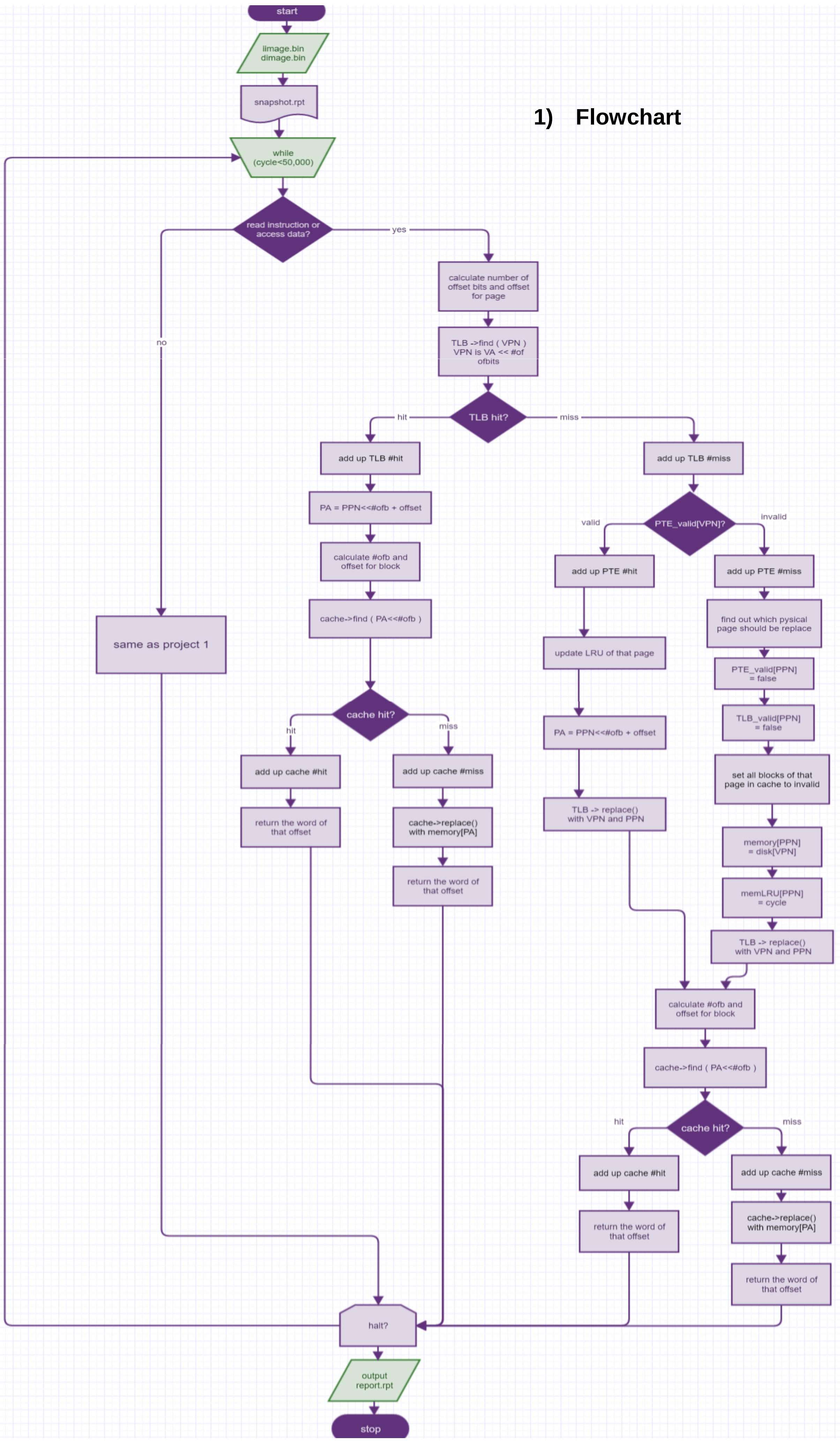
1) Flowchart


start
iimage.bin
dimage.bin
snapshot.rpt
while
(cycle<50,000)
read instruction or
access data?
yes
no
calculate number of
offset bits and offset
for page
TLB ->find (VPN)
VPN is VA << #of
ofbits
TLB hit?
hit
miss
add up TLB #hit
PA = PPN<<#ofb + offset
calculate #ofb and
offset for block
cache->find (PA<<#ofb)
cache hit?
hit
miss
add up cache #hit
return the word of
that offset
add up cache #miss
cache->replace()
with memory[PA]
return the word of
that offset
add up TLB #miss
PTE_valid[VPN]?
valid
invalid
add up PTE #hit
update LRU of that page
PA = PPN<<#ofb + offset
TLB -> replace()
with VPN and PPN
add up PTE #miss
find out which pysical
page should be replace
PTE_valid[PPN]
= false
TLB_valid[PPN]
= false
set all blocks of that
page in cache to invalid
memory[PPN]
= disk[VPN]
memLRU[PPN]
= cycle
TLB -> replace()
with VPN and PPN
calculate #ofb and
offset for block
cache->find (PA<<#ofb)
cache hit?
hit
miss
add up cache #hit
return the word of
that offset
add up cache #miss
cache->replace()
with memory[PA]
return the word of
that offset
same as project 1
halt?
output
report.rpt
stop

2) Project Description

1. **Cache**
   a. Implement through class,
   b. With following content :
      i. Array of inner class **Cache_Unit**, each include :
         1. Valid, MRU, tag, Data
      ii. total_size : total size of the cache,
      iii. block_size : size of each block,
      iv. n_way : #-way-associative ,
      v. set_num : the number of sets in the cache,
      vi. miss, hit : the total hit and miss of the cache,
   c. find() :
      i. The function will find out which block the desired data was stored, or return -1 if miss,
      ii. Two parameters: physical address and a Boolean to verify whether should set MRU or not,
