GUIDE DE L'UTILISATEUR

Fonctions

( )	Function call	jump(TestFunt) call(TestFunct)
[ ]	Array element reference	
.	Structure member reference	AutoPilot.althold switch althold on device AutoPilot AltitudePH.altitude element altitude of pigeon hole AltitudePH

	Increment	operand++;
		The value of the operand is incremented by 1.
		testByte++; if testByte was 3, will equal 4 after execution
		same as
		testByte = testByte + 1;
		and
		testByte += 1;

--	Decrement	operand--; The value of the operand is decremented by 1. TestByte--; if testByte was 4, will equal 3 after execution Same as testByte = testByte – 1; and testByte -= 1;
!	Logical negation	!operand Returns the logical NOT operation on the operand. A true operand returns false, a false operand returns true. If(!testFlag) Proc1 else Proc2 will execute Proc1 if testFlag is "false"
~	Ones compliment	~operand enclose in parenthesis Returns the compliment of the operand. The returned value is the operand with its bits reversed (1's become 0's, 0's become 1's). testByte = (~testByte); if testByte = 0x3B, will equal 0xC2 after execution 00111101 (0x3B) 11000010 (0xC2)
&	AND	expression1 & expression2 Returns a bitwise AND operation done on expression1 and expression2. testByte = testByte1 & 0xF0; if testByte1 was 0x3B, testByte will equal 0x30 after execution. 1111000 (0xF0) 00111011 (0x3B) 00110000 (0x30)

*	Multiplication	expression1 * expression2
		The result of this is the value of expression1 multiplied by expression2.
		testByte1 = 10;
		testByte2 = 25;
		testByte = testByte1 * testByte2; testByte will equal 250 after execution

^	Exclusive or	Expression1 ^ expression2 If 2 bits in a column are the same the result is 0, otherwise 1 00001010 (0x0A) 10000011 (0x83) 10001001 (0x89)

Fonctions (suite)

/	Division	expression1 / expression2
		The result of this is the value of expression1 divided by expression2.
		testByte1 = 125;
		testByte2 = 10;
		testByte = testByte1 / testByte2; testByte will equal 12 after execution. Integer math, you will lose the remainder.

%	Modulus	expression1 % expression2
		The result of this is the value of the remainder after dividing expression1 by expression2.
		testWord = testWord % 256; //this will aways return a number from 0 to 255
		if testWord was 513 before this line testWord will equal 1 after execution.
		testWord = 10510 % 25; testWord will equal 10 after this line

+	Addition	expression1 + expression2 The result of this is the sum of the two expressions. If testWord1 = 500 and testWord2 = 35 testWord3 = testWord1 + testWord2; testWord3 will equal 535 after execution
-	Subtraction	expression1 - expression2 The result of this is the value of expression2 subtracted from expression1. If testWord1 = 500 and testWord2 = 35 testWord3 = testWord1 – testWord2; testWord3 will equal 465 after execution.
|	Bitwise OR	expression1 | expression2 Returns a bitwise OR operation done on expression1 and expression2. If testWord1 = 0x0801 and testWord2=0x0080 testWord3 = testWord1 | testWord2; testWord3 will equal 0x881 after execution 0000100000000001 (0x0801) 0000000010000000 (0x0080) 0000100010000001 (0x0881) if testWord1 = 0xFF and testword2 = 0x81 testWord3 will equal 0xFF after execution. 10000001 (0x81) 11111111 (0xFF) 11111111 (0xFF)
@	Address of operator	word_ptr = @myWord; set word_ptr to the address of myWord EPICenter version 1.0.7.6 and above

Fonctions (suite)

<<	Left shift	*expression1* **<<** *shift_value* Returns *expression1* with its bits shifted to the left by the *shift_value*. The rightmost bits are replaced with zeros. This result is the value of *expression1* multiplied by the value of 2 raised to the power of *shift_value*. If *expression1* is signed, then the result is implementation specific. testWord1 = 0x0035; (0b0000000000110101) testWord1 = testWord1 << 2; testWord1 will equal 0x00D4(0b0000000011010100) after execution.
>>	Right shift	expression1 >> shift_value Returns expression1 with its bits shifted to the right by the shift_value. The leftmost bits are replaced with zeros if the value is nonnegative or unsigned. This result is the integer part of expression1 divided by 2 raised to the power of shift_value. If expression1 is signed, then the result is implementation specific. testWord1 = testWord2 >> 3; if testWord2 was 0x0100 (0b0000000100000000) it will equal 0x0020 (0b0000000000100000) after execution
=	Assignment operators	expression1 = expression2 The value of expression2 is stored in expression1.

