Arrays and Pointer Arithmetic

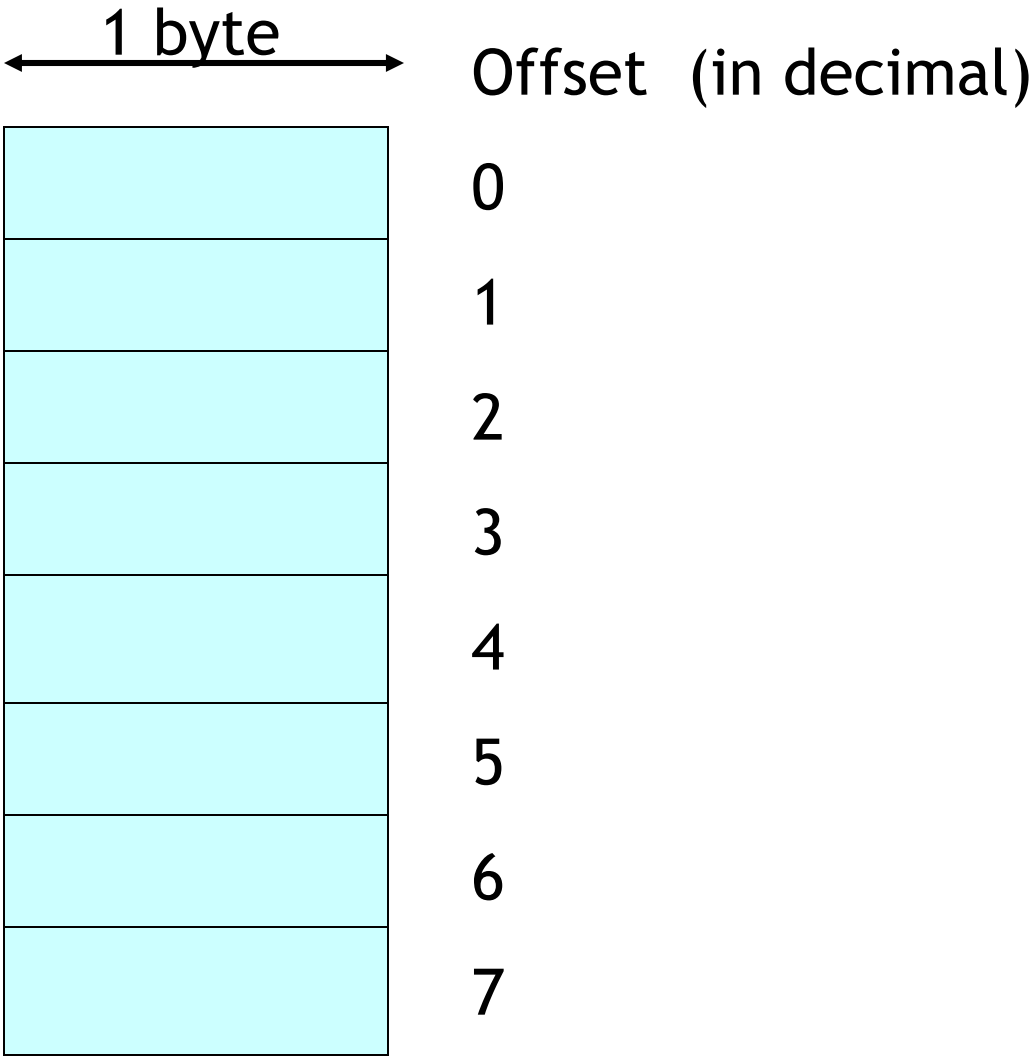
September 25, 2000

CSC201 Section 002

Fall, 2000

Memory

- Can be viewed as a 1-dimensional array of bytes
- Every byte has an offset, or index, relative to the start of memory

