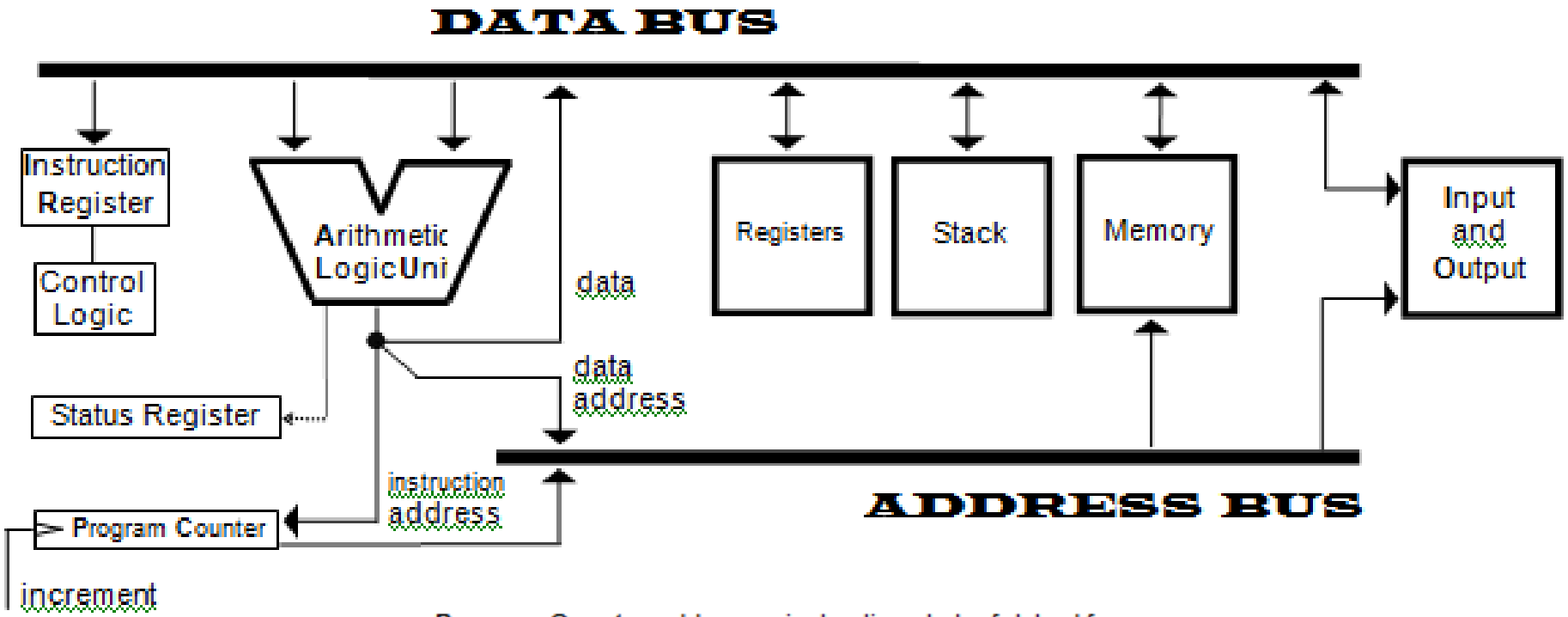
Communications

J. Wunderlich, Ph.D.
Computer and Engineering and Science
Elizabethtown College

Communication within Computer and/or CPU



Program Counter addresses instructions to be fetched from memory

Instruction Register receives fetched instruction

Control