Assembly Language

Part 3: arrays
Indexed addressing and arrays

- Recall these basic concepts concerning variables:
  - A variable in HLL is a memory chunk at machine level
  - In HLL, we refer to variable by name; at machine level, we refer to it by address
  - A variable in assembly language may be referred to by name, but the value of that name is the address of the memory chunk
Indexed addressing and arrays

- An array at either level is just a bigger chunk of memory
- At the high level, we refer to an array by name, and its individual elements by index
  - The index is an offset added to the initial address (referred to by name)
  - The amount added is the index multiplied by the size of the data item
- At the machine architecture and assembly levels, we refer to the address of the array’s start – in assembly language, the symbol of an array is the address of the first element
Arrays and addressing modes

- Pep/8 programs with arrays use the following addressing modes:
  - indexed: Memory[Op specifier + X]
  - stack-indexed: Memory[SP + Op specifier + X]
  - stack-indexed deferred:
    Memory [Memory[SP + Op specifier] + X]

- Each element of an array is accessed by:
  - loading its index into X (the index register)
  - multiplying by the number of bytes per cell
  - using stack-indexed addressing
#include <iostream>
using namespace std;

int array[4];  // global array variable
int n;         // global index variable

int main() {
    cout << "Enter integer values: " << endl;
    for (n=0; n<4; n++) {
        cout << "Value " << n << " : ";
        cin >> array[n];
    }
    cout << "Here they come again:" << endl;
    for(n=3; n>=0; n--) {
        cout << "Value " << n << " : " << array[n] << endl;
    }
    return 0;
}
Pep8 Translation

br main
;*********** global string constants
out1: .ascii "Enter integer values:\x00"
value: .ascii "Value \x00"
colon: .ascii ":: \x00"
out2: .ascii "Here they come again:\x00"
;*********** global variables
array: .block 8 ; space for 4 ints
n: .block 2 ; space for 1 int
Pep8 Translation continued

main: stro out1, d
charo '\n', i
ldx 0, i ; set up counter for first loop
stx n, d
forloop1: cpx 4, i
brge endf1
aslx ; an integer is 2 bytes so we
stro value, d ; multiply by 2 to get indexing right
deco n, d
stro colon, d
stro colon, d
deci array, x ; read value into array using indexed mode
ldx n, d
addx 1, i
stx n, d
br forloop1
endf1: stro out2, d  
charo '\n', i  
ldx 3, i                    ; set up counter for 2nd loop  
stx n, d  
forloop2: cpx 0, i  
brlt endf2  
stro value, d  
decco n, d  
stro colon, d  
aslx  
decco array, x              ; write value using indexed mode  
charo '\n', i  
ldx n, d  
subx 1, i  
stx n, d  
br forloop2  
endf2: stop  
.end
Local array variables

- Local array variables are allocated on the runtime stack during program execution:
  - SUBSP allocates space for array
  - ADDSP deallocates it

- Next slides show same program with local variables instead of global
Array program with local variables

br main
;*********** global string constants
out1: .ascii "Enter integer values: \x00"
value: .ascii "Value \x00"
colon: .ascii ": \x00"
out2: .ascii "Here they come again: \x00"
;*********** local variables
array: .equate 2 ; offsets only; actual
n: .equate 0 ; memory is stack
main: subsp 10, i ; set up stack memory
stro out1, d ; for array & n
charo '\n', i

Previous version:

Memory allocated for global variables using .block:

array: .block 8
n: .block 2
; didn’t use stack
Local array, continued

ldx 0, i
stx n, s
forloop1: cpx 4, i
brge endf1
asl x
stro value, d
deco n, s
stro colon, d
deci array, sx
ldx n, s
addx 1, i
stx n, s
br forloop1

