

Virtual Memory

Consider a call to the following MIPS code (no delay slots) with the given initial page table. Assume that pages are 4KiB and that all page faults (but not protection faults) can be serviced by the OS without evicting pages. \$sp is initially 0x6004, \$ra is initially 0x1040, and \$a0 is initially 0x1.

```

MIPS
V.A.      Instructions
0x2004    Foo: addiu $sp, $sp, -4
0x2008    sw $ra, 0($sp)
0x200C    beq $a0, $0, Skip
0x2010    addiu $a0, $a0, -1
0x2014    jal Foo
0x2018    Skip: lw $ra, 0($sp)
0x201C    addiu $sp, $sp, 4
0x2020    jr $ra
    
```

Valid	Dirty	A. R.	P.P.N.
0	0	None	4
1	0	Read, Exec	5
0	0	Read, Exec	1
0	0	None	1
0	0	Read, Write	12
1	0	Read, Write	3
1	0	Read, Write	2
...

1. Where will page faults occur in this function's execution?
2. Assuming that we don't have a TLB, (or that all the TLB was flushed), what will be in the page table after this function is completely executed?

Valid	Dirty	A. R.	P.P.N.
...

3. Suppose \$a0 were initially 0xC00 instead of 0x1, what other exceptions can occur?

MapReduce

Use pseudocode to write MapReduce functions necessary to solve the problems below. Also, make sure to fill out the correct data types. Some tips:

- The input to each MapReduce job is given by the signature of the **map()** function
- The function **emit(key k, value v)** outputs the key-value pair **(k, v)**
- You may use the **for(var in list)** syntax to iterate through iterable types, or use the **next()** and **hasNext()** methods of iterable types
- You may also use **sum()**, **length()**, or **sort()** on collections of values
- Data types you may use are **Integer**, **Float**, **String**, **List**, and any custom data types you might define yourself

1. Given a set of classes that students have taken, output each student's name and total GPA.

Declare any custom data types here: CourseData: Integer courseID Float studentGrade // a number form 0-4	
map(String student, CourseData value): 	reduce(_____ key, Iterable<_____> values):

2. Compute the list of mutual friends between each pair of friends in a social network. Each person on the network is identified by a unique **Integer** ID. You can use an **intersection(list1, list2)** method that returns a list that is the intersection of **list1** and **list2**.

Declare any custom data types here: 	
map(Integer personID, List<Integer> friendIDs): 	reduce(_____ key, Iterable<_____> values):

3. Given a set of coins and each coin's owner, compute the number of coins of each denomination that each person has.

Declare any custom data types here:	
map(String person, String coinType):	reduce(_____ key, Iterable<_____> values):

4. Using the output of the previous MapReduce job, compute the amount of money each person has. The function **valueOfCoin(String coinType)** returns a float corresponding to the dollar value of the given coin.

map(_____ key, _____ value):	reduce(_____ key, Iterable<_____> values):
---	---