Programming at Different Levels:
A Teaching Module for Undergraduate Computer Architecture Course

Xuejun Liang, Loretta A. Moore, and Jacqueline Jackson
Department of Computer Science
Jackson State University

Outlines

A. Introduction
B. Software
C. Evaluate Arithmetic Expressions
D. Sort Integer Arrays
E. Input/Output Integer Arrays
F. Problem Decomposition - Subroutines
G. Future Work

A. Introduction

- Computing with using computers can be carried out at different levels.
  - Application software, such as Microsoft Office Excel.
  - High-Level Programming Languages, such as C++, Java, Python.
  - Low-Level Assembly Programming Languages, such as MIPS, IA-32.
- Computing examples to illustrate how to solve the same problems using different computer programming languages
  - will expose students with varieties of computer architectures and programming languages, their similarities and differences, and different ways to solve these problems.
  - will help students to get a better understanding of computer architectures and programming languages, and to enhance their problem solving skills with using computers.

A. Introduction (Cont.)

- Four computer architectures (or assembly languages) are considered
  - (1) MIPS, (2) IA-32, (3) Accumulator-based, and (4) Stack-Based.
  - Real assembly languages are used for MIPS and IA-32 architectures.
  - The Accumulator-based and Stacked based architectures are simulated by using Python, C++, and Java.
- Three high-level programming languages are used
  - (1) Python, (2) C++, and (3) Java.
  - Eclipse is used for both C++ and Java programming.
- The Teaching Module Website
  - http://143.132.8.23/robotics/CTmodule/index.htm

B. Software

All the following software packages have been installed and tested on the Windows 7 platform

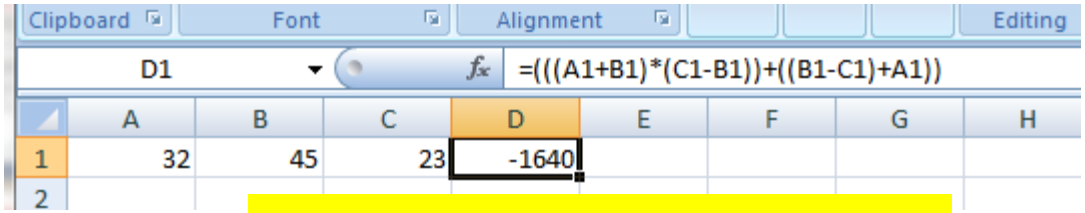
- Python 2.7
- PCSpim 9.1.4
- MinWG-w32 (GCC)
- NASM 2.10.09
- Eclipse IDE for Java Developers (indigo)
- Eclipse IDE for C/C++ Developers (indigo)

B. Software (Cont.)

- **Add C:\NASM\nasm-2.10.09 into the Path environment variable in Windows 7**
  - Select Window's Start →Control Panel → System and Security → System → Advanced system settings
  - Then, the System Properties dialogue window shows up, click the Environment Variables button.
  - Under System variables list, find and select the variable Path, and click the Edit... button. Then type "C:\NASM\nasm-2.10.09;" at the end of the variable value. When you are done, click OK.
- **Compile and run X86 assembly language program**
  - Run cmd to get a command line window
  - Compile your source code *exprX86. asm* by typing
    - *>nasm -f win32 exprX86.asm*
    - *>gcc exprX86.obj -o exprX86*
  - Run the program by typing
    - *>exprX86*

C. Evaluate Arithmetic Expressions

$$d = (a+b) * (c-b) + (b-c) + a$$



Microsoft Office Excel

```
a = 32
b = 45
c = 23
d = (a+b) * (c-b) + (b-c) + a
print d
```

Python

```
#include <iostream>
int main() {
   int a = 32;
   int b = 45;
   int c = 23;
   int d = (a+b)*(c-b)+(b-c)+a;
   std:: cout << “Answer: “ << d << endl;
   return 0;
}
```

C++

```
public class ComputeExpression {
   public static void main(String[] args) {
      int a = 32;
      int b = 45;
      int c = 23;
      int d = (a+b)*(c-b)+(b-c)+a;
      System.out.println(“Answer is  " + d);
   }
}
```