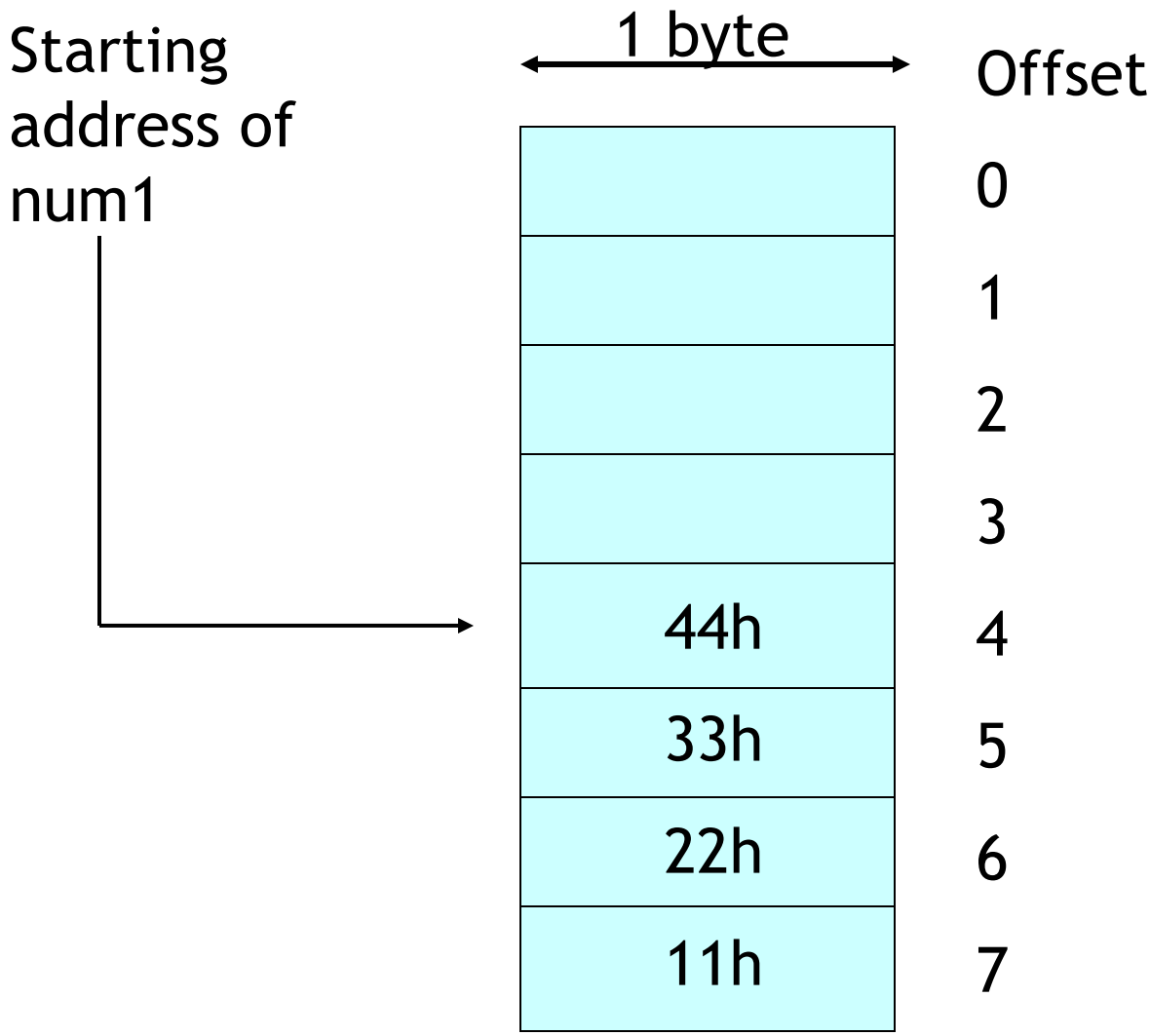


Doublewords

- Address of doubleword = address of first byte in memory
- Sometimes doublewords have to be aligned in memory
  - starting byte address must be a multiple of 4
- Storage of the bytes of a doubleword in memory is little-endian (!)

