#46

EECS 281: **Test #2 (3 pages)** **Due: Thursday, March 3, 2005**

Name: Solutions Email: ______ Grade ______ (100 max)

1. (25%) Assume char is **6-bits** and short is **9-bits**, convert the following into formats described by each column.

	8051 assembler	big endian two's compl.	big endian one's compl.	big endian signed magnitude
unsigned short range:		0~511	0~511	
signed char range:		-32~31	-31~31	-31~31
unsigned char x=042;	.byte 42o	100010	100010	out of range
unsigned char x=42;	.byte 42	101010	101010	out of range
unsigned char x=0x42;	.byte 42h .byte 0x42	Out of range		
signed short x=−16;	.word -16	111110000	111101111	100010000

2. (5%) For column 2, disassemble the following 8-bit binary number into **assembler** and for columns 3 to 6 convert into **decimal** under the different interpretations. (8051 Hint: MOV A,Rn; ADD A,Rn; ADDC A,Rn or ANL A,Rn).

	8051 instruction	signed big endian two's compl.	unsigned big endian two's compl.	signed big endian one's compl.	big endian signed magnitude
00111101	ADDC A,R5	61	61	61	61

3. (5%) Using C++ operator precedence, **add** the correct parenthesis:

```
w = (a << (b + (c * d)));
w += ((a & b) | (c % d));
```

4. (20%) Using standard C++ data types (i.e. 8-bit char) convert the following using the following values:

register unsigned char u=0xff, f=0x96, g=0x33; register signed char s=1, t=0x46, x=0xc3;

Assign the 8051 registers as follows: s=R0, t=R1, x=R2, u=R3, f=R4, g=R5; State if **overflow or carry** has occurred after execution. 8051 MUL, DIV and SUBB instructions are **not** allowed.

	two's complement big endian	8051 instructions	OV	C
u = f ^ g;	1010 0101	Mov A,R4 ; XRL A,R5; Mov R3,A	–	–
u = g & 0x0f;	0000 0011	Mov A,R5; ANL A,#0x0F; Mov R3,A	–	–
u = g * 4;	1100 1100	Mov A,R5; ~~RLC A~~ CLR C; RLC A CLR C; RLC A ; Mov R3,A	0	0
g--;	0011 0010	dec R5	0	0
s = t – x;	1000 0011	Mov A,R2 ; CPL A ; Inc A ADD A,R1 ; Mov R0,A	1	0
s += x;	1100 0100	Mov A,R2 ; ADD A,R0 ; Mov R0,A	0	0
s = t | x;	1100 0111	Mov A,R1; ORL A,R2 ; Mov R0,A	–	–

$s = t + (\sim x + 1)$

$OV = C_{out} \oplus C_{in} = 1 \oplus 0 = 1$

5. (15%) Using the 8051 instructions, assemble the instruction into hex, and then execute it showing all changing values:

Mem. Addr.	Machine instructions	Assembly	PC	OV	CY	AC	Reg. A	Reg. R5
0x0	1110 0100 E 4	CLR A	0x0	1	1	1	0x42	0x13
0x1	1011 0011 B 3	CPL C	0x1	1	1	1	0x00	0x13
0x2	0011 1101 3 D	ADDC A,R5	0x2	1	0	1	0x00	0x13
0x3	1111 0100 F 4	CPL A	0x3	0	0	0	0x13	0x13
0x4	1111 1101 F D	MOV R5,A	0x4	0	0	0	0xEC	0x13
		Final Values=	0x5	0	0	0	0xEC	0xEC

$x = 0xC3 = 1100\ 0011\ b$

$\sim x = 0x3C = 0011\ 1100\ b$

$\sim x+1 = 0x3D = 0011\ 1101\ b$

$C_{in} = 1$

$t = 0x46 = 0100\ 0110$

$-x = 0x3D = 0011\ 1101$

$1000\ 0011$

$C_{out} = 0$

6. (5%) Assuming standard C++ data types, write a "single" C++ code statement of clearing bit d_6 to zero, setting bit d_3 to one, complement bit d_1 (i.e. invert the bit d_1) and leaving all other bits unchanged in the variable char x.

x & 1011 1111
B F

x | 0000 1000
0 8

x ^ 0000 0010
0 2

=> $\boxed{\text{x = (((x \& 0xBF) | 0x08) \^{} 0x02;}}$

7. (5%) Write the 8051 instructions for problem 6 and assume x is register R3.

```
MOV A,R3
ANL A,#0xBF
ORL A,#0x08
XRL A,#0x02
MOV R3,A
```

8. (5%) Write the 8051 assembler for char w[8] = "-128";

.byte "-128",0,0,0,0 need 8 characters

end of string zero

9. (10%) Rewrite the following C statement **without** using multiply, divide or modulo: u = x*13 + y/4 + z%8;

$x \cdot 13$
$\equiv x \cdot (8+4+1)$
$\equiv x \cdot 8 + x \cdot 4 + x$
$\equiv x \cdot 2^3 + x \cdot 2^2 + x$
$\equiv x<<3 + x<<2 + x$

$u = ((x<<3) + (x<<2) + x) + y>>2 + \underbrace{(z \,\&\, 0111b)}_{z \,\&\, 0x07}$

$= y/4$
$= y \cdot 2^{-2}$
$= y>>2$

9. (5%) Why is two's complement representation preferred over one's complement?

Two's complement has only one representation of zero and has one greater decimal value range.

x1 (5% extra credit). What is the advantage of binary representation over ASCII representation of numbers? (i.e. 128 decimal number = 10000000 binary = "128" ASCII);

Binary representation uses less memory to store the same numbers.

$128_{10} \Rightarrow \underbrace{10000000}_{\text{8 bits binary}} \Rightarrow \underbrace{\text{"128"}}_{\text{24 bits ASCII}}$

x2 (5% extra credit). Convert the 24-bit number 0x031879 to mime base64: Axh5

000000	110001	100001	111001
0	49	33	57
'A'	'x'	'h'	'5'