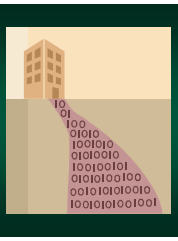
Memory Basics

Part 2

1



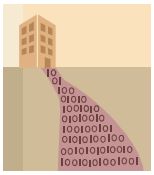
Computer Memory

Its… um…. I forgot….

2

Computer Memory

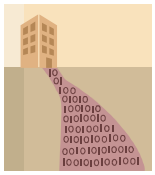
- Programs access and manipulate memory far more than you realize
- So, understanding it…
  - is vital to becoming a great assembly programmer
  - and understanding computer architecture



3

What is Memory?

- Memory is essentially an enormous array
- It is also, sometimes, referred to as *storage*
- It stores both running programs and their related data



4

Memory Addresses

- Memory is divided into a storage locations that can hold 1 byte (8 bits) of data
- Each byte has an *address*
  - unique value that refers to that specific byte
  - used to locate the exact byte the processor wants

Memory	
0	01000100
1	01000011
2	01101111
3	01101111
4	01101011

5

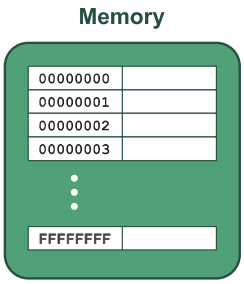
What is Memory?

- Each address is conceptually the same as an "index" in arrays
- … and you will write access memory as would an array

Memory	
0	01000100
1	01000011
2	01101111
3	01101111
4	01101011

6

Memory is a Hardware Array



7

von Neumann Architecture

The Information Superhighway

8

von Neumann Machine Architecture

- Modern computers are based on the design of John von Neumann
- His design greatly simplified the construction of (and use) computers

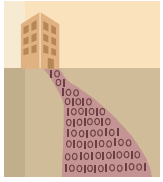
9

Some von Neumann Attributes

1. Programs are stored and executed in memory
2. Separation of processing from storage
3. Different system components communicate over a shared bus

10

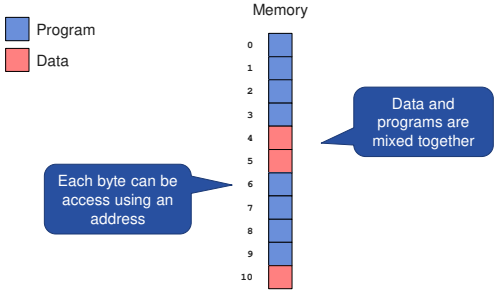
Memory Contains Data & Programs



- Data and programs are just binary numbers (stored in a series of bytes)
- …and are stored together
- Appreciating this is vital to understanding computer architecture

11

Memory Contains Data & Programs



12

The Bus

- Electronic pathway that transports data between components
- Think of it as a "highway"
  - data moves on shared paths
  - otherwise, the computer would be very complex

13

System Bus

- The information sent on the memory bus falls into 3 categories
- Three sets of signals
  - address bus
  - data bus
  - control bus

14

Address Bus

- Used by the processor to access a specific piece of data
- This "address" can be
  - a specific byte in memory
  - unique IO port
  - etc…
- The more bits it has, the more memory can be accessed

15

Address Bus Size Examples

- 8-bit → 2^8 = 256 bytes
- 16-bit → 2^{16} = 64 KB (65,536 bytes)
- 32-bit → 2^{32} = 4 GB (4,294,967,296 bytes)
- 64-bit → 2^{64} = 18 EB (18,446,744,073,709,551,616)

16

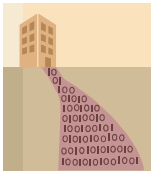
Historic Address Sizes

- Intel 8086
  - original 1982 IBM PC
  - 20-bit address bus (1 MB)
  - only 640 KB usable for programs
- MOS 6502 computers
  - Commodore 64, Apple II, Nintendo, etc…
  - 16-bit address bus (64 KB)

17

Data Bus

- The actual data travels over the *data bus*
- The number of bits that the processor uses – as its natural unit of data – is called a *word*



18

Data Bus

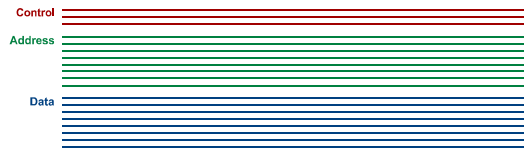
- Typically we define a system by word size
- Example:
  - 8-bit system uses 8 bit words
  - 16-bit system uses 16 bits (2 bytes) words
  - 32-bit system uses 32 bits (4 bytes) words
  - etc…

