CENG331 Final
28 January 2022
Fall 2021- Section 1
Part I (40 minutes – 2 questions)
Name :
ID# : KEY

<Place your ID here>

Important: justify your solutions, unjustified solutions will get zero grade.
[ ] check this box if you were not able to submit your solution via gradescope.com at the end of the session

Question 1:[30pts] Architectural Vulnerability Factor (AVF) quantifies how much a bit-flip matters. AVF for branch predictor is 0% and AVF for program counter is ~100%.

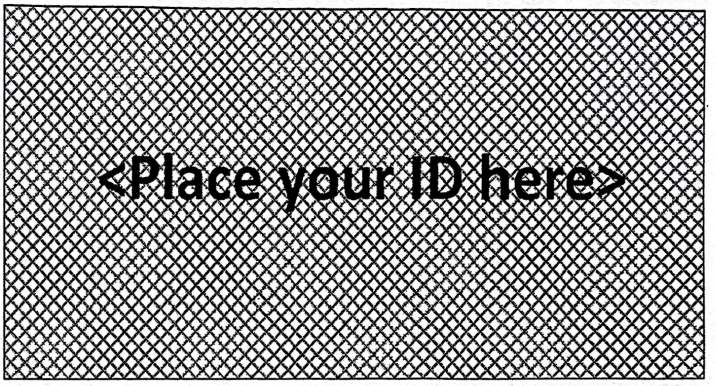
a) Compare the AVFs of %rax and %r9 on an Intel X86-64 architecture.
b) Assume you are designing a new processor and you are allowed to double the number of general-purpose registers. Ignoring the increased chance of a cosmic ray strike, how would average AVF of the register file change?

a) %rax is used for the return value and %r9 is typically used for passing the 6th argument. Therefore it is more likely for a bit-flip to cause a problem for %rax more than %r9. The exact quantification of their architectural vulnerability factors (values) depends on the code, but in general AVF(%rax) > AVF(%r9).

b) For a given code, as more registers are available, average AVF of the register file is expected to decrease. This is due to the fact that some registers would be ~~accessed~~ not used or less frequently used so that bit-flips in them do not matter much.

<turn the page for Question 2>

Question 2:[20pts] Mark the following statements as True (T) or False (F), and fill in the blanks:



a) One should always start with the double nested loops when optimizing a code. [T/F]
b) The problem of adding n numbers is inherently sequential that can not take advantage of multiple arithmetic units in the processor. [T/F]
c) Fully associative cache hardware is more complicated than direct mapped cache [T/F]
d) load/use hazards can not be fixed by forwarding.
e) While base and bound address translation can introduce external fragmentation, paged Address translation does not but it can introduce internal fragmentation, especially for larger page sizes.

<end of part I>

CENG331 Final
28 January 2022
Fall 2021- Section 1
Part II (40 minutes)
Name : KEY
ID# :

<Place your ID here>

[ ] check this box if you were not able to submit your solution via gradescope.com at the end of the session

Question 3 [45pts]: Given an unoptimized C code below:

```
void my_sort(*arr a)
{
 int i,j;
 double tmp1,tmp2;
 for (i=initialize();i<get_arr_len(a)-1;i++)
 {
  for (j=initialize();j<get_arr_len(a)-i-1;j++)
  {
   get_arr_element(a,j,  &tmp1);
   get_arr_element(a,j+1,&tmp2);
   if (tmp1>tmp2) {
      a->data[j]   = tmp2;
      a->data[j+1] = tmp1;}}}}
```

```
typedef struct{
 int len;
 double *data;
} arr;
```

```
int get_arr_len(*arr a)
{
  return(a->len);
}
```

```
int initialize(void)
{
  return 0;
}
```

```
int get_arr_element(*arr a, int idx, double *val)
{
  if (idx >= a->len) return 0;
  *val  = a->data[idx];
  return 1;
}
```

a) Describe all possible optimizations.

* This code sorts an array using bubble sort, the highest level optimization is using a better sorting algorithm such as quicksort, mergesort, etc.

Other than this:
- get rid of "initialize"
- get rid of "get_arr_len"
- " " " "get_arr_element"

