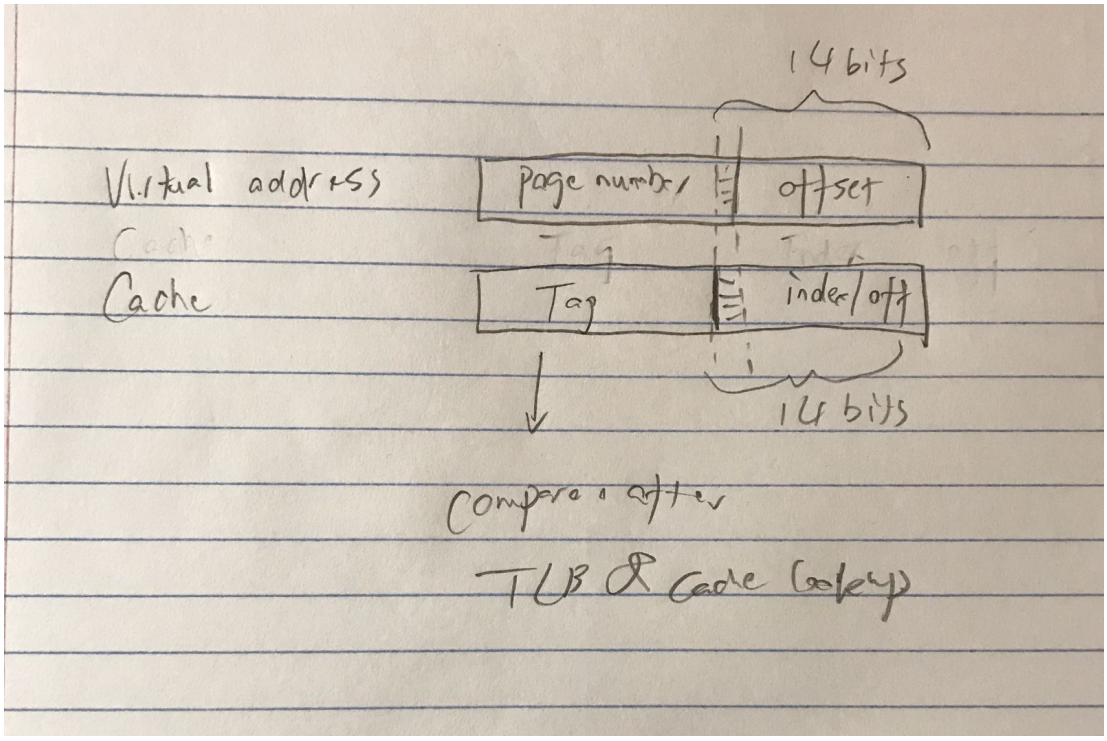
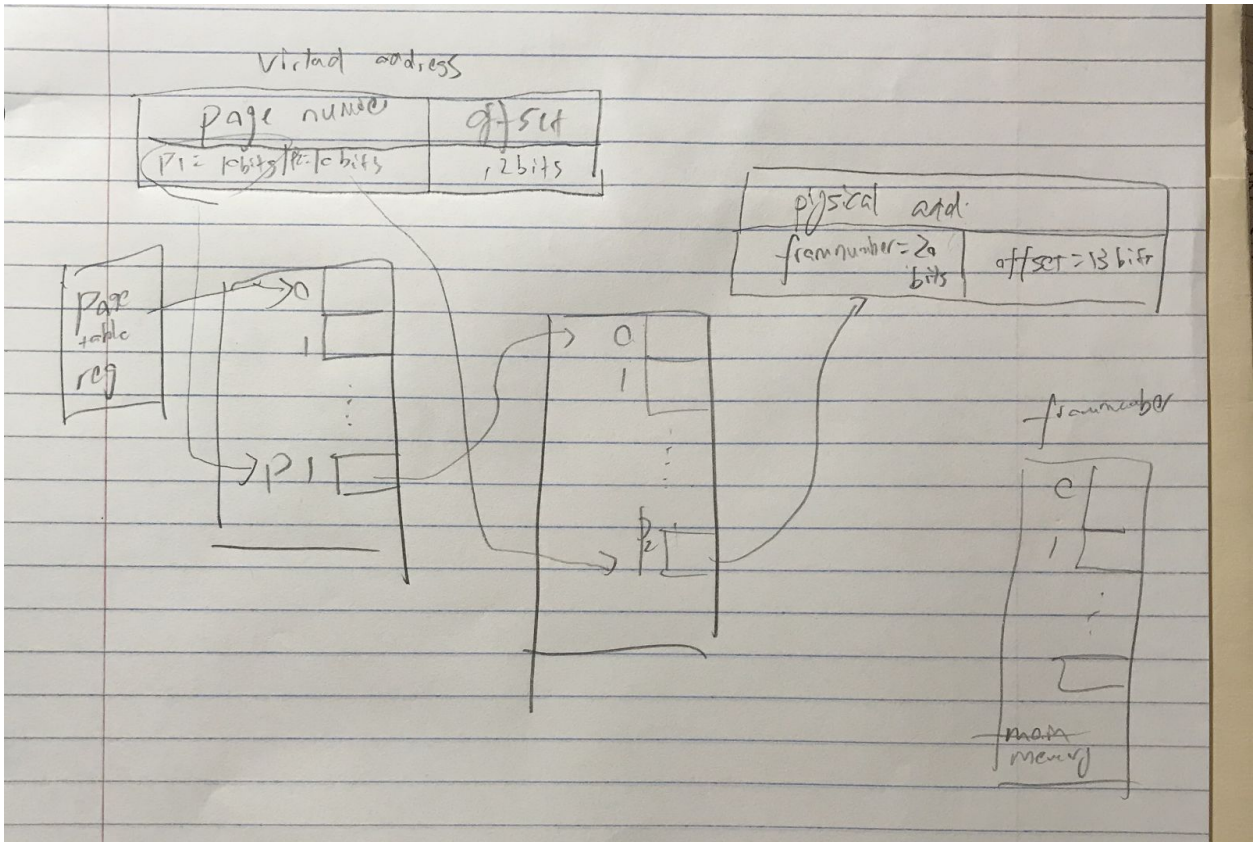
Virtual Memory Translation

