CS61C Fall 2012 – 13 – Final Review

One, two, three… SIMD! – Spring 2011 Final (Katz, Patterson)

a) SIMDize the following code by filling in the spaces provided. Assume n is a multiple of 4.

```
void count( int n, float *c) {
        for (int i = 0; i < n; i++)
        c[i] = I;
}
```

```
void countfast( int n, float *c) {
        float m[4] = {___, ___, ___, ___};
        __m128 iterate = _mm_loadu_ps(m);
        for (int i = 0; i < ___; i++) {
                _mm_storeu_ps (___, iterate);
                iterate = _mm_add_ps (iterate, _mm_set1_ps(___));
        }
}
```

b) Homer's rule is an efficient way to find the value of polynomial $p(x) = c_0x_{n-1} + c_1x_{n-2} + \ldots + c_{n-2}x + c_{n-1}$:

```
float poly( int n, float *c, float x) {
    float p = 0;
    for ( int i = 0; i < n; i++){
        p = p * x + c[i];
    return p;
}
```

```
float fastpoly( int n, float *c, float x) {
        _m128 p = _mm_setzero_ps();
        for (int i = 0; i < n; i += 4) {
                p = _mm_mul_ps( p, _mm_set1_ps(___));
                p = _mm_add_ps(p, _mm_loadu_ps(___));
        }
        float m[4] = {____, ____, ____, ____};
        p = _mm_mul_ps( p, _mm_loadu_ps(m));
        _mm_storeu_ps(m, p);
        return ___________________;
}
```

Thread Level Parallelism – Summer 2011 Final (Greenbaum)

For the following snippets of code below, **Circle** one of the following to indicate what issue, if any, the code will experience. Assume no thread will complete before another thread starts executing. Assume `arr` is an int array with length `len`.

//Set all elements in arr to 0
int i;
#pragma omp parallel for
for (i = 0; i < len; i++)
    arr[i] = 0;
	Sometimes incorrect	Always incorrect	Slower than serial	Faster than serial

//Set element i of arr to i
#pragma omp parallel
for (int i = 0; i < len; i++)
    arr[i] = i;
	Sometimes incorrect	Always incorrect	Slower than serial	Faster than serial

//Set arr to be an array of Fibonacci numbers.
arr[0] = 0;
arr[1] = 1;
#pragma omp parallel for
for (int i = 2; i < len; i++)
    arr[i] = arr[i - 1] + arr[i - 2];
	Sometimes incorrect	Always incorrect	Slower than serial	Faster than serial

Verilog and Logic – Spring 2004 Final (Garcia)

a) We've seen 6-input, 1-output logic gates like ANDs, ORs, NORs, NANDs, XORs, XNORs, etc. Radio Shack needs one copy of *ALL* the possible 6-input, 1-output logic gates possible and each costs $1. How much does Radio Shack need to spend?

b) Given the following sum-of-products expression for Foo, simplify this expression to a sum of products of (at most) 3 two-variable terms (e.g., AB + CD + AD): What's a good name for Foo?

$$Foo = A*\overline{B}*\overline{C} + \overline{A}*\overline{B}*\overline{C} + \overline{A}*\overline{B}*C + \overline{A}*B*\overline{C}$$

Foo = _________________________, and is better named "______________________________".

c) Given the following simplified sum-of-products expression for Bar, given A, B and C:

$$Bar = A*\overline{B} + A*C + \overline{B}*C$$

Draw the circuit diagram for the Bar function. You are required to instantiate *one 1-bit multiplexor*, and plug C into its select line (label its 0 and 1 inputs). You may only use basic gates AND, OR & NOT. Full credit will only be given to solutions with a mux and *3* basic gates.

Pipelining – Spring 2004 Final (Garcia)

Given the following MIPS code snippet (note that instruction #6 could be anything):

```
loop:
1 addi $t0, $t0, 4
2 lw $v0, 0($t0)
3 sw $v0, 20($t0)
4 lw $s0, 60($t0)
5 bne $s0, $0, loop
6 ## The following instruction could be anything!
```

a) Detect hazards and insert no-ops to insure correct operation. Assume no delayed branch, no forwarding units and no interlocked pipeline stages.

b) Rewrite the program to maximize performance. Assume delayed branch and forwarding units, but no interlocked pipeline stages. For unknown reasons, the first instruction after the loop label must be the addi. Feel free to insert no-ops. You should be able to do it using 6 or only 5 instructions per loop.

```
_________## Extra instructions before the loop if necessary
_________## Extra instructions before the loop if necessary
loop:
1 addi $t0, $t0, 4
2
3
4
5
6
## The following instruction could be anything!
```