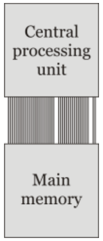
Outline

- In this lesson, we will:
  - Describe the purpose of main memory
  - Explain how each byte has its own address
  - See how these addresses are passed *in parallel* to main memory
    - This limits the maximum amount of memory that can be accessed
  - See how main memory is used for executing programs
  - See how main memory is used for storing local arrays

Main memory

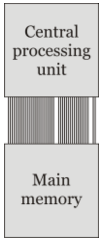
- Programs are stored in persistent memory
- While a program is running,
  the program requires temporary memory to execute
- Long term memory can be slow,
  but memory required during execution must be relatively fast

- Main memory provides temporary memory that can be accessed
  - The central processing unit (CPU) communicates with main memory



Main memory

- To access main memory:
  - Each byte in main memory has a unique address
  - The CPU sends an address and either flags to either:
    - Retrieve the value of the byte at that address
    - Set the byte at that address to a specific value



Main memory

- Okay, so each byte has its own address
  - This is called byte-addressable
- If you want to change just one bit,
  you must use the bit-wise and
  bit-shift operators on a byte

0	00000000
1	00000000
2	00000000
3	00000000
4	00000000
5	00000000
6	00000000
7	00000000
8	00000000
9	00000000
10	00000000
11	00000000
12	00000000
13	00000000
14	00000000
15	00000000
16	00000000
17	00000000
18	00000000
19	00000000
20	00000000
21	00000000
22	00000000
23	00000000
24	00000000
25	00000000
26	00000000
27	00000000
⋮	⋮
53433	00000000

Main memory

- Next, since we are in binary:
  - Each address will be a binary number

0	00000000
1	00000000
10	00000000
11	00000000
100	00000000
101	00000000
110	00000000
111	00000000
1000	00000000
1001	00000000
1010	00000000
1011	00000000
1100	00000000
1101	00000000
1110	00000000
1111	00000000
10000	00000000
10001	00000000
10010	00000000
10011	00000000
10100	00000000
10101	00000000
10110	00000000
10111	00000000
11000	00000000
11001	00000000
11010	00000000
11011	00000000
⋮	⋮
1101000010111001	00000000

Serial versus parallel communication

- If you read a 10 digit number to a friend,
  you are communicating *serially*; one digit at a time
  - Also, you must know when you're starting and stopping
- If you and nine friends each communicates one of those digits to one of ten corresponding friends,
  you are communicating *in parallel*; all ten digits at once
- The first is cheaper, the second is faster
- The communication between the CPU and main memory is parallel
  - A bus of n lines has each line carrying one bit of an address

Addresses

- In a computer,
  an *address bus* has n lines, each sending a `0` or a `1`
  - This allows 2^n different addresses
- The Intel 386 was the first common CPU with a 32-bit address bus
  - 32 wires connected the CPU and main memory carrying the address
- The first common CPU with a 64-bit addresses was the Nintendo 64
  - 64 wires connected the CPU and main memory carrying the address
- Incidentally, the Commodore 64 had 64 KiB of main memory
  - 64 KiB = 2^{16} bytes
  - This could be addressed with a 16-bit address

Addresses

- If every byte has its own address, then
  - A 32-bit address can uniquely address 2^{32} = 4 GiB
  - A 64-bit address can uniquely address 2^{64} = 67 108 864 TiB
- The restriction of 32-bit computers to accessing only 4 GiB of main memory led to the general adoption of 64-bit computers