Java

Using MIPS Machine

$$d = (a+b) * (c-b) + (b-c) + a$$

```
            .text                       # text segment
            .globl main
main:
            lw      $t0, A              # $t0 = a
            lw      $t1, B              # $t1 = b
            lw      $t2, C              # $t2 = c
            sub     $t3, $t2, $t1       # $t3 = c-b
            sub     $t4, $t0, $t3       # $t4 = a - (c-b) = (b-c) + a
            add     $t5, $t0, $t1       # $t5 = a+b
            mul     $t6, $t5, $t3       # $t6 = (a+b)*(c-b)
            add     $t7, $t6, $t4       # $t7 = (a+b)*(c-b) + (b-c) + a
            sw      $t7, D
            ......
            li      $v0,10
            syscall                     # au revoir...

            .data                       # data segment
A:          .word 32
B:          .word 45
C:          .word 23
D:          .word 0
```

- Register-to-register arithmetic
- 3 operands

- Result display is omitted.
- SPIM uses system calls to print data in registers or character strings stored in memory.

Using IA-32 Machine

$$d = (a+b) * (c-b) + (b-c) + a$$

```
            global  _main
            section .text                   ; Text section
_main:
            mov     eax, [A]                ; eax = a;
            mov     ebx, [B]                ; ebx = b;
            add     eax, ebx                ; eax = a+b;
            sub     ebx, [C]                ; ebx = b-c;
            imul    ebx                     ; edx:eax = (a+b)*(b-c);
            add     ebx, [A]                ; ebx = (b-c)+a;
            sub     ebx, eax                ; ebx = (a+b)*(c-b)+(b-c)+a;
            mov     [D], ebx                ; [D] = (a+b)*(c-b)+(b-c)+a;
            ……
            ret

            section .data                   ; Data section
A:          dd      32
B:          dd      45
C:          dd      23
D:          dd      0
```

- Not register-to-register arithmetic
- 2 operands
- At most 1 operand is memory address

- Result display is omitted.
- The C function _printf are used for printing.
- Call _printf in the assembly code after pushing its actual parameters onto the stack.

Using Accumulator Machine

$$d = (a+b) * (c-b) + (b-c) + a$$

Assume that the accumulator-based machine has the following five instructions (functions), where acc is the accumulator.

Instruction	Meaning
ld(A)	(acc = memory[A])
add (A)	(acc = acc + memory[A])
mul (A)	(acc = acc * memory[A])
sub (A)	(acc = acc - memory[A])
st (A)	(memory[A] = acc)

```
# Accumulator code
ld(a)
add(b)
st(d)       # d = a+b
ld(c)
sub(b)    #acc = c-b
mul(d)   #acc = (a+b)*(c-b)
add(b)   #acc = (a+b)*(c-b)+b
sub(c)    #acc = (a+b)*(c-b)+(b-c)
add(a)    #acc = (a+b)*(c-b) +(b-c)+a
st(d)       # d =(a+b)*(c-b) + (b-c) + a
```

1 operand

Simulating Accumulator Machine

Python

```
acc = 0

Def ld(A):
    global acc
    acc = A

def add(A):
    global acc
    acc = acc + A

def mul(A):
    global acc
    acc = acc * A

def sub(A):
    global acc
    acc = acc - A
```

global variable acc

```
def st(A):
    global acc
    A[0] = acc

a = 32
b = 45
c = 23
d = [0]

# Accumulator code

print d
```

Python uses Call-by-value

Integer array d

Simulating Accumulator Machine

Java

```
public class ExprAccumulator {
   static int acc;

   static void ld(int A){
      acc = A;
   }

   static void add(int  A){
      acc = acc + A;
   }

   static void sub(int A){
      acc = acc - A;
   }

   static void mul(int A){
      acc = acc * A;
   }
```

static variable acc

static functions

```
   static void st(int [] A){
      A[0] = acc;
   }

   public static void main(String[] args) {
      int a = 32;
      int b = 45;
      int c = 23;
      int d = new int[1];

      //# Accumulator code

      System.out.println("The answer: " + d[0]);
   }
}
```

