Name: _____

Directions: Work on this sheet (**on both sides**, if needed) only; **do not turn in any supplementary sheets of paper**. There is actually plenty of room for your answers, as long as you organize yourself BEFORE starting writing. In order to get full credit, **SHOW YOUR WORK.**

1. Suppose our memory system on p.19 of the printed lecture notes on digital design were to be of size 32x8 instead of 8x4, but still using the same 4x2 chips as building blocks.

(a) (5) How many chips would we need?

(b) (5) How many data lines would we need in the bus?

(c) (10) Fill in the blank with the correct count: Each word of system memory (e.g. each C variable of type **int**) would be stored in _____ chips.

2. Look at the table at the bottom of p.547 in Patterson and Hennessy. Suppose after the access to word 18, we then have accesses to words 19 and 20.

(a) (5) Will the accesses to 19 and 20 produce a hit or miss?

(b) (5) Fill in the blank: The sequence of accesses to 18, 19 and 20 is an example of _____ locality.

3. Look at Figure 4.18 of Patterson and Hennessy.

(a) (5) How many wires are in the line labeled Binvert? How many in Operation?

(b) (5) What is the numerical value on the green Carryin at the very top of the figure? Give a clear explanation for this.

4. (10) On p.12 of the printed lecture notes on storage of program variables, it was stated that most compilers (on 32-bit machines) would leave a gap between the two consecutive **struct**s U and V in the example:

```
struct S {
   int X;
   char A,B;
};
...
...
struct S U,V;
```

(Assume that storage is in "nonreverse order.") Write a *short* (but complete) C program to test this, using the above example. Your program must end its output by reporting either "gap between structs" or "no gap", according to whatever its findings are.

5. We need to build a circuit whose inputs are u, x and y, and whose output is r. If u = 1, then r will be equal to x, and otherwise r will be equal to y.

(a) (5) Write a boolean equation (this term is introduced on p.1 of our digital logic notes) that expresses r in terms of u, x and y.

(b) (5) Show how to use the MUX on p.4 of the notes to produce r from u, x and y. Use the same notation for the MUX inputs (A, D1, D0) and output (Z) as on p.4. We will set r to Z. State what A, D1 and D0 should be set to.

6. (10) Look at the simple CPU on p.21 of the digital logic notes. Though the connections are not shown in the figure, the clock will be connected to one or more of the components shown. Give a complete list of such components.

7. Look at the cache on p.546 of Patterson and Hennessy. Let wa denote the word address of the item we hope to find in the cache, and let ln denote the line number in the cache at which we will look for this item. Instead of using the mapping

```
ln = wa mod 8
```

as our index into the cache, in this problem we will consider alternative mappings, of the type

```
ln = (k wa) mod 8
```

(The original design on p.546 has k = 1.) To make matters simple, we will use all 5 bits of wa as the tag (even though this may be wasteful).

(a) (10) Why would it be very undesirable to use k = 4? Be very specific and clear in your answer.

(b) (10) Give a value for k, other than 1, which would not have the disadvantage of k = 4.

8. (10) Consider the cache on p.557 of Patterson and Hennessy. Suppose we have a program which includes a declaration

```
int x[1000];
```

and suppose the address of x[0] is 32. If an access to x[50] causes a cache miss, which other elements of x—if any—will be brought into the cache besides x[50]?

Solutions:

1.a. 32/4 x 8/2 = 32.

1.b. 8.

1.c. 4.

2.a. Miss, miss.

2.b. Spatial.

3.a. 1,2.

3.b. It's 0 unless we want to computer a-b, in which case it's 1; see Patterson and Hennessy.

4.

```
struct S {
   int X;  char A,B;
}

struct S X,Y;

main()

{  if ((int) &V - (int) &U > 6) printf("gap\n");
   else printf("no gap\n");
}
```

5.a. r = u x + $\overline{u}$ y.

5.b. A = u, D1 = x, D0 = y.

6. R, N/Z, IR, PC, memory.

7.a. We would not use all of the cache.

7.b. Any number relatively prime to 8. (Credit not allowed unless get part (a) correct.)

8. x[50] will be 50 x 4 = 200 bytes from x[0], thus at 232. Block size is 16, so x[50] will start in the 232 mod 16 = 8th byte of its block (counting the beginning byte of the block as the 0th). Since each x[i] is 4 bytes, that means x[48] is at byte 0 of the block, and so on, with x[51] being the last element of x in this block.