+=	Add and assign	expression1 += expression2
		The value of expression1 plus expression2 is stored in expression1.
		testWord += 5;
		Same as
		testWord = testWord + 5;

-=	Subtract and assign	expression1 -= expression2 The value of expression1 minus expression2 is stored in expression1. testWord -= 5; Same as testWord = testWord – 5;

***=**	Multiply and assign	expression1 *= expression2
		The value of expression1 times expression2 is stored in expression1.
		testWord *= 5;
		same as
		testWord = testWord * 5;

/=	Divide and assign	expression1 /= expression2
		The value of expression1 divided by expression2 is stored in expression1.
		testWord /= 5;
		same as
		testWord = testWord / 5;

%=	Modulus and assign	expression1 %= expression2
		The value of the remainder of expression1 divided by expression2 is stored in expression1.
		testWord %= 256;
		same as
		testWord = testWord % 256;

&=	AND (mask) and assign	expression1&= expression2
		The value of expression1 AND(mask) expression2 is stored in expression1.
		testWord &= 0x25;
		same as
		testWord = testWord & 0x25;

^=	Exclusive OR and assign	expression1 ^= expression2
		The value of the bitwise XOR of expression1 and expression2 is stored in expression1.
		testWord ^= 0x25;
		same as
		testWord = testWord ^ 0x25;

|=	Or and assign	expression1 |= expression2
		The value of the bitwise OR of expression1 and expression2 is stored in expression1.
		testWord |= 0x25;
		same as
		testWord = testWord | 0x25;

<<=	Shift left and assign	expression1 <<= shift_value
		The value of expression1's bits are shifted to the left by shift_value and stored in expression1.
		testWord <<= 2;
		same as
		testWord = testWord << 2;

>>=	Shift right and assign	expression1 >>= shift_value
		The value of expression1's bits are shifted to the right by shift_value and stored in expression1.
		testWord >>= 2;
		same as
		testWord = testWord >> 2;

Opérateurs de comparaisons

<	Less than	expression1 < expression2
		Returns 1 (true) if expression1 is less than expression2, otherwise the result is 0 (false).
		testWord1 = 100;
		testWord2 = 200;
		if(testWord1 < testWord2) call xxxx; else call yyyy;
		will execute xxxx since 100 is less than 200

>	Greater than	expression1 > expression2
		Returns 1 (true) if expression1 is greater than expression2, otherwise the result is 0 (false).
		testWord1 = 100;
		testWord2 = 200;
		If(testWord1 > testWord2) call xxxx; else call yyyy;
		will execute yyyy since 100 is not greater than 200

>=	Greater than or equal to	expression1 >= expression2
		Returns 1 (true) if expression1 is greater than or equal to expression2, otherwise the result is 0 (false).
		testWord1 = 100;
		testWord2 = 200;
		If(testWord1 >= testWord2) call xxxx; else call yyyy;
		will execute yyyy since 100 is not greater than 200
		if testWord2 = 100
		will execute xxxx since 100 equals 100

<=	Less than or equal to	expression1 <= expression2
		Returns 1 (true) if expression1 is less than or equal to expression2, otherwise the result is 0 (false).
		testWord1 = 100;
		testWord2 = 200;
		if(testWord1 <= testWord2) call xxxx; else call yyyy;
		will execute xxxx since 100 is less than 200
		if testWord2 = 100 would also execute xxxx since 100 equals 100

==	Equality	expression1 == expression2
		Returns 1 (true) if expression1 is equal to expression2, otherwise the result is 0 (false).
		testWord1 = 100;
		testWord2 = 200;
		if(testWord1 == testWord2) call xxxx; else call yyyy;
		will execute yyyy since 100 is NOT equal to 200

!=	Inequality	expression1 != expression2
		Returns 1 (true) if expression1 is not equal to expression2, otherwise the result is 0 (false).
		testWord1 = 100;
		testWord2 = 200;
		if(testWord1 != testWord2) call xxxx; else call yyyy;
		will execute xxxx since 100 is NOT equal to 200

||	Logical OR	Expression1 || expression2 Returns 1 (true) if expression1 is true(not zero) OR expression2 is true (not zero) TestWord1 = 0; TestWord2 = 5; If(TestWord1 || TestWord2) call xxxx; else call yyyy; Will execute xxxx since TestWord2 evaluates to true(1);
&&	Logical AND	Expression1 && expression2 Returns 1 (true) if expression1 is true(not zero) AND expression2 is true (not zero) TestWord1 = 0; TestWord2 = 5; If(TestWord1 && TestWord2) call xxxx; else call yyyy; Will execute yyyy since TestWord1 evaluates to false(0).