Addresses

- We could thus display all addresses by showing all 32 bits

00000000000000000000000000000000	00000000
00000000000000000000000000000001	00000000
00000000000000000000000000000010	00000000
00000000000000000000000000000011	00000000
00000000000000000000000000000100	00000000
00000000000000000000000000000101	00000000
00000000000000000000000000000110	00000000
00000000000000000000000000000111	00000000
00000000000000000000000000001000	00000000
00000000000000000000000000001001	00000000
00000000000000000000000000001010	00000000
00000000000000000000000000001011	00000000
00000000000000000000000000001100	00000000
00000000000000000000000000001101	00000000
00000000000000000000000000001110	00000000
00000000000000000000000000001111	00000000
00000000000000000000000000010000	00000000
00000000000000000000000000100001	00000000
00000000000000000000000000100010	00000000
00000000000000000000000000100011	00000000
00000000000000000000000000100100	00000000
00000000000000000000000000100101	00000000
00000000000000000000000000100110	00000000
00000000000000000000000000100111	00000000
00000000000000000000000000101000	00000000
00000000000000000000000000101001	00000000
00000000000000000000000000101010	00000000
00000000000000000000000000101011	00000000
⋮	⋮
11111111111111111111111111111111	00000000

Addresses

- Recall, however, that we can represent four bits with one hexadecimal digit
  - By convention, we will
    - Leave off leading zeros
    - Use ellipsis for intermediate `f`s
  - For example,
    `a310` instead of `0000a310`
    `f…fb08` instead of `fffffb08`
  - If we are obviously discussing addresses, we may leave off the `0x`

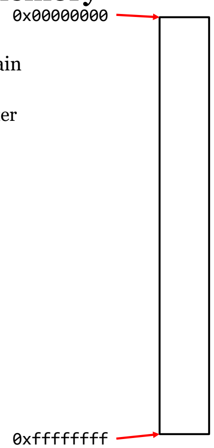
00000000	00000000
00000001	00000000
00000002	00000000
00000003	00000000
00000004	00000000
00000005	00000000
00000006	00000000
00000007	00000000
00000008	00000000
00000009	00000000
0000000a	00000000
0000000b	00000000
0000000c	00000000
0000000d	00000000
0000000e	00000000
0000000f	00000000
00000010	00000000
00000011	00000000
00000012	00000000
00000013	00000000
00000014	00000000
00000015	00000000
00000016	00000000
00000017	00000000
00000018	00000000
00000019	00000000
0000001a	00000000
0000001b	00000000
⋮	00000000
ffffffff	00000000

Addresses

- Thus, given this 32-bit address,
  `0b11110101011011110000101010101110`
  we could write it as
  `0xf56e155e`
- Similarly, given this 64-bit address in hexadecimal:
  `0x0003a58f293e5b80`
  we could determine the bits
  `0b0000000000000011101001011000111100101001001111100101101110000000`

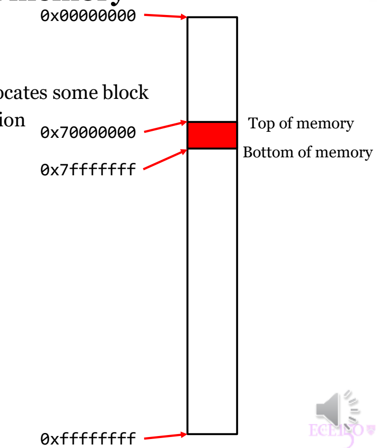
Main memory

- Thus, we could visualize all of main memory as shown here
  - Assume this is a 32-bit computer with 4 GiB of main memory



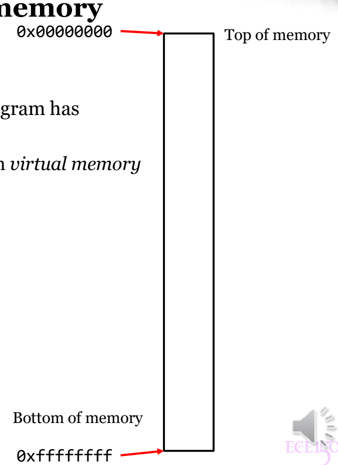
Main memory

- When a program is executed,
  the operating system allocates some block of memory for its execution



