Computer Architecture Project 1 Part A

1. The function ques1 declares a 2’s complement number and the corresponding value is assigned to variable y. Variable z is given the value of NOT (x OR y), and is then shifted 31 bits right. The function returns this value of z AND 1. More simplified, the function returns 0 AND 1, or just a value of 0.
2. The function ques2 declares a variable “mask” which is shifted 31 bits right. Variable y gets the value (x XOR mask). Finally, variable z represents the 2’s complement of NOT mask and 1. The sum of (y+z) is returned, which can be simply written as x|0 depending on the value of variable x passed into the function.
3. Two variables are declared to represent the negation of x and y. Variable f takes the OR of these two variables, and the overall negation of f is returned, which is just returning the OR of the two variables x and y being passed into the function.
4. The sum of variables x and y is taken. The fields x\_neg, s\_neg, and y\_neg are all assigned the shift 31 bits right from the original variables x and y and the localized variable sum. The XOR of x\_neg and s\_neg AND the XOR of x\_neg and y\_neg are flipped using the logical not operation (!), and the resulting value is returned, which can be done more easily by returning the result that represents the sum of 0 and 1.
5. The variable result is left shifted 31 bits before being right shifted the same number of bits before being returned. The end result is the variable x being passed in.
6. “Byte” is given the value 0x55 which is 55 in hexadecimal. Variable word represents the value of byte (55) OR the value of byte left shifted 8 bits (5,500). The value of “word” OR word left shifted 16 bits is returned.
7. The variable being passed in, x AND the 2’s complement of the same variable is returned, which is already in its simplest form.
8. Variable y is given the 2’s complement representation of x. After the computation of x OR y is taken, it shifted to the right 31 bits and assigned to z. The value returned is z AND 1 which is just x divided by itself.
9. The variable n8 represents the value n being bit shifted to the left three places. “Mask” then shifts 255 (0xff in hexadecimal) these many places to the left. Another variable, cshift, shifts c, a variable passed in the parameter of the function, the same number of places left as mask did. Z represents the AND of the third variable passed in, x, and the negation of mask [~]. The OR of z and cshift is returned.
10. An integer byte is assigned the value of 170 (0xAA in hexadecimal). The variable word of type int as well is the OR of 170 (byte) and this value shifted left 8 bits. The OR of word is returned and the representation of word bit shifted to the left 16 bits.
11. Variable m8 is assigned the same value of 170 as in question 10. Variable m89 is then shifted left 8 bits (making the value 43,520). M16 represents the OR of m8 (170) or m8 shifted to the left 8 bits. Going further, m32 follows the same pattern of taking the OR of the previous variable and bit shifting it to the left twice as many places, so in this case, m16 OR m16 shifted left 16 bits, which results in the value 2,863,311,530. “Fillx”, represents x which is passed in OR m32. The bits of fillx are flipped, and the logical negation of this is returned.
12. An int declared a represents the 2’s complement of m. Another new localized variable b is assigned the 2’s complement of x. A is then re-assigned the sum of itself plus x, and b is re-assigned the sum of itself plus n. After the OR of these new values of a and b is taken, the value is shifted to the right 31 bits, and the logical not is returned. Since variable n is unused, the function really returns the OR of the 2’s complement numbers of x and m, right shifted 31 bits.
13. Five localized variables are declared, mask1, mask2, mask4, mask8, and mask16. Mask2 is given the sum of 51 (0x33 in hexadecimal) and 51 shifted to the left 8 bits. Mask2 is then shifted to the left 16 bits and that value is added to the previous value (when shifted left 8 bits). Mask4 mimics this, except with the decimal 15 (0x0F in hexadecimal). Mask8 and mask16 are calculated with some variations from mask2 and mask4. Both masks handle the decimal 255 (0xFF in hexadecimal), which is added to the result of 255 shifted left 8 bits (mask16), or the result of 255 shifted left 16 bits (mask8). The value x which was passed in the function ques13, is given the AND of itself and each mask declared. This value is added to x being shifted to the right the same number of bits as the number on the variable mask (mask 1 shifted right 1 bit, mask2 shifted right 2 bits, etc.) and the AND of this value and the mask represents the right portion of the sum. After all of these sums, the variable x which was passed in is returned.
14. An iterator i is used in a for loop which loops through a 32 bit number. After each iteration, the variable result is given the XOR of x shifted right the iterator i bits to the right and then given the AND with 1. After the loop is complete, the variable result is returned.
15. The variable n passed in to function ques15 is a positive number. A variable temp is given 1 shifted to the left by n bits. Z represents the sum of temp and NOT 0 (which is 1). Finally, the AND of x and z is returned.