Java uses Call-by-value

Integer array d

Simulating Accumulator Machine

C++

```
#include <iostream>

int acc;

void ld(int A){
   acc = A;
}

void add(int A){
    acc = acc + A;
}

void mul(int A){
    acc = acc * A;
}

void sub(int A){
    acc = acc - A;
}
```

global variable acc

```
void st(int &A){
    A = acc;
}

int main() {
    int a = 32;
    int b = 45;
    int c = 23;
    int d = 0;

    //# Accumulator code

     std::cout << "The answer: " << d << endl;
     return 0;
}
```

C++ allows Call-by-reference

Integer variables a, b, c, d

Using Stack-Based Machine

d = (a+b) * (c-b) + (b-c) + a

Assume that the stack-based machine has the following five instructions (functions).

Instruction	Meaning
push (A)	(tos++; S[tos] = memory[A])
pop (A)	(memory[A] = S[tos]; tos--)
add	(S[tos-1] = S[tos-1]+S[tos]; tos--)
sub	(S[tos-1] = S[tos-1]-S[tos]; tos--)
mul	(S[tos-1] = S[tos-1]*S[tos]; tos--)

Postfix notation

a b + c b - * b c - + a +

```
# Stack-based code
push(a)
push(b)
add()        # a+b

push(c)
push(b)
sub()        # c-b

mul()        # (a+b)*(c-b)

push(b)
push(c)
sub()        # b-c

add()        # (a+b)*(c-b)+(b-c)

push(a)
add()        # (a+b)*(c-b)+(b-c)+a

pop(d)
```

0 operand

Simulating Stack-Base Machine

Python

```
Stack = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
tos = -1

def push(A):
    global tos
    tos = tos + 1
    Stack[tos] = A

def pop(A):
    global tos
    A[0] = Stack[tos]
    tos = tos - 1

def add():
    global tos
    Stack[tos-1] = Stack[tos-1] + Stack[tos]
    tos = tos - 1
```

global array Stack

global variable tos

Python uses Call-by-value

```
def sub():
    global tos
    Stack[tos-1] = Stack[tos-1] - Stack[tos]
    tos = tos - 1

def mul():
    global tos
    Stack[tos-1] = Stack[tos-1] * Stack[tos]
    tos = tos - 1

a = 32
b = 45
c = 23
d = [0]

# Stack-based code

print d
```

Integer array d

Simulating Stack-Base Machine

Java

```
public class ExprStackArray {
   static int [] st = new int[10];
   static int tos = -1;

   static void push(int A){
      tos = tos +1;
      st[tos] = A;
   }
   static void pop (int [] A){
      A[0] = st[tos];
      tos = tos - 1;
   }
   static void add(){
      st[tos-1] = st[tos-1] + st[tos];
      tos = tos - 1;
   }
   static void sub(){
      st[tos-1] = st[tos-1] - st[tos];
      tos = tos - 1;
   }
   static void mul(){
      st[tos-1] = st[tos-1] * st[tos];
      tos = tos - 1;
   }
   public static void main(String[] args) {
      int a = 32;
      int b = 45;
      int c = 23;
      int[] d = {0};

      //# Stack-based code
      System.out.println("The answer: " + d[0]);
   }
}
```

static
st
tos

Java uses
Call-by-value

Integer array
d

Simulating Stack-Base Machine

C++

```
#include <iostream>

int Stack[10];
int tos = -1;

void push(int A){
    tos = tos + 1;
    Stack[tos] = A;
}

void pop(int &A){
    A = Stack[tos];
    tos = tos - 1;
}

void add(){
    Stack[tos-1] = Stack[tos-1] + Stack[tos];
    tos = tos - 1;
}

void sub(){
    Stack[tos-1] = Stack[tos-1] - Stack[tos];
    tos = tos - 1;
}

void mul(){
    Stack[tos-1] = Stack[tos-1] * Stack[tos];
    tos = tos - 1;
}

int main() {
    int a = 32;
    int b = 45;
    int c = 23;
    int d = 0;

    //# Stack-based code

    std::cout << "The answer: " << d << endl;
    return 0;
}
```

global array Stack

global variable tos

C++ allows Call-by-reference

Integer variables a, b, c, d

Using Stack instead of Array to simulate Stack-base machine

```
Stack = []

def push(A):
    Stack.append(A)

def pop(A):
    A[0] = Stack.pop()

def add():
    x = Stack.pop()
    y = Stack.pop()
    z = y+x
    Stack.append(z)

def sub():
    x = Stack.pop()
    y = Stack.pop()
    z = y-x
    Stack.append(z)

def mul():
    x = Stack.pop()
    y = Stack.pop()
    z = y*x
    Stack.append(z)
```

