

Homework 5 – due Mon. 6 April

(1) We would like to implement 64-bit integers, by using arrays of 4 regular integers (a regular integer has 2 bytes). We use **indexed addressing** for accessing arrays.

(1.a) Write a Pep8 assembly program that adds two 64-bit integers (it doesn't matter whether they are unsigned or in two's complement). The two 8-byte arrays are assumed to be already in memory starting at address x , respectively y . The sum should be written into an array in memory starting at address s .

Algorithm:

```
allocate 8 bytes to s, initialized to zero
carry = 0
for i in [0, 3]:
    s[i] = carry
    s[i] += x[i]
    if C: carry = 1      # C is one of the status flags
    s[i] += y[i]
    if C: carry = 1
```

(1.b) Write a Pep8 assembly program that does two's complement sign-change. Assume an array representing a number in 64-bit two's complement is already in memory, starting at address x . The result after sign-change should be written into an 8-byte array in memory starting at address s .

Algorithm:

```
allocate 8 bytes to s, each initialized to 0x00
carry = 1
for i in [0, 3]:
    s[i] = NOT x[i]      # bitwise complement
    s[i] += carry
    carry = C
```

Explain how your programs work (use a flow-chart).

Test your programs using the Pep8 simulator; save your program in a *file* while you debug. (The Pep8 simulator lets you open a Pep8 program file.)

(2) We want to check whether a string w of characters from $'(, '[, ')', ']'$ is a *well-formed parenthesis expression*. E.g., $([](([]))$ is well-formed, and $([])$ isn't. The output should be the string **True** or the string **False**.

We use a stack, and the following algorithm.

```

s = Stack()
for p in w:
    if p == '(' or p == '[' : s.push(p)
    if p == ')' or p == ']' :
        if s.empty() : goto false
        else:
            top = s.pop()
            if not( ( top == '(' and p == ')' ) or
                    ( top == '[' and p == ']' ) ): goto false
    if s.empty(): goto true
    else: goto false
false: print "False"
       stop
true:  print "True"
       stop

```

Write a Pep8 assembly program for this. We assume that the string is an array of characters starting at address `w`, and that it is already in memory; the end of `w` is marked by the character `\x00`.

Use **indexed addressing** for accessing arrays. Use **stack-relative addressing** for accessing the stack.

Explain how your program works (use a flow-chart).

Test your program using the Pep8 simulator; save your program in a file while you debug.