Main differences from original in this section are addressing modes: using stack-relative (s) for variable n and stack-indexed (sx) for access to array elements.
Local array, continued

```
endf1: stro out2, d
charo '\n', i
ldx 3, i
stx n, s
forloop2: cpx 0, i
brtl endf2
stro value, d
deco n, s
stro colon, d
aslx
deco array, sx
charo '\n', i
ldx n, s
subx 1, i
stx n, s
br forloop2
endf2: addsp 10, i
stop
.end
```

Basically same as previous version except for use of stack relative and stack-indexed addressing mode; at end of program, we give back the stack memory.
Array parameters

- To pass array as parameter:
  - address of first element is pushed on runtime stack using MOVSPA and ADDA with immediate addressing
  - Element is accessed by loading its index into X, multiplying by # bytes per cell, and using stack-indexed deferred addressing
C++ program with array parameters

#include <iostream>
using namespace std;

void fillArray(int array[], int amt);
void displayArray (int array, int amt);

int main()
{
    int vector [10];
    int numVals;
    cout << "Enter number of values to input (10 or less): ";
    cin >> amt;
    fillArray(vector, numVals);
    cout << "Here are the values:"
    displayArray(vector, numVals);
    return 0;
}
C++ program with array parameters

```cpp
void fillArray(int array[], int amt)
{
    int n;
    for (n=0; n<amt; n++)
    {
        cout << "Enter an integer: ";
        cin >> array[n];
    }
}

void displayArray (int array, int amt)
{
    int n;
    for(n=0; n<amt; n++)
    {
        cout << array[n] << ' ';
        cout << endl;
    }
}
```
Pep8 equivalent

br main
;************ global string constants
prnum: .ascii "Enter number of values to input (10 or less): \x00"
outshow: .ascii "Here are the values:\x00"
prompt: .ascii "Enter an integer: \x00"
endl: .ascii "\n\x00"
space: .ascii " \x00"

As usual, we declare the string literals as global constants
Pep8 equivalent

;************ fillArray(int array[], int amt)
array1: .equate 6 ; relative positions (from the SP) of the
amt1: .equate 4 ; 2 parameters passed from main
n1: .equate 0
fillArr: subsp 2, i ; allocate memory for n1
ldx 0, l ; use X register to step thru array, starting at 0
stx n1, s ; for (n=0;
loop1: cpx amt1, sf ; n < amt
brge end1
Aslx ; multiply index by 2 to account for size of int
stro prompt, d ; cout << “Enter an integer: ”;
deci array1, sxf ; cin >> array[n]; note use of stack-indexed
ldx n1, s ; deferred mode (sxf)
addx 1, l ; n++
stx n1, s
br loop1
end1: ret2
Pep8 equivalent

;************* displayArray(int array[], int amt)
array2: .equate 6
amt2: .equate 4
n2: .equate 0
display: subsp 2, i
ldx 0, i
stx n2, s
loop2: cpx amt2, s
brge end2
aslx
deco array2, sxf
stro space, d
ldx n2, s
addx 1, i
stx n2, s
br loop2
end2: stro endl, d
ret2
Pep8 equivalent: first part of main

;************* main()
vector: .equate 2
numVals: .equate 0
main: subsp 22, I
movspa
adda vector, i
sta -2, s
movspa
adda numVals, I
sta -4, s
stro prnum, d
deci numVals, s

; make space for local variables
; 22: 2 for numVals (an int), 20 for array with
; with 10 int elements
; store addresses of vector & numVals
; on stack to get ready for parameter passing
; & stack-relative deferred addressing
Pep8 equivalent: second part of main

subsp 4, I ; move SP past parameters &
call fillArr ; call first function
addsp 4, I ; restore SP to original location
movspa
addda vector, I ; repeat previous process to set up for 2\textsuperscript{nd} function call
sta -2, s
lda numVals, s
sta -4, s
subsp 4, i
stro outshow, d ; cout << “Here are the values” << endl;
stro endl, d
call display ; call 2\textsuperscript{nd} function
stro endl, d
addsp 4, I ; restore SP from parameters
addsp 22, I ; restore SP from local variables (including array)
stop
.end