ENGN2219/COMP6719
Computer Systems & Organization
Problem Set 5

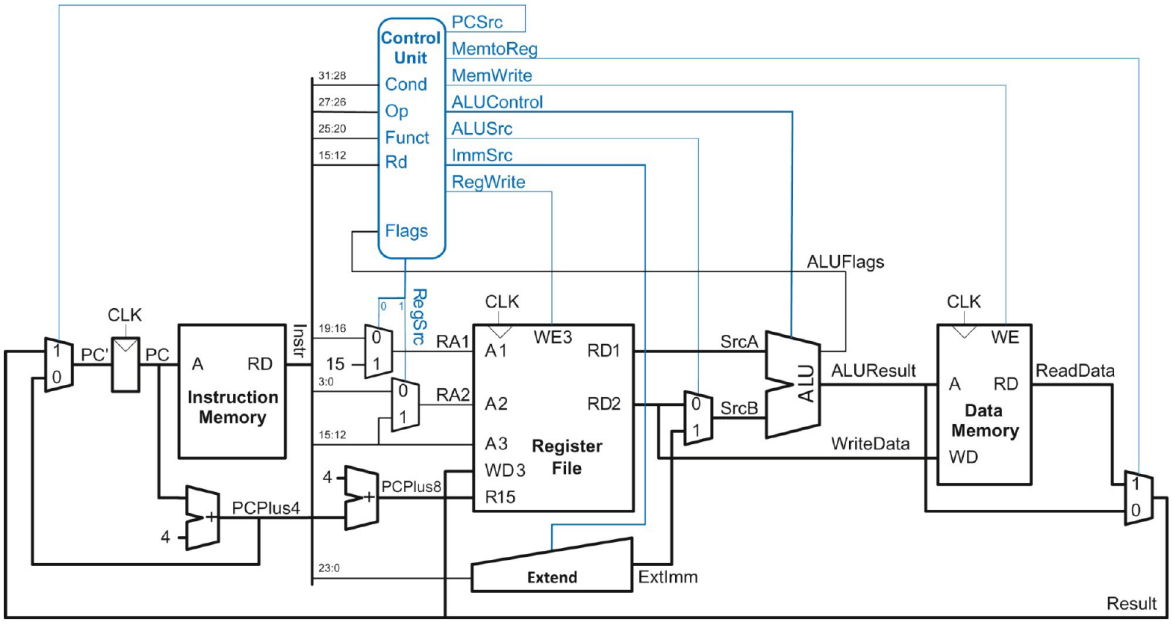
Note: This problem set is optional for your practice only and not part of the assessment scheme.

Question 1:

For this question, we provide the following C code, the ARM CPU microarchitecture, and the delays of logic elements:

```
int array[32];
int k;
array[0] = 2;

for (k = 1; k < 32; k = k + 1)
     array[k] = array[k-1] + 16;
```



t_{pcq_pc} = 35 ps, t_{mem} = 180 ps, t_{dec} = 70 ps, t_{mux} = 25 ps, t_{RFread} = 110 ps, t_{ALU} = 128 ps, and $t_{RFsetup}$ = 66 ps

Part A: Transform the above C code to ARM assembly. Note that there are multiple ways to translate a given piece of C code to assembly. We are only concerned with correctness of your solution and not with code size or performance. A correct translation from C to assembly will receive full marks.

Part B: How long does the following instructions take to execute: (1) ORR, (2) STR, (3) B

Part C: Suppose your assembly code is executed on the single-cycle CPU above. Find the time it takes to execute the assembly code.

Question 2:

Compilers impact the performance of applications in different ways. For a program, compiler X results in an instruction count of 1 billion instructions, and an execution time of one second. A second compiler Y results in an execution time of 1.5 seconds, and an instruction count of 1.2 billion instructions. For a processor with a clock cycle time of one nano seconds, find the average CPI for each of the two programs.

Question 3:

A cache has 64 KB capacity, 128-byte lines, and is 4-way set-associative. The system containing the cache uses 32-bit addresses. (To implement the LRU replacement policy, assume two bits are needed in each tag array entry to hold the age of the line.)

a) How many lines and sets does the cache have?
b) How many entries are required in the tag array?
c) How many bits of tag are required for each entry in the tag array?

Question 4:

An architect is trying to analyze the impact of different replacement policies for a 2-way set-associative cache with four blocks. Specifically, they are considering the following address sequence: 0, 2, 4, 8, 10, 12, 14, 16, 0

a) How many hits does the above sequence exhibit assuming the most recently used (MRU) policy?
b) What about the LRU policy?
c) Simulate a random replacement policy. How many hits does this address sequence exhibit?

Question 5:

For a cache with 128-byte cache lines, give the address of the first word in the line containing the following addresses:

a) 0xa23847ef
b) 0x7245e824

Question 6:

For a direct-mapped cache design with a 32-bit address, the following bits of the address are used to access the cache: Tag (31 – 10), Index (9 – 5), Offset (4 – 0).

Part A: What is the cache block size (in words)?
Part B: How many entries does the cache have?
Part C: What is ratio between the total bits required for such a cache implementation over the data storage bits?

Question 7:

Consider the three functions in the figure below that perform the same operation with varying degrees of spatial locality. Rank-order the functions with respect to the spatial locality enjoyed by each. Explain how you arrived at your ranking.

(a) An array of structs

```
#define N 1000

typedef struct {
    int vel[3];
    int acc[3];
} point;

point p[N];
```

(b) The clear1 function

```
void clear1(point *p, int n)
{
    int i, j;

    for (i = 0; i < n; i++) {
        for (j = 0; j < 3; j++)
            p[i].vel[j] = 0;
        for (j = 0; j < 3; j++)
            p[i].acc[j] = 0;
    }
}
```

(c) The clear2 function

```
void clear2(point *p, int n)
{
    int i, j;

    for (i = 0; i < n; i++) {
        for (j = 0; j < 3; j++) {
            p[i].vel[j] = 0;
            p[i].acc[j] = 0;
        }
    }
}
```

(d) The clear3 function

```
void clear3(point *p, int n)
{
    int i, j;

    for (j = 0; j < 3; j++) {
        for (i = 0; i < n; i++)
            p[i].vel[j] = 0;
        for (i = 0; i < n; i++)
            p[i].acc[j] = 0;
    }
}
```