Section 1: Memory, Data, and Addressing

- Preliminaries
- Representing information as bits and bytes
- Organizing and addressing data in memory
- <u>Manipulating data in memory using C</u>
- Boolean algebra and bit-level manipulations

Addresses and Pointers in C

& = 'address of value'
** = 'value at address'*
or 'dereference'

- **Variable declarations**
  - int x, y;
  - Finds two locations in memory in which to store 2 integers (1 word each)
- **Pointer declarations use ***
  - int *ptr;
  - Declares a variable ptr that is a pointer to a data item that is an integer
- **Assignment to a pointer**
  - ptr = &x;
  - Assigns ptr to point to the address where x is stored

Addresses and Pointers in C

& = 'address of value'
** = 'value at address'*
or 'dereference'

- **To use the value pointed to by a pointer we use *dereference (*)***
  - Given a pointer, we can get the value it points to by using the * operator
  - *ptr is the value at the memory address given by the value of ptr
- **Examples**
  - If ptr = &x then y = *ptr + 1 is the same as y = x + 1
  - If ptr = &y then y = *ptr + 1 is the same as y = y + 1
  - What is *(&x) equivalent to?

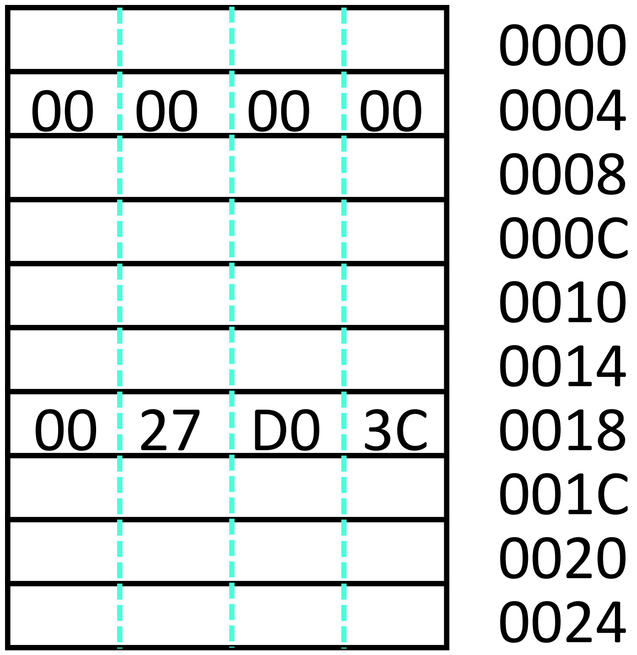
Addresses and Pointers in C

& = 'address of value'
** = 'value at address'*
or 'dereference'

- **We can do arithmetic on pointers**
  - ptr = ptr + 1;   *// really adds 4: type of ptr is int*, and an int uses 4 bytes!*
  - Changes the value of the pointer so that it now points to the next data item in memory (that may be y, or it may not – this is dangerous!)

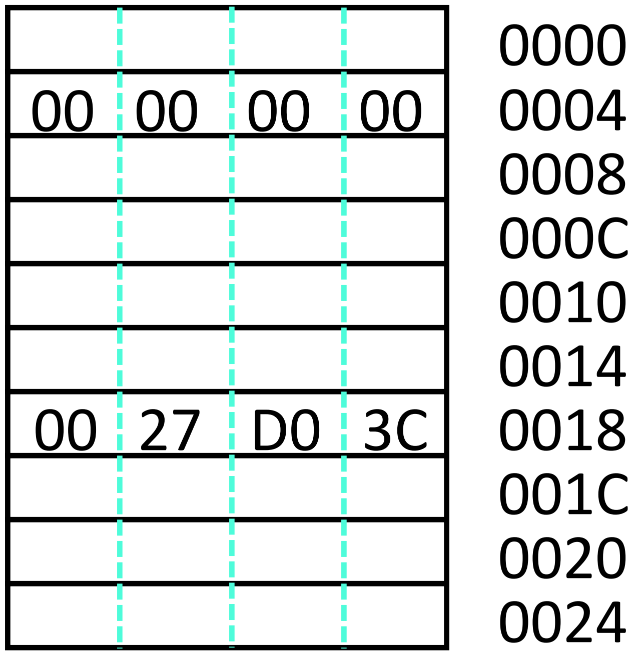
Assignment in C

- **Left-hand-side = right-hand-side**
  - LHS must evaluate to a memory *location* (a variable)
  - RHS must evaluate to a *value* (could be an address!)
- **E.g., x at location 0x04, y at 0x18**
  - x originally 0x0, y originally 0x3CD02700