Python

```
import java.util.Stack;

public class ExprStackStack {
   static Stack<Integer> st = new
Stack<Integer>();

   static void push(int A){
      st.push(A);
   }

   static void pop (int [] A){
      A[0] = st.pop();
   }

   static void add(){
      int x, y, z;
      x = st.pop();
      y = st.pop();
      z = x + y;
      st.push(z);
   }

   static void sub(){
      int x, y, z;
      x = st.pop();
      y = st.pop();
      z = y - x;
      st.push(z);
   }

   static void mul(){
      int x, y, z;
      x = st.pop();
      y = st.pop();
      z = x * y;
      st.push(z);
   }
```

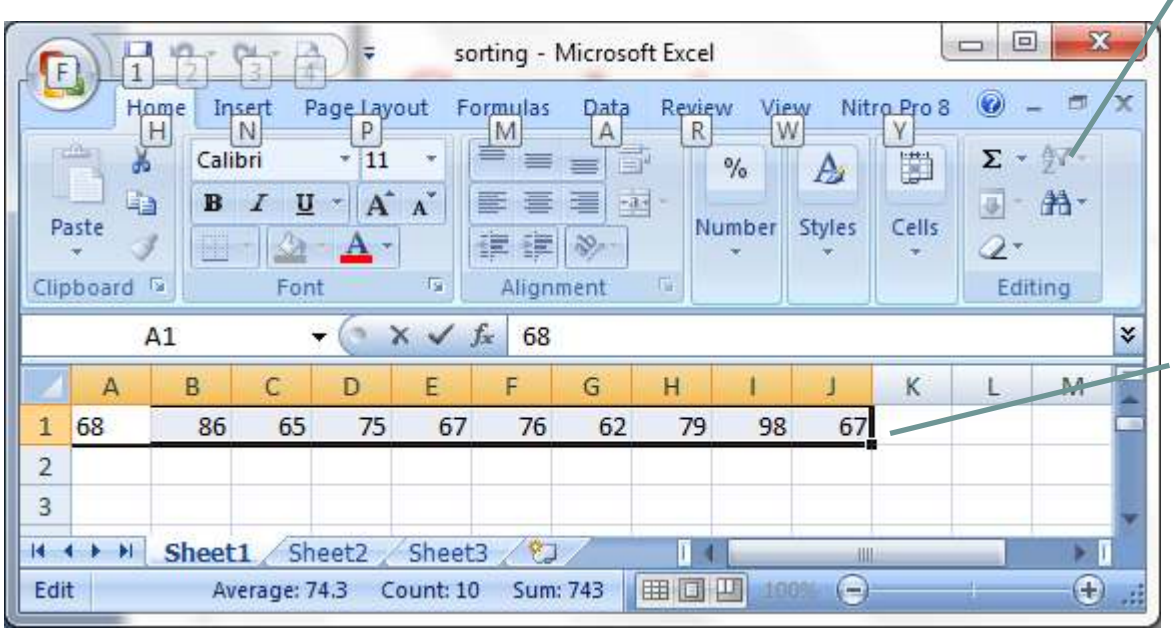
Java

```
#include <stack>
stack<int> st;

void push(int A){
   st.push(A);
}

void pop(int &A){
   A = st.top();
   st.pop();
}

void add(){
   int x, y, z;
   x = st.top(); st.pop();
   y = st.top(); st.pop();
   z = y+x;
   st.push(z);
}

void sub(){
   int x, y, z;
   x = st.top(); st.pop();
   y = st.top(); st.pop();
   z = y-x;
   st.push(z);
}

void mul(){
   int x, y, z;
   x = st.top(); st.pop();
   y = st.top(); st.pop();
   z = y*x;
   st.push(z);
}
```