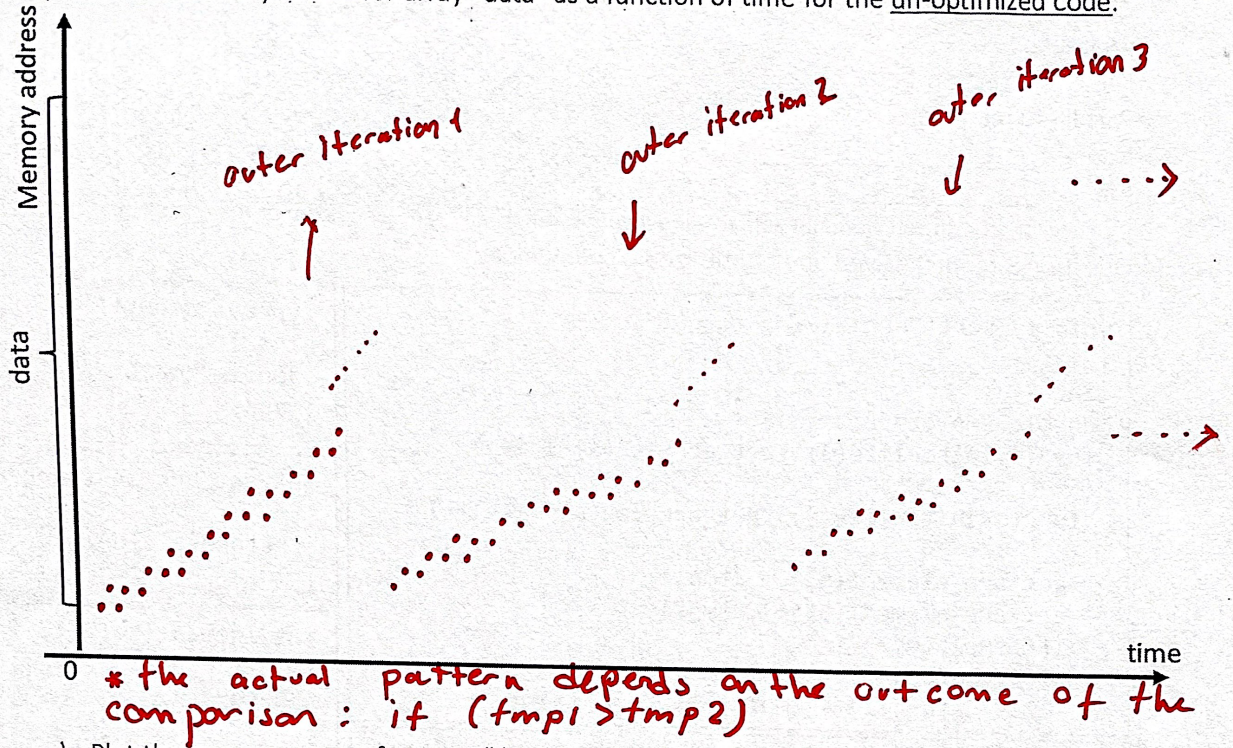
} No need for a procedure call for doing these ops. Since the array length is not ~~expected~~ expected to change during the runtime, there is no need to check array index exceeding the bound ~~every~~ every time accessing an element.

- reuse the array elements from the earlier iterations.
- in general loop unrolling would be good.

<turn the page over for part b and c>

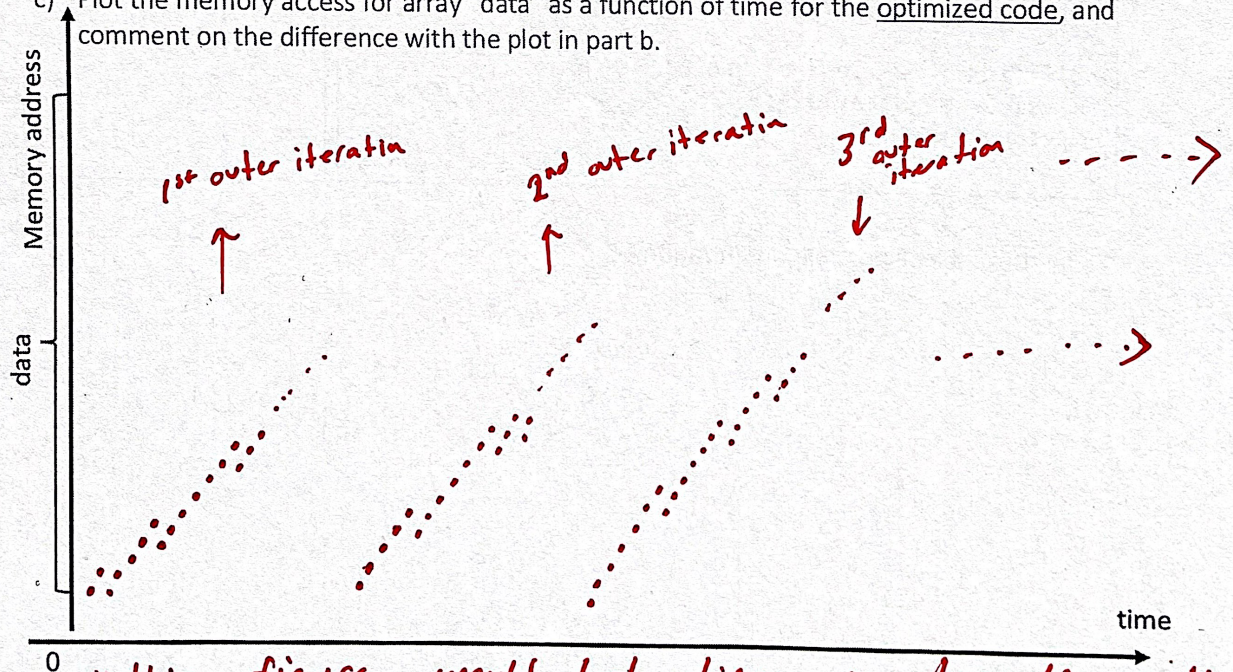
- if the number of registers are limited and that is a priority, number of temporary variables could be reduced.

b) Plot the memory access for array "data" as a function of time for the un-optimized code.



* the actual pattern depends on the outcome of the comparison: if (tmp1 > tmp2)

c) Plot the memory access for array "data" as a function of time for the optimized code, and comment on the difference with the plot in part b.



* this figure would look different depending on the sorting algorithm used, assuming bubble sort with optimization we would get something like the figure but again the details depend on the data (out come of the comparisons).

<End of Part II, do not forget to upload your cheatsheet as Q4>