Computer Architecture Homework 1, Yu Liu, Oct 7th, 2019.

- The files in the package are for all problem 2, problem3 and problem4 in homework1.
- The source codes are under the c language, including head are <stdio.h> , “stdbool.h” and <math.h>, can run under linux , unix, window 7 or above.
- Below process is all run under linux as example
- Any questions call my phone: +1 704 858 7806

Problem2:

Run my composed simulator to analyze trace.din under linux :

1. Put below file in a folder:
   - cachesim_trace.c
   - stdbool.h
   - trace.din

2. Configure the cache simulator parameter:
   Use any text edit (eg. vi) or c language composer (eg. c-free) to edit as below red words in constant declarations area in file of cachesim_trace.c

   //constant declarations
   #define CACHESIZE 32768 //must be 2^n, 65536 for 64K, 16384 for 16K
   #define BLOCKSIZE 8 //must be 2^n, 32 for 32B, 128 for 128B
   #define WAYLEN 1 //direct way is 1, 4 way is 4
   #define RECORDTYPE 's' // u for unified, s for splitted.
   #define FILENAME_INPUT "trace.din"
   #define FILENAME_OUTPUT "result_trace.txt"

3. Compile file of cachesim_trace.c, cachesim_trace.out will be generate by gcc, -lm is for including <math.h>.
   $ gcc cachesim_trace.c –o cachesim_trace.out –lm

4. Run
   ./ cachesim_trace.out

5. Output data , a file named "result_trace.txt” will be generated (a sample already listed), open and read the data, as below example.

```
Cache Size: 32768
Block Size: 8
Way: 1
Data Type: s
Data File: trace.din

              Total       Instruction       Data(Read+Write)          Read         Write
Misses         5959                87                   5872          2105          3767
Hits         826518            597222                 229296        128550        100746
Total        832477            597309                 235168        130655        104513
Miss Rate    0.0072            0.0001                 0.0250        0.0161        0.0360
```

Problem 3:

File of “data_problem3.pdf” contains the raw data of cache L1 to L3 miss/hit rate run under different compiler optimization.

Problem 4:

Run my composed simulator to analyze pin generated data under linux :

Similar process as what in problem2, the source code is changed a little to fit the 64 bits data generated by pin.

1. Put below file in a folder:

- cachesim_pin.c
- stdbool.h
- pinatrace_dhrystone.out
- pinatrace_linpack.out

2. Configure the cache simulator parameter:

- use any text edit (eg. vi) or c language composer (eg. c-free) to edit below red words in constant declarations area in file of cachesim_trace.c. Eg. : change pinatrace_dhrystone.out or pinatrace_dhrysonte.out as a data source selection., change result_drystone.txt or result_dhrystone.txt as a result output change.

```
//constant declarations
#define CACHESIZE_INST 0 //must be 2^n , eg. 0 for 0K, 16384 for 16K
#define CACHESIZE_DATA 65536 //must be 2^n, 65536 for 64K, 16384 for 16K
#define BLOCKSIZE 32          //must be 2^n, 32 for 32B, 128 for 128B
#define WAYLEN 4          //direct way is 1, 4 way is 4
#define RECORDTYPE 'u' //u: unified, s: shared
#define FILENAME_INPUT    "pinatrace_dhrystone.out" // if run dhrystone data, change to
"pinatrace_dhrysonte.out"
#define FILENAME_OUTPUT "result_drystone.txt" // if run dhrystone data, change to
"result_dhrystone.txt"
```

3. Compile file of cachesim_trace.c, cachesim_pin.out will be generated by gcc. –lm is for including <math.h>:
   $ gcc cachesim_pin.c –o cachesim_pin.out –lm

4. Run
   ./ cachesim_pin.out

5. Output data
   A file named “result_dhrystone.txt” or “result_drystone.txt” will be generated automatically (a sample already listed). Open and read the data inside.