a function $\text{Fact}(n)$ that calculates the $n!$ ($n$ factorial). Assume $n$ is in register $1$ at the beginning of the program. Place the result in register $3$ at the end of the program. Remember that standard convention has the first function argument in register $4$ and return value in register $2$. 
Recursive Factorial Solution

```
addi $4, $1, 0 ;move n to $4
addi $30, $30, -4
sw $31, 0($30) ;save ret. addr
jal Fib ;call function
lw $31, 0($30)
addi $30, $30, 4 ;rest. ret. addr
addi $3, $2, 0 ;move result $3
jr $31 ;exit

Fact:
beq $4, $0, base
addi $30, $30, -8 ;reserve 2 stack spots
sw $4, 0($30) ;save argument
sw $31, 4($30) ;save ret. addr
addi $4, $4, -1 jal Fact
addi $4, $4, 1
jal Fact
mult $4, $2
mflo $2
lw $4, 0($30)
lw $31, 4($30)
addi $30, $30, 8
jr $31

base:
addi $2, $0, 1
jr $31

;check for Fact(0)
;reserve 2 stack spots
;save argument
;save ret. addr
;call Fact(n-1)
;multiply n*Fact(n-1)
;get multiply result
;restore return addr
;return
;Fact(0) is 1
```
Assignment reminders

- Always test your code on the university servers before you hand it in
- Hand in the .asm file (*not the .mips file*)