19

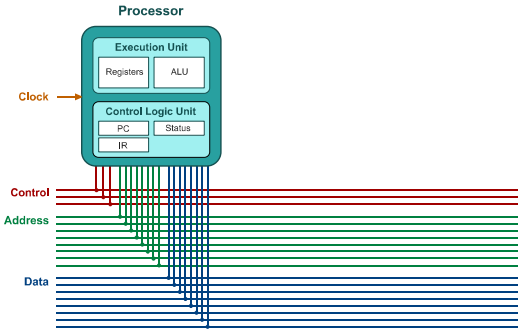
Control Bus

- The *control bus* controls the timing and synchronizes the subsystems
- Specifies what is happening
  - read data
  - write data
  - reset
  - etc…

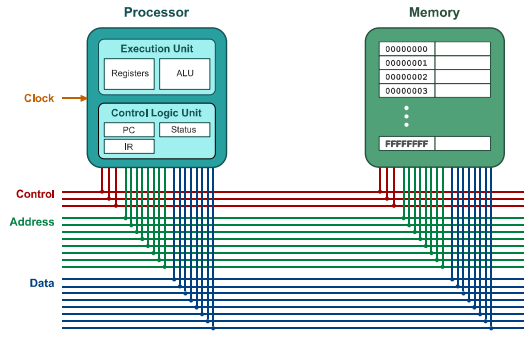
20



21



22



23

von Neumann Architecture Today

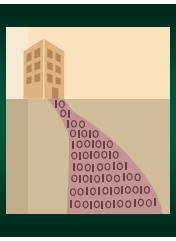
- Because of the emphasis on memory, most real-world systems use a modified version of his design
- In particular, they have a special high-speed bus between the processor and memory

24

von Neumann Architecture Today

- Think of it as a diamond-lane on a freeway
- … or as high-speed rail – which has a fixed source and destination and goes faster than the freeway

25



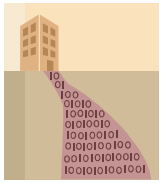
Accessing Data

Filling the registers

26

Accessing Data

- Processors use registers to hold data being computed
- So, how is data put in the registers to begin with?
- Data can come from two major sources



27

Immediates

- In programming, it is common to assign a constant to a variable
- This is also the case with processors and instructions

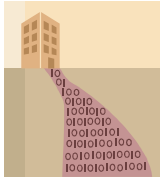
28

Immediates

- When a constant is stored as part of instruction, it is called an *immediate*
- Once the instruction is loaded by the processor, it is "immediately" available from the IR – hence, the name

29

Copying Data

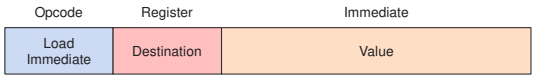


- Processors have a number of instructions that can copy data
- Each has a unique name – *not surprising since each does something different*

30

Load Immediate

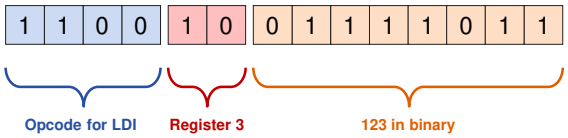
- A *Load Immediate* instruction, stores a constant into a register
- The instruction must store the destination register and the immediate value



31

Load Immediate Example (not x86)

```
LDI r2, 123
```



Opcode for LDI | Register 3 | 123 in binary

32

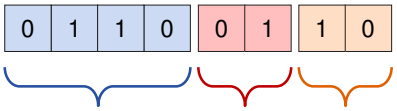
Transfer

- A *Transfer* instruction, copies the contents of one instruction into another
- The instruction must store both the destination and source register

33

Transfer Example (not x86)

```
TRA r1, r2
```



Opcode for Transfer | Register 1 | Register 2

34

Effective Addresses

- Processors also have the ability to access memory
- Since memory is a massive array, the processor needs to know the address

35

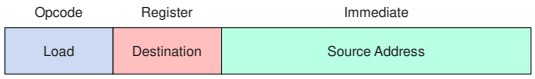
Accessing Memory

- The *effective address* is used to access memory
- Often it is created by combining multiple values
- … but we will cover that later in the semester

36

Load

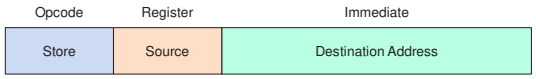
- A *Load* instruction, reads data from memory (at a specified address)
- This data is then stored into the destination register



37

Store

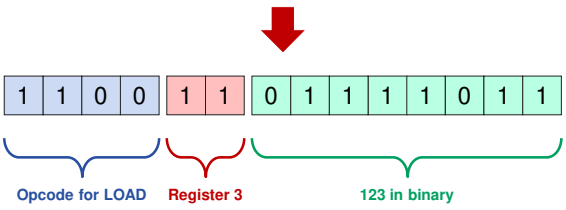
- A Store instruction, writes data from a register into the specified address
- Note: the structure is identical to Load



38

Load Example (not x86)

```
LDR r3, [123]
```



39