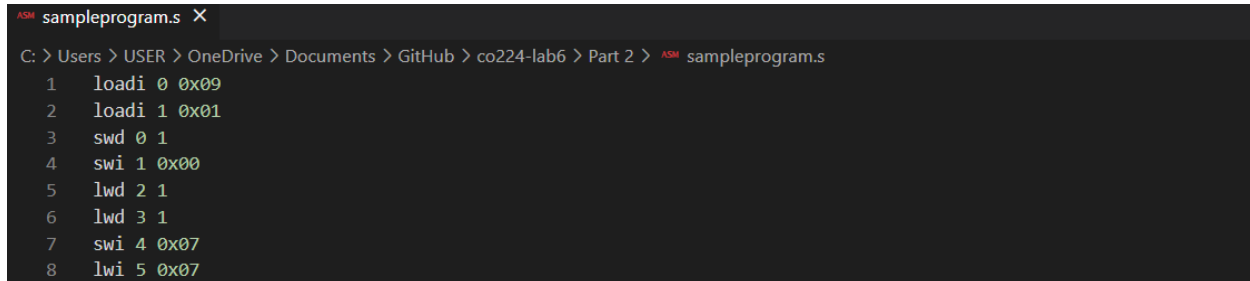


## CO224 – Software Architecture

### Group 34 : E/19/003, E/19/443

#### Lab 6 : Part 2

"When reading or writing for the first time, a system with a cache may not be as efficient as a cacheless system. This is because the system with a cache needs to load data blocks from memory to perform the operation, which takes more time than a single-word read.



```
sampleprogram.s X
C: > Users > USER > OneDrive > Documents > GitHub > co224-lab6 > Part 2 > sampleprogram.s
1  loadi 0 0x09
2  loadi 1 0x01
3  swd 0 1
4  swi 1 0x00
5  lwd 2 1
6  lwd 3 1
7  swi 4 0x07
8  lwi 5 0x07
```

In the specific sample program, the instructions primarily involve accessing a small number of memory locations repeatedly, such as register 1. If the system has a cache, initially these memory locations will not be present in the cache, leading to cache misses. This results in the need to fetch data from the main memory and load it into the cache.

In a cacheless system, there is no cache to manage, and each memory access directly fetches data from the main memory. Since the program's memory access pattern involves a limited number of memory locations being accessed repeatedly, a cacheless system can potentially perform better.

However, once the cache is loaded, subsequent reads or writes that result in cache hits can be completed within one clock cycle. The only time consumed is for operations such as extracting tag, valid bits, and dirty bits, tag comparison, and writing data to the cache.

Therefore, if a program achieves a high hit rate in the cache, the system with a cache is very efficient. However, if there are a lot of cache misses, a system without a cache may consume less time to complete instructions compared to a system with a cache.

In the given program, since the memory access pattern is limited and does not exhibit frequent access to the same memory locations, the potential benefits of a cache may be limited. A cacheless system could potentially perform similarly or even faster due to the avoidance of cache management overhead.

If we consider the given program and introduce additional instructions that exhibit different memory access patterns, we can explore how a cache might be useful. Let's extend the program with the following instructions:

lwd 6 1: This instruction loads a word from the memory location specified by register 1 into register

lwd 7 2: This instruction loads a word from the memory location specified by register 2 into register

lwd 8 3: This instruction loads a word from the memory location specified by register 3 into register

lwd 6 1: If the memory location accessed by register 1 (previously written by swd 0 1) is present in the cache (cache hit), the load instruction can be completed faster. The cache would store the data from the previous store operation, reducing the need for main memory access.

lwd 7 2: Similarly, if the memory location accessed by register 2 (previously written by lwd 3 1) is present in the cache (cache hit), the load instruction can be completed faster.

lwd 8 3: Again, if the memory location accessed by register 3 (previously written by lwi 5 0x07) is present in the cache (cache hit), the load instruction can be completed faster.

In this extended program, if the cache effectively retains the frequently accessed memory locations, subsequent load instructions can benefit from cache hits and complete faster compared to a cacheless system. The cache helps reduce the latency associated with main memory access, especially when data exhibits temporal and spatial locality.

It's important to note that the size of the cache and the specific cache replacement policies employed can affect the cache's effectiveness. Proper cache management and appropriate cache configurations, such as choosing an optimal cache size and replacement policy, can maximize the cache's benefits and improve overall performance.