      iii. Calculate index and tag of the physical address, and search through every blocks of that set to know if there's a valid block that tag is matched,
      iv. Set the MRU to 1 if block is found and valid, and check if need to reset MRU of the set,
   d. replace()
      i. The function will store the data into the cache according to its physical address,
      ii. Two parameters: physical address and the data,
      iii. Calculate index and tag of the physical address
      iv. Search through every blocks of the set, replace the first invalid block if exist, otherwise replace the one which MRU is 0
      v. Set the MRU of the block to 1, and check if need to reset MRU of the set;

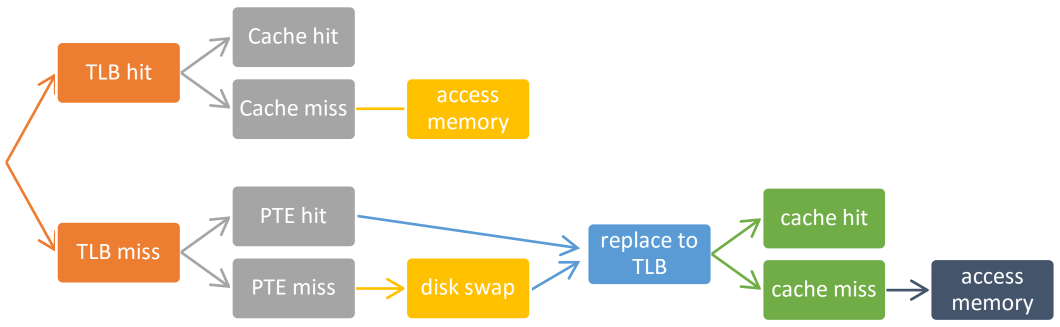
2. **TLB**
   a. Implement through class,
   b. Contain following content :
      i. Array of inner class **TLB_Unit**
         1. valid,
         2. tag : the virtual page number that the unit represent,
         3. LRU : last used cycle,

        4. ppn : physical page number of the unit
    ii. size : number of entries of TLB
    iii. page_size : the size of each page
    iv. hit, miss : the total hit and miss of the TLB,
  c. find()
    i. The function will find the physical page number of the virtual page number, or return -1 if miss,
    ii. Two parameters: virtual page number and current cycle,
    iii. Search through every block of TLB to know if there’s a valid block that tag matches the virtual page number,
    iv. Set the LRU to the cycle if found,
  d. replace():
    i. The function will find the physical page number of the virtual page number,
    ii. Three parameters: virtual page number, relative physical page number and current cycle,
    iii. Search through all unit of TLB to , replace the invalid one if exist, or replace the one with least LRU (haven’t been use for the longest cycle)
    iv. Set the LRU to the cycle number;