C++

D. Sorting Arrays

Microsoft Office Excel



Then, press this button

Select your data

Using the Built-in Sorting Function

```
a = [68, 86, 65, 75, 67, 76, 62, 79, 98, 67]
a.sort()
print a
```

Python

```
import java.util.Arrays;

public class Sorting {
    public static void main(String[] args) {
        int [] a = {68, 86, 65, 75, 67, 76, 62, 79, 98, 67};
        Arrays.sort(a);

        System.out.println(Arrays.toString(a));
    }
}
```

Java

Using the Built-in Sorting Function

```
#include <iostream>
#include <algorithm>
using namespace std;

int main() {
        const int len = 10;
        int a[len] = {68, 86, 65, 75, 67, 76, 62, 79, 98, 67};
        sort(a, a+len);

        for (int i=0; i< len; i++)
                cout << a[i] << " ";
        cout << endl;
        return 0;
}
```

C++

The Bubble Sort Algorithm

		1	2	3	4	5	6	7
0	68	62	62	62	62	62	62	62
1	86	68	65	65	65	65	65	65
2	65	86	68	67	67	67	67	67
3	75	65	86	68	67	67	67	67
4	67	75	67	86	68	68	68	68
5	76	67	75	67	86	75	75	75
6	62	76	67	75	75	86	76	76
7	79	67	76	76	76	76	86	79
8	98	79	79	79	79	79	79	86
9	67	98	98	98	98	98	98	98

Using the Bubble Sort Algorithm

```
a = [68, 86, 65, 75, 67, 76, 62, 79, 98, 67]
len = len(a)

current = 0;
while (current < len - 1):
    index = len-1
    while (index > current):
        if (a[index] < a[index-1]):
                tmp = a[index]
                a[index] = a[index-1]
                a[index-1] = tmp
        index = index - 1
    current = current + 1

print a
```

Python

Using the Bubble Sort Algorithm

Java

```
import java.util.Arrays;

public class SortBubble {
   public static void main(String[] args) {
      int [] a = {68, 86, 65, 75, 67, 76, 62, 79, 98, 67};
      int len = a.length;

      int current = 0;
      while (current < len - 1)
      {
         for (int index = len-1; index > current; index--)
            if (a[index] < a[index-1]){
                int tmp = a[index];
                a[index] = a[index-1];
                a[index-1] = tmp;
            }
         current++;
      }

      System.out.println(Arrays.toString(a));
   }
}
```

Using the Bubble Sort Algorithm

C++

```
int main() {
    const int len = 10;
    int a[len] = {68, 86, 65, 75, 67, 76, 62, 79, 98, 67};

    int current = 0;
    while (current < len - 1)
    {
        for (int index = len-1; index > current; index--)
            if (a[index] < a[index-1]){
                int tmp = a[index];
                a[index] = a[index-1];
                a[index-1] = tmp;
            }
        current++;
    }

    for (int i=0; i< len; i++)
        std::cout << a[i] << " ";
    std::cout << endl;

    return 0;
}
```

Branch and Loop in Assembly

High level

```
if (x < y)
   A;
C;
```

```
if (x < y)
   A;
else
   B;
C;
```

```
while (x < y)
   A;
C;
```

```
        if (x >= y) goto c
        A
c:      C;
```

```
        if (x >= y) goto b
        A
        goto c
b:      B;
c:      C
```

```
a:      if (x >= y) goto c
        A
        goto a
c:      C
```

Assembly level

Array Access in Assembly

High level	
A[0]	**68**
A[1]	86
A[2]	65
A[3]	75
A[4]	67
A[5]	76
A[6]	62
A[7]	79
A[8]	98
A[9]	67

Assembly level	
A+0	**68**
A+4	86
A+8	65
A+12	75
A+16	67
A+20	76
A+24	62
A+28	79
A+32	98
A+36	67

