CSPB 2400 - Park - Computer Systems

Started on	Friday, 19 January 2024, 6:53 PM
State	Finished
Completed on	Friday, 19 January 2024, 7:21 PM
Time taken	28 mins 4 secs
Marks	38.00/38.00
Grade	**10.00** out of 10.00 (**100**%)

Question **1**

Correct

Mark 4.00 out of 4.00

Add the two following binary numbers:

	1	0	1	0	0
+	0	1	1	0	0
	0	0	0	0	0

Question **2**

Correct

Mark 4.00 out of 4.00

Add the two following binary numbers:

	1	0	0	0
+	0	0	1	1
	1	0	1	1

Question **3**

Correct

Mark 4.00 out of 4.00

Subtract the second binary number from the first:

	1	1	0	1
-	1	0	1	0
	0 ✔	0 ✔	1 ✔	1 ✔

Question **4**

Correct

Mark 4.00 out of 4.00

Compute the sum of these two unsigned 4-bit numbers as a 4-bit result. We'll show you the numbers in decimal and binary, but you should enter your answer in decimal.

$$\begin{array}{rrr} & 5 & 0101_2 \\ + & 13 & 1101_2 \\ \hline \end{array}$$

You may enter an expression if you like.

2

Your last answer was interpreted as follows: 2

Because the values are unsigned and 4 bits in length, the answer is 5 + 13 mod 16, or 2.

Correct answer, well done.

Question **5**

Correct

Mark 4.00 out of 4.00

Compute the sum of these two signed 4-bit numbers as a 4-bit result. We'll show you the numbers in decimal and binary, but you should enter your answer in decimal.

$$\begin{array}{rll} & 7 & 0111_2 \\ + & 6 & 0110_2 \\ \hline \end{array}$$

You may enter an expression if you like.

-3

Your last answer was interpreted as follows: -3

Correct answer, well done.

The correct response is -3.

Signed and unsigned addition perform the same bit-level operations.

If we solved this problem as an unsigned 4 bits problem, the answer would be $7 + 6 \bmod 16$, or 13 (1101) when interpreted as unsigned values. This same bitstring would be interpreted as $5 + -8$ or -3 in two's complement signed representation.

Question **6**

Correct

Mark 4.00 out of 4.00

Compute the sum of these two signed 5-bit numbers as a 5-bit result. We'll show you the numbers in decimal and binary, but you should enter your answer in decimal.

$$\begin{array}{rll} & 9 & 01001_2 \\ + & 7 & 00111_2 \\ \hline \end{array}$$

You may enter an expression if you like.

-16

Your last answer was interpreted as follows: -16

Correct answer, well done.

The correct response is -16.

Signed and unsigned addition perform the same bit-level operations.

If we solved this problem as an unsigned 5 bits problem, the answer would be $9 + 7 \bmod 32$, or 16 (10000) when interpreted as unsigned values. This same bitstring would be interpreted as $0 + -16$ or -16 in two's complement signed representation.

Question **7**

Correct

Mark 1.00 out of 1.00

Compute the sum of these two signed 5-bit numbers as a 5-bit result. We'll show you the numbers in decimal and binary, but you should enter your answer in decimal.

$$\begin{array}{rll} & -15 & 10001_2 \\ + & -15 & 10001_2 \\ \hline \end{array}$$

You may enter an expression if you like.

2

Your last answer was interpreted as follows: 2

Correct answer, well done.

The correct response is 2.

Signed and unsigned addition perform the same bit-level operations.

If we solved this problem as an unsigned 5 bits problem, the answer would be $17 + 17 \bmod 32$, or 2 (00010) when interpreted as unsigned values. This same bitstring would be interpreted as $2 + 0$ or 2 in two's complement signed representation.

Question **8**

Correct

Mark 1.00 out of 1.00

Write a C function **overflow** that returns true if the sum of the two unsigned long arguments would overflow when using unsigned 64-bit arithmetic.

For example:

Test	Result
`if (overflow(ULONG_MAX, 0) == solution(ULONG_MAX,0)) printf("OK1!\n");`	OK1!