3. **Memory**
  a. Implement through an array of data and an array recoding LRU,
  b. The physical address is the index that the data store in array,
  c. LRU array store the last-used cycle of each physical page,
  d. When coping data from disk, copy to the first empty entry, or store to least LRU entry;

4. **Page Table**
  a. Implement through an array storing physical page number and an array for valid,
  b. The size of page table is the total page that disk have (1024/page size),
  c. The index of each entry representing each virtual page number, the content of each entry is it’s physical page number ,
  d. If a virtual page is store in memory, the valid of the entry is true,

5. Detail of reading instruction or data



   a. find out TLB hit or miss by function find() of TLB,
   b. find out cache hit or miss by function find() of cache,
   c. access memory and replace to cache by function replace() of cache,
   d. replace TLB by function replace() of TLB,
   e. disk swap :
      i. find out which physical page that the desire data should be put in (the least LRU entry of memory),
      ii. set the physical page that will be replace in page table, TLB and cache to invalid (by using find() to look up all blocks of that page),
      iii. Copy the whole page from disk to the physical page of the memory,
   f. Get the word in the block by offset and return it
   g. Pack this whole process into a function, and replace the original way to access data or instruction in project 1;

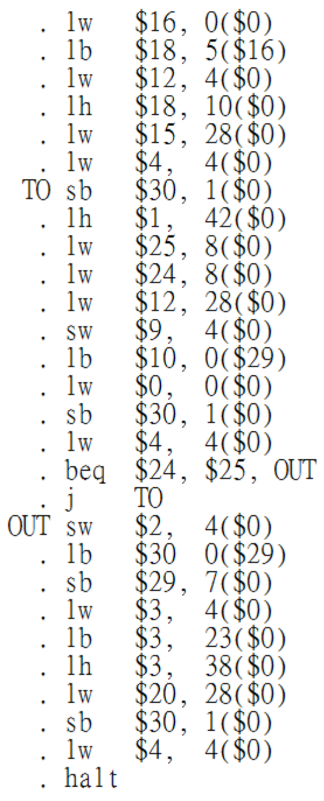
6. Detail of writing data
   a. The flow and most mechanism is same as reading data,
   b. But change parts that are accessing data to storing data, will directly write through memory and cache only when cache miss
   c. Pack this whole process into a function, and replace the original way to store data in project 1;

7. All other mechanism and instruction implement remains the same as project 1, only error detect been deleted.

3) Test case Design

1. dimage.bin
   a. random create values;

2. iimage.bin
   a. since the project is all about accessing data and instructions, only use instructions are relative to,
   b. make sure have test all lw, lh, and lb with varies of address
   c. to avoid data misaligned, use $zero as base so I can simply control which address that instruction will access for lw and lh
   d. put some branches so that instruction page table must swap pages



```
    . lw   $16, 0($0)
    . lb   $18, 5($16)
    . lw   $12, 4($0)
    . lh   $18, 10($0)
    . lw   $15, 28($0)
    . lw   $4,  4($0)
   TO sb   $30, 1($0)
    . lh   $1,  42($0)
    . lw   $25, 8($0)
    . lw   $24, 8($0)
    . lw   $12, 28($0)
    . sw   $9,  4($0)
    . lb   $10, 0($29)
    . lw   $0,  0($0)
    . sb   $30, 1($0)
    . lw   $4,  4($0)
    . beq  $24, $25, OUT
    . j    TO
  OUT sw   $2,  4($0)
    . lb   $30  0($29)
    . sb   $29, 7($0)
    . lw   $3,  4($0)
    . lb   $3,  23($0)
    . lh   $3,  38($0)
    . lw   $20, 28($0)
    . sb   $30, 1($0)
    . lw   $4,  4($0)
    . halt
```

3. Use the same function that I use to read inputs in simulator to transfer my iimage.bin and dimage.bin to small-endian.