Little-Endian Byte Order

```
.data
Num1  dd      11223344h
```



Arrays

- A 1-D byte array can start anywhere in memory
- A byte in this array is specified by its offset relative to this starting address
- How does a programmer know the address of the start of the array?
  - SASM macro: "la addr, myarray"
  - addr is a doubleword which stores the address of variable myarray
  - equivalent to the & operator in the C programming language

In C:

```
Addr = &myarray;
```

Indexed Addressing Mode

- A SASM instruction refers to (I.e., addresses) a byte in memory at address addr as "m(addr)"
  - addr is a doubleword variable
- Can be used as the source or destination operand

```
.data
Ch1    db     0
Num1   dd     28
Num2   dd     ?
Addr1  dd     ?
Addr2  dd     ?

.code
       la     addr1, ch1
       la     addr2, num1
       moveb  m(addr1), 'a'
       move   num2, m(addr2)
```

1-D Array Definitions

- Use the "dup" directive to indicate number of elements

```
Charray     db     10  dup  (?)
```

Addressing Elements in a 1-D Array

- Offset = index * element_size_in_bytes
  - index starts at 0

C:

```
B1 = charray[2];
```

SASM:

```
la      addr1, charray
iadd    addr1, 2
moveb b1, m(addr1)
```

C:

```
Int1 = intarray[4];
```

SASM:

```
la      addr2, intarray
ladd    addr2, 16
move  int1, m(addr2)
```

Defining Records

- A record, or structure, is just an array of bytes

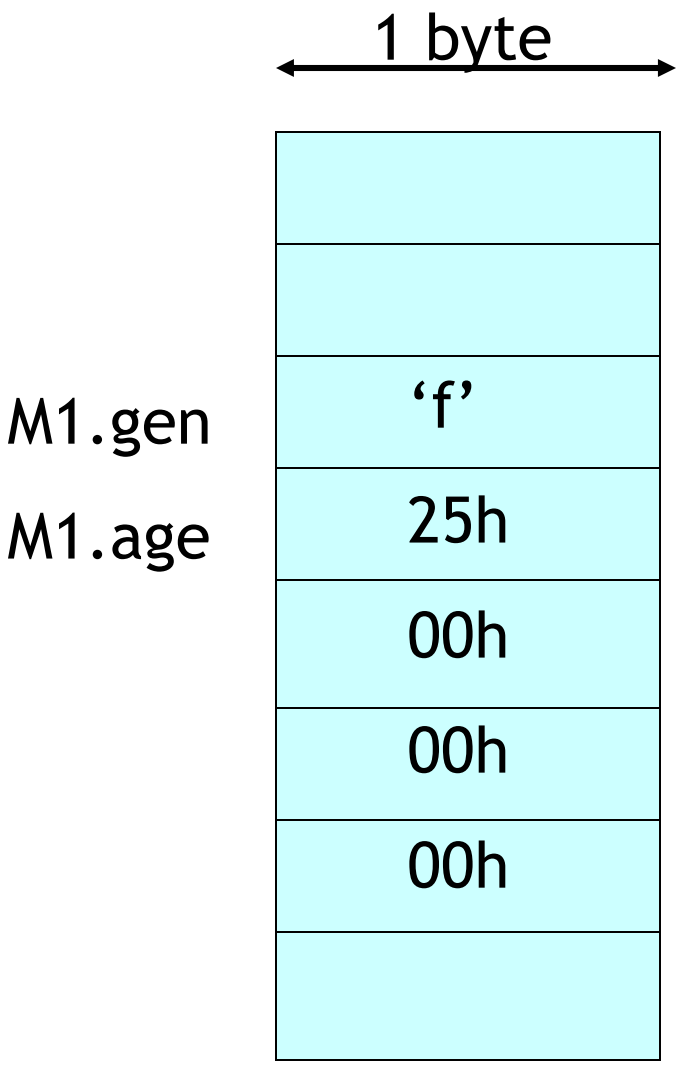
C:

```
Struct mystruct {
Char   gen;
Int    age;
} m1;
…
M1.gen = ‘f’;
M1.age = 00000025h;
```

SASM:

```
m1   db      5 dup (?)
Addr1 dd     ?
…
La      addr1, m1
Moveb m(addr1), ‘f’
Iadd    addr1, 1
Move   m(addr1), 00000025h
```

Defining Records



Two-Dimensional Arrays

- A 2-D array is actually a 1-D array, with elements that are 1-D arrays
- If a byte array has 2 rows and 3 columns, you can store it as...

