Noah Hanks

# ECEN 424 HW 1

2.61

```c
int any_bit_one(int x) { return !!x; }

int any_bit_zero(int x) { return !!(~x); }

int lsb_one(int x) { return !!(x & 0xFF); }

int msb_zero(int x) {
    int shiftValue = (sizeof(int) - 1) << 3; // Find the size of int for shifting.
    int xShifted = x >> shiftValue;          // Shift right to leave just msb.
    int msb = xShifted & 0xFF;               // Mask with FF to expose 1's
    return !!(msb ^ 0xFF);
}
```

2.71

A) The wrong code is not able to perform sign extension.

B)

```c
int myxbyte(packed_t word, int bytenum) { return (((int)word << ((3 - bytenum) << 3)) >> 24); }
```

2.72

A) Since on of the values in the conditional statement is unsigned, it will compare both numbers as if they were both unsigned making it so that is always comes out to true.

B)

```c
void copy_int(int val, void *buf, int maxbytes) {
    // if (maxbytes - sizeof(val) >= 0) // Previous conditional statement
    if (maxbytes >= sizeof(val))
        memcpy(buf, (void *)&val, sizeof(val));
}
```

2.77

```c
/* Part A */
K = 17;
result1 = (x << 4) + x; /* replace this 0 with your C expression */
result2 = K * x;
printf("K=17: +-<< result = %d, * result = %d\n", result1, result2);

/* Part B */
K = -7;
result1 = x - (x << 3); /* replace this 0 with your C expression */
result2 = K * x;
printf("K=-7: +-<< result = %d, * result = %d\n", result1, result2);

/* Part C */
K = 60;
result1 = (x << 6) - (x << 2); /* replace this 0 with your C expression */
result2 = K * x;
printf("K=60: +-<< result = %d, * result = %d\n", result1, result2);

/* Part D */
K = -112;
result1 = (x << 4) - (x << 7); /* replace this 0 with your C expression */
result2 = K * x;
printf("K=-112: +-<< result = %d, * result = %d\n", result1, result2);
```

2.8

A) Not true when x is INT_MIN and y is 0.
B) Always true because the shifting and multiplying yields the same result.
C) Not true when x and y are both 0.
D) Always true because of the casting of unsigned values to int.
E) Always true because the values would shift towards INT_MIN.

424-1:

A) The statement shouldn't terminate.
B) Strangely, the statement does end.
C) It runs up to 16777216. I would think that the float value doesn't have enough bits to represent the next value.
D) Doubles would take a lot longer since they can store more bits.
E) If the variables were of a different type, they might wrap around and not stop, end shorter, or end longer.