Answer: (penalty regime: 10, 20, ... %)

Reset answer

```
int overflow(unsigned long a, unsigned long b)
{
  /* your code here*/
  unsigned long max = 18446744073709551615UL;
  return (a > (max - b));
}
```

	Test	Expected	Got	
✔	`if (overflow(ULONG_MAX, 0) == solution(ULONG_MAX,0)) printf("OK1!\n");`	OK1!	OK1!	✔
✔	`if (overflow(ULONG_MAX, 1) == solution(ULONG_MAX,1)) printf("OK2!\n");`	OK2!	OK2!	✔
✔	`if (overflow(ULONG_MAX/2, ULONG_MAX/2) == solution(ULONG_MAX/2,ULONG_MAX/2)) printf("OK3!\n");`	OK3!	OK3!	✔
✔	`if (overflow(LONG_MIN, LONG_MIN) == solution(LONG_MIN,LONG_MIN)) printf("OK4!\n");`	OK4!	OK4!	✔

Passed all tests! ✔

Correct

Marks for this submission: 1.00/1.00.

Question **9**

Correct

Mark 4.00 out of 4.00

Compute the sum of these two unsigned 6-bit numbers as a 6-bit result. We'll show you the numbers in decimal and binary, but you should enter your answer in decimal.

$$\begin{array}{rll} & 62 & 111110_2 \\ + & 39 & 100111_2 \\ \hline \end{array}$$

You may enter an expression if you like.

37

Your last answer was interpreted as follows: 37

Because the values are unsigned and 6 bits in length, the answer is $62 + 39 \bmod 64$, or 37.

Correct answer, well done.

Question **10**

Correct

Mark 4.00 out of 4.00

Compute the sum of these two signed 6-bit numbers as a 6-bit result. We'll show you the numbers in decimal and binary, but you should enter your answer in decimal.

$$\begin{array}{rll} & -28 & 100100_2 \\ + & -25 & 100111_2 \\ \hline \end{array}$$

You may enter an expression if you like.

11

Your last answer was interpreted as follows: 11

Correct answer, well done.

The correct response is 11.

Signed and unsigned addition perform the same bit-level operations.

If we solved this problem as an unsigned 6 bits problem, the answer would be $36 + 39 \bmod 64$, or 11 (001011) when interpreted as unsigned values. This same bitstring would be interpreted as $11 + 0$ or 11 in two's complement signed representation.

Question **11**

Correct

Mark 2.00 out of 2.00

Multiplying a 2's complement number by 8 is the same as:

Select one:

- a. Shifting the value left by 3 bit positions ✔
- b. Shifting the value left by 1 bit positions
- c. Shifting the value right by 2 bit positions
- d. Shifting the value left by 2 bit positions
- e. Shifting the value right by 3 bit positions

Your answer is correct.

Multiplying by 8 is the same as shifting the value left by three bit positions.

Question **12**

Correct

Mark 1.00 out of 1.00

Dividing an unsigned number by 4 is the same as:

Select one:

- a. Shifting the value right by 2 bit positions using a logical shift. ✔
- b. Shifting the value left by 3 bit positions
- c. Shifting the value right by 2 bit positions using an arithmetic shift.
- d. Shifting the value right by 3 bit positions using an arithmetic shift.
- e. Shifting the value left by 2 bit positions using a logical shift.

Your answer is correct.

Dividing an unsigned number by 4 is the same as shifting the value right by two bit positions using a logical shift.

Question **13**

Correct

Mark 1.00 out of 1.00

In the following code, we have omitted the definitions of constants M and N:

```
#define M
#define N
int arith(int x, int y) {
   int result = 0;
  result = x*M + y/N;
  return result;
}
```

We compiled this code for particular values of M and N. The compiler optimized the multiplication and division using the methods discussed in Chaper 2.3. The following is a translation of the generated machine code back into C:

```
int optarith(int x, int y) {
    int t = x;
    x <<= 6;
    x -= t*3;
   if (y < 0) y += 7;
    y >>= 3;
    return x+y;
}
```

What is the values of M? 61

Your last answer was interpreted as follows: 61

Correct answer, well done.

What is the values of N? 8

Your last answer was interpreted as follows: 8

Correct answer, well done.

For M, we are left shifting by 6 bits, effectively multiplying by 64. We are then subtracting 3 multiples of the original value, yielding 61.

For N, we are dividing by 8 using an arithmetic left shift of 3 bits. We need to correct for rounding of negative numbers by adding in 7 before doing the shift.