MIPS

```
                .text
                .globl main
main:
                move    $t0, $0                 # $t0 = current
                lw      $t1, count              # $t1 = 10
                sll     $t1, $t1, 2             # $t1 = 40
                addi    $t1, $t1, -4            # $t1 = 36
loop1:
                beq     $t0, $t1, done
                move    $t2, $t1                # $t2 = index
loop2:
                beq     $t2, $t0, next
                addi    $t3, $t2, -4
                lw      $t4, A($t2)
                lw      $t5, A($t3)
                bge     $t4, $t5, continue
                sw      $t5, A($t2)
                sw      $t4, A($t3)
continue:
                addi    $t2, $t2, -4
                b       loop2
next:
                addi    $t0, $t0, 4
                b       loop1
done:
                li      $v0,10
                syscall                         # au revoir...

                .data
A:              .word 68, 86, 65, 75, 67, 76, 62, 79, 98, 67
count:          .word 10
```

IA-32

```
                global  _main
                section .text
_main:
                sub     edi, edi            ; edi = current
                mov     ecx, [count]        ; ecx = 10
                shl     ecx, 2              ; ecx = 40
                sub     ecx, 4              ; ecx = 36
loop1:
                cmp     edi, ecx
                je      done
                mov     eax, ecx            ; eax = index
loop2:
                cmp     eax, edi
                je      next
                mov     ebx, [eax+A]
                mov     edx, [eax+A-4]
                cmp     ebx, edx
                jge     continue
                mov     [eax+A], edx
                mov     [eax+A-4], ebx

continue:
                sub     eax, 4
                jmp     loop2
next:
                add     edi, 4
                jmp     loop1
done:
                ret

                section .data
A:              dd  68, 86, 65, 75, 67, 76, 62, 79, 98, 67
count:          dd  10
```

E. Read/Write Integer Arrays

Python

```
a = []
length = int(raw_input("Enter the array length: "))
for i in range(0, length):
    if( i == 0):
        a.append(int(raw_input(Enter your integer array)))
    else
        a.append(int(raw_input()))
```

Java

```
public class SortBubble {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.print("Enter the array length:  ");
        int length = in.nextInt();
        int [] a = new int[length];
        System.out.print("Enter your integer array:  ");
        for(int i = 0; i < length; i++)
            a[i] = in.nextInt();
    }
}
```

E. Read/Write Integer Arrays (Cont.)

C++

```
int main() {
    int length;
    cout << "Enter the array length: " << endl;
    cin >> length;
    int a[length];
    cout << "Enter your integer array: " << endl;
    for(int i = 0; i < length; i++)
        cin >> a[i];
    return 0;
}
```

C

```
int main() {
    int length;
    printf("Enter the array length: " );
    scanf("%d", length);
    int a[length];
    printf( "Enter your integer array:" );
    for(int i = 0; i < length; i++)
        scanf("%d", a[i]);
    return 0;
}
```

MPIS

```
        .text
        .globl main
main:
        #Read count
        li      $v0, 5
        syscall                 # read an integer
        sw      $v0, count

        # Read integer array
        move $t0, $0
        lw      $t1, count
        sll     $t1, $t1, 2
loop1:
        beq     $t0, $t1, done1
        li      $v0,5
        syscall                 # read an integer
        sw      $v0, A($t0)
        addi    $t0, $t0, 4
        b       loop1
done1:
```

```
        # print the integer array
        move $t0, $0
        lw      $t1, count
        sll     $t1, $t1, 2
loop2:
        beq     $t0, $t1, done2
        lw      $a0, A($t0)
        li      $v0,1
        syscall                 # print an integer
        la      $a0, sp
        li      $v0,4
        syscall                 # print the space
        addi    $t0, $t0, 4
        b       loop2
done2:
        li $v0,10
        syscall                 # au revoir...

        .data
A:          .word 0:20
count:      .word 0
sp:         .asciiz " "
```

IA-32

```
        global  _main
        extern  _printf
        extern _scanf

        section .text
_main:
        ;read count
        push   count
        push   rfmat
        call    _scanf       ;read integer
        add    esp, 8
        ;read integer array
        mov    ecx, [count];
        mov    edi, A
read:
        push   ecx
        push   edi
        push   rfmat
        call    _scanf       ;read integer
        add    esp, 8
        add    edi, 4
        pop    ecx
        loop   read
```

```
        ;print integer array
        mov    edi, A
        mov    ecx, [count]
print:
        push   ecx
        mov    ebx, [edi]
        push   ebx
        push   wfmat
        call    _printf      ;print integer
        add    esp, 8
        add    edi, 4
        pop    ecx
        loop   print
        ret

        .section .bss
A:          resd 20
count:   resd 1

        . section .data
rfmat:   db  '%d', 0
wfmat:  db  '%6d', 0
end:      db  10, 0
```