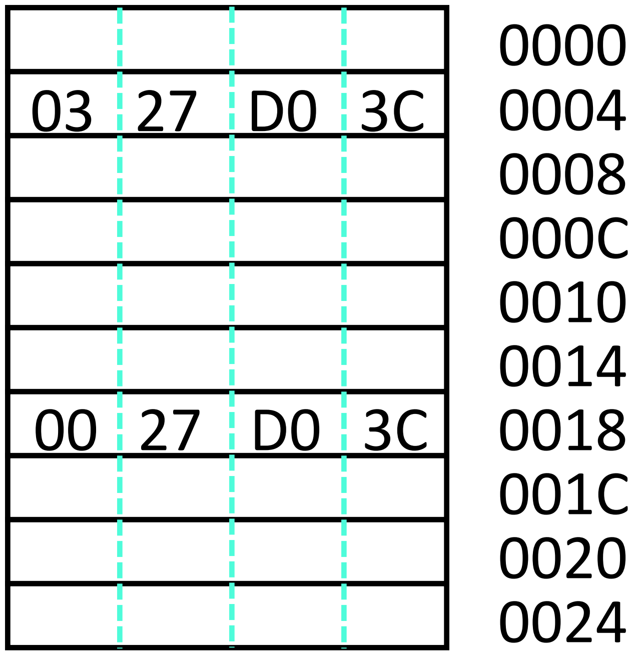
Assignment in C

- **Left-hand-side = right-hand-side**
  - LHS must evaluate to a memory *location* (a variable)
  - RHS must evaluate to a *value* (could be an address!)
- **E.g., x at location 0x04, y at 0x18**
  - x originally 0x0, y originally 0x3CD02700
  - int x, y;
    x = y + 3; //get value at y, add 3, put it in x



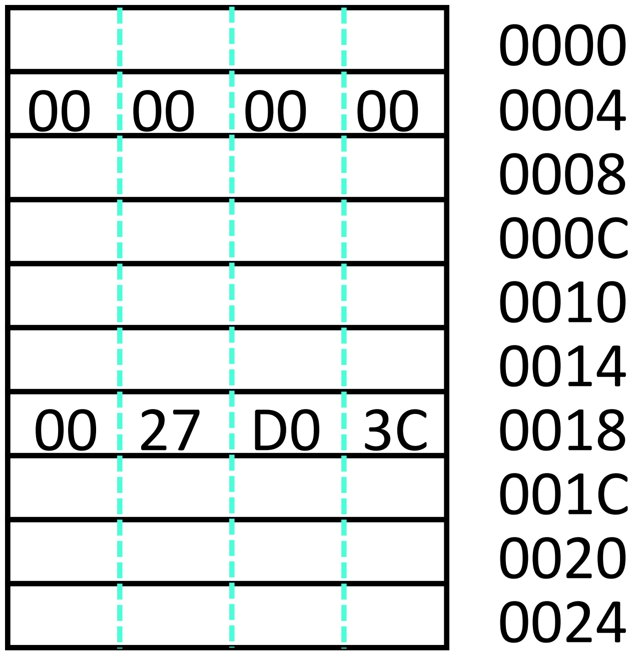
Assignment in C

- **Left-hand-side = right-hand-side**
  - LHS must evaluate to a memory *location* (a variable)
  - RHS must evaluate to a *value* (could be an address!)
- **E.g., x at location 0x04, y at 0x18**
  - x originally 0x0, y originally 0x3CD02700
  - int x, y;
    x = y + 3; //get value at y, add 3, put it in x



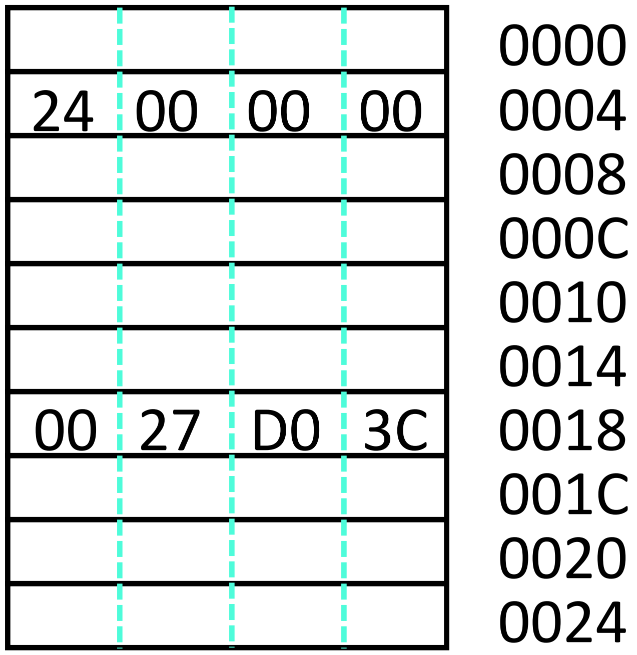
Assignment in C

- **Left-hand-side = right-hand-side**
  - LHS must evaluate to a memory *location* (a variable)
  - RHS must evaluate to a *value* (could be an address!)
- **E.g., x at location 0x04, y at 0x18**
  - x originally 0x0, y originally 0x3CD02700
  - int *x; int y;
    x = &y + 3; // get address of y, add ??