- In this assignment, we will build a virtually-indexed and physically tagged cache.
  For index part, we don't need to read through TLB because we just use virtual address for indexing.
  As the assignment requires, we need to make the low-order of 14bits of a virtual page number matches the cache's low-order 14 bits.



  As the diagram show, we can directly copy the offset part of virtual address into the index and offset part of the cache. However, we want to make sure the two shadow parts in the diagram are still the same. Therefore, we will make sure the last few bits of the virtual page number is the same as the last few bits of the physical frame number.
  In our program, page size= 4KB means we have 12bits offset in virtual address and
  14 - 12 = 2 bits
  Thus, we need to match the lower 2 bits of virtual page number to the lower 2 bits of the physical page number.

- Two-level page table



As the diagram show, we make each level's table an array. The parameter is in page table 1 is the content of p1 and the parameter in page table 2 is the content of p2. The content of p1 is a cursor which points to the beginning of page table 2. The content of p2 is a frame number and we take it out. We combined it with the offset of the virtual address and we have a physical address.

- Implementation
  - Define a model for TLB.
    Since both I_TLB and D_TLB has the same structure. We define the TLB in this way:

```
struct model_TLB{
      int valid;
      int tag;
      int index;
      uint32_t physicalPageNumber;
      uint32_t virtualPageNumber;
}
```

  - Declare two two array that contain two TLBs

```
//  declare the global cache for TLB, each has 8 entries
struct model_TLB I_TLB[8];
struct model_TLB D_TLB[8];
```

- Function that use X_TLB to look for physicalAddress

```
/*
     FUNCTION: TLB_searching
     PURPOSE: read through the TLB, try to give the
address needed.
     PARAMETERS: base address from PTBR, page number from
vitual address, struct model to update
*/
uint32_t TLB_searching(struct model_TLB TLB[], int tag,
int index, int offset){
     struct model_TLB temp = TLB[index];
     if( (temp->tag == tag) && (temp->valid == 1))
          uint32_t physicalAddress =
temp->physicalPageNumber << 12;
          physicalAddress = physicalAddress | offset;
          return physicalAddress;
}
```

- If the previous function cannot return a physical address:

```
new_?_reg->stall = true;
```

Update the TLB by using page table, then fetch it again next cycle;

- Codes for page table

```
/*
     page table 1
     DEFINE: an entry to built up with pointers(addresses)
pointing to the base of page table 2
*/
struct model_pageTable1{
     int valid;
     uint32_t * pt2_base;
}
uint32_t *pageTable_1[1024];

/*
     FUNCTION: run_pt1
     PURPOSE: look for the page table 1 to obtain the base
address of page table 2
     PARAMETERS: base address from PTBR, page number from
vitual address
*/
uint32_t* run_pt1(uint32_t * base, int pageNumber){
```

```
        uint32_t * ptr;
        calloc a space to insert new entry with values
    provided;
        return ptr;
    }

    /*
        FUNCTION: update_pt1
        PURPOSE: update value in page table 1
        PARAMETERS: base address from PTBR, page number from
    vitual address, struct model to update
    */
    void update_pt1(uint32_t * base, int pageNumber, struct
    model_pageTable1 mpt){
        Look for physical address
    }

    // declare page table 2
    uint32_t pageTable_2[1024];
```

Task assignment

- Hang Yuan
    - The basic structure of the program
    - Debugging
    - Writing design doc.
- Zhewei Wang
    - The basic structure of the program
    - Debugging
    - Writing design doc.

- Andrew Tsai
    - Debugging

Testing Plan

See README