1. a 1-D array with 2 elements, each of which is 3 bytes long (row-major order)
2. or, a 1-D array with 3 elements, each of which is 2 bytes long (column-major order)

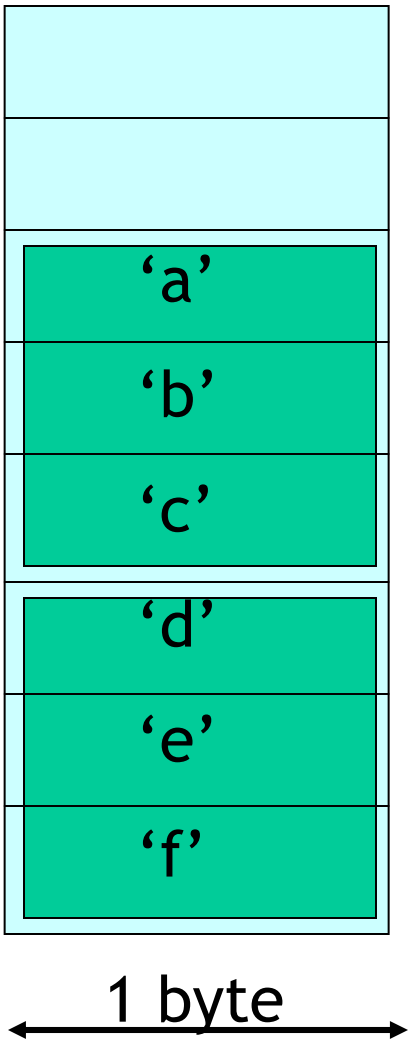
   row-major order is the common case

2-D Arrays

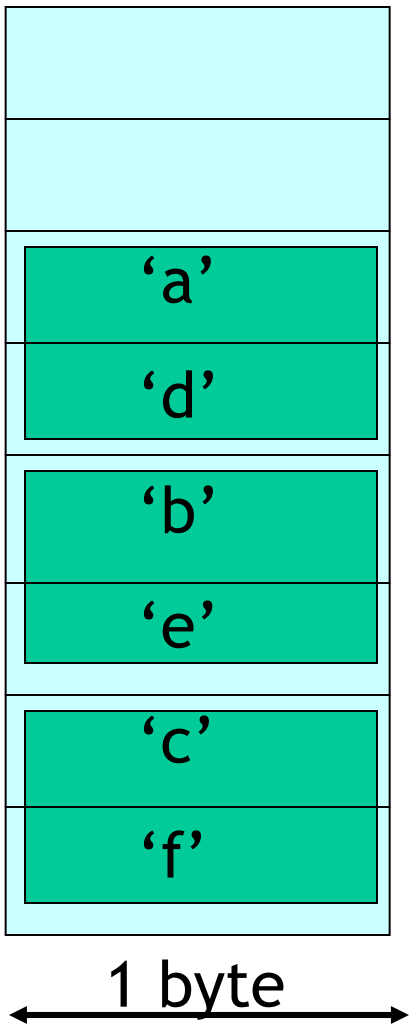
Here is a 2x3 array of characters to be stored:

‘a’	‘b’	‘c’
‘d’	‘e’	‘f’

Row-major order



Column-major order



Addressing Elements in a 2-D Array (Row-Major Order)

- Offset =
  row_index * number_of_columns * element_size_in_bytes +
  col_index * element_size_in_bytes

- Example:
  - number_of_columns = 6
  - row_index = 2
  - column_index = 4
  - element_size_in_bytes = 1
  - offset = 2 * 6 * 1 + 4 * 1 = 16

2-D Array Example

SASM:

```
Intarray        dd      1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
11, 12  ; 3x4 array, row1 = 1 2 3 4, etc.
Addr1           dd      ?


...     ; The code below changes the value of the 3rd
        ; element in the 2nd row to 99
la      addr1, intarray
move    off1, 1         ; 2nd row = row index of 1
imult   off1,4          ; number of columns = 4
imult   off1, 4         ; element size = 4 bytes
move    off2, 2         ; 3rd element = column index 2
imult   off2, 4         ; element size = 4 bytes
iadd    off1, off2
iadd    addr1, off1
move    m(addr1), 99
```