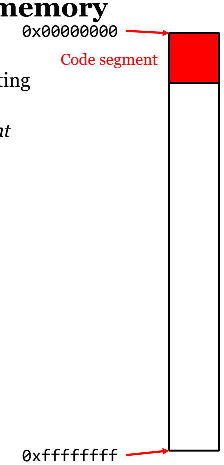
Main memory

- For the purpose of this course,
  we will assume that the program has access to all of memory
  - This is actually achievable with *virtual memory*



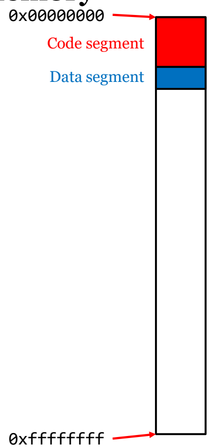
Main memory

- The instructions are stored starting at the top of memory
  - This is called the *code segment*



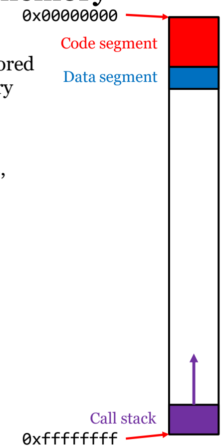
Main memory

- Literals are stored next in the *data segment*



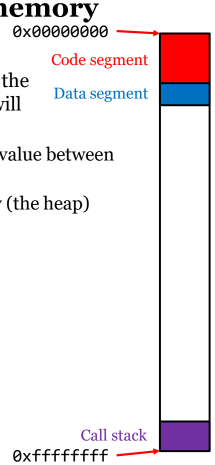
Main memory

- Memory for local variables is stored starting at the bottom of memory
  - This is called the *call stack*
- As we need more local variables, the call stack will grow towards the top of memory



Main memory

- The remaining memory between the data segment and the call stack will be used for additional features:
  - Local variables that keep their value between function calls (`static`)
  - Dynamically allocated memory (the heap)



Local arrays

- Suppose we have this program:

```
#include <iostream>

int main();

int main() {
    int data[5];

    std::cout << data << std::endl;
    return 0;
}
```

Output:
`0xffff3d80`

⋮	⋮	
ffff3d7f	00000000	
ffff3d80	00000000	data[0]
ffff3d81	00000000	
ffff3d82	00000000	
ffff3d83	00000000	
ffff3d84	00000000	data[1]
ffff3d85	00000000	
ffff3d86	00000000	
ffff3d87	00000000	
ffff3d88	00000000	data[2]
ffff3d89	00000000	
ffff3d8a	00000000	
ffff3d8b	00000000	
ffff3d8c	00000000	data[3]
ffff3d8d	00000000	
ffff3d8e	00000000	
ffff3d8f	00000000	
ffff3d90	00000000	data[4]
ffff3d91	00000000	
ffff3d92	00000000	
ffff3d93	00000000	
ffff3d94	00000000	
⋮	⋮	

Summary

- Following this lesson, you now
  - Know that main memory is byte addressable and each byte has its own unique address
  - Know addresses are passed in parallel through an address bus with a fixed number of *n* lines or bits
  - Understand that this limits available main memory to 2^n bytes
  - Know that addresses are represented as hexadecimal digits
  - Understand that an executing program occupies a
    - Code segment
    - Data segment
    - Call stack
  - Are aware of how an array may be stored in main memory

References

[1] No references?

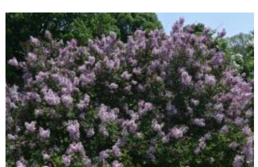
Colophon

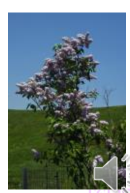
These slides were prepared using the Georgia typeface. Mathematical equations use Times New Roman, and source code is presented using `Consolas`.

The photographs of lilacs in bloom appearing on the title slide and accenting the top of each other slide were taken at the Royal Botanical Gardens on May 27, 2018 by Douglas Wilhelm Harder. Please see

https://www.rbg.ca/

for more information.





Disclaimer