F. Problem Decomposition: Subroutines

- Get an input integer array from the keyboard,
- Sort the array using the bubble-up algorithm, and
- Print the sorted integer array

Python

```
def bubbleSort(nums, len):
    current = 0
    while (current < len - 1):
        index = len-1;
        while (index > current):
            if (nums[index] < nums[index-1]):
                tmp = nums[index]
                nums[index] = nums[index-1]
                nums[index-1] = tmp
            index = index - 1
        current = current + 1

# get the array length and the array
bubbleSort(a, length)
# print the result
```

Java

```
import java.util.*;
import java.io.*;

public class SortBubble1 {
    public static void BubbleSort(int [] nums) {
      int len = nums.length;
      int current = 0;
      while (current < len - 1){
         for (int index = len-1; index > current; index--)
            if (nums[index] < nums[index-1]){
               int tmp = nums[index];
               nums[index] = nums[index-1];
               nums[index-1] = tmp;
            }
         current++;
      }
    }

    public static void main(String[] args) {
      //get input array
      BubbleSort(a);
      //print results
    }
}
```

C++

```
#include <iostream>
using namespace std;

void bubbleSort(int nums[], int len) {
    int current = 0;
    while (current < len - 1) {
        for (int index = len-1; index > current; index--)
            if (nums[index] < nums[index-1]){
                int tmp = nums[index];
                nums[index] = nums[index-1];
                nums[index-1] = tmp;
            }
        current++;
    }
}

int main() {
    //get input array
    bubbleSort(a, length);
    //print results
}
```

MIPS

```
        .text
        .globl main
main:
        #Read count
        # Read integer array

        la      $a0, A
        lw      $a1, count

        jal     bubbleSort

        # print the result

        li $v0,10
        syscall                         # au revoir...

#-----------------------------------------------------
# bubbleSort:: Sort an array in ascending order
#               a0 - holds starting address of array
#               a1 - holds the length of array
#-----------------------------------------------------
```

```
bubbleSort:
        sll     $a1, $a1, 2
        addi    $a1, $a1, -4
        add     $a1, $a1, $a0
loop1:
        beq     $a0, $a1, done1
        move $t0, $a1
loop2:
        beq     $t0, $a0, done2
        lw      $t1, ($t0)
        lw      $t2, -4($t0)
        bge     $t1, $t2, continue
        sw      $t2, ($t0)
        sw      $t1, -4($t0)
continue:
        addi    $t0, $t0, -4
        b       loop2
done2:
        addi    $a0, $a0, 4
        b       loop1
done1:
        jr      $ra
```

IA-32

```
        global  _main
        extern  _printf
        extern  _scanf

        section  .text
_main:
        ;Read count
        ;Read integer array and stored in A

        mov   edi, A
        mov   ecx, [count]

        call    bubbleSort

        ;Print the sorted integer array:
        ret

;--------------------------------------------------
; bubbleSort:: Sort an array in ascending order
;             edi - holds starting address of array
;             ecx - holds the length of array
;--------------------------------------------------
```



```
bubbleSort:
        shl     ecx, 2
        sub     ecx, 4
        add     ecx, edi
loop1:
        cmp     edi, ecx
        je      done
        mov     eax, ecx
loop2:
        cmp     eax, edi
        je      next
        mov     ebx, [eax]
        mov     edx, [eax-4]
        cmp     ebx, edx
        jge     continue
        mov     [eax], edx
        mov     [eax-4], ebx
continue:
        sub     eax, 4
        jmp     loop2
next:
        add     edi, 4
        jmp     loop1
done:
        ret
```

Future Work

- Illustrate the stack-based machine by using the Java byte code (or Java Virtual Machine) directly.
- Illustrate implementations of basic data type such as two-dimensional array and class at assembly language level.
- Provide some applications that can be more effectively and simply solved by using assembly languages than high-level programming languages.
- Multi-core processor programming and architecture
- GPU (graphics processing unit) programming model and architecture

Thank You

Questions?