Assignment in C

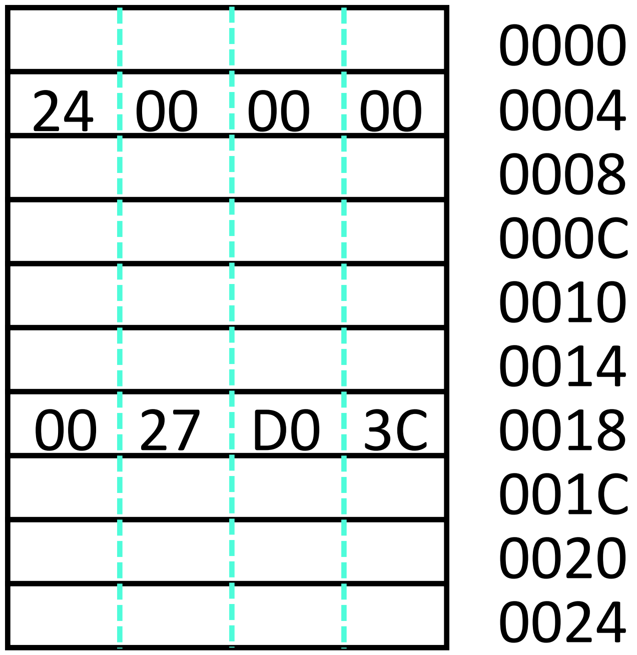
- **Left-hand-side = right-hand-side**
  - LHS must evaluate to a memory *location* (a variable)
  - RHS must evaluate to a *value* (could be an address!)
- **E.g., x at location 0x04, y at 0x18**
  - x originally 0x0, y originally 0x3CD02700
  - int *x; int y;
    x = &y + 3; // get address of y, add 12
    // 0x0018 + 0x000C = 0x0024



Assignment in C

- **Left-hand-side = right-hand-side**
  - LHS must evaluate to a memory *location* (a variable)
  - RHS must evaluate to a *value* (could be an address!)
- **E.g., x at location 0x04, y at 0x18**
  - x originally 0x0, y originally 0x3CD02700
  - int *x; int y;
    x = &y + 3; // get address of y, add 12
    // 0x0018 + 0x000C = 0x0024

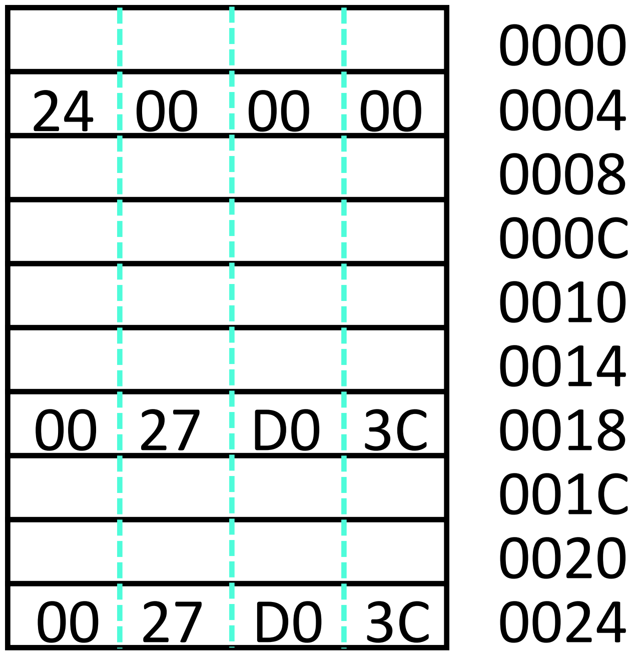
    *x = y; // value of y copied to
    // location to which x points



Assignment in C

- **Left-hand-side = right-hand-side**
  - LHS must evaluate to a memory *location* (a variable)
  - RHS must evaluate to a *value* (could be an address!)
- **E.g., x at location 0x04, y at 0x18**
  - x originally 0x0, y originally 0x3CD02700
  - int *x; int y;
    x = &y + 3; // get address of y, add 12
                        // 0x0018 + 0x000C = 0x0024

    *x = y; // value of y copied to
                // location to which x points



24 00 00 00
00 27 D0 3C
00 27 D0 3C
0000
0004
0008
000C
0010
0014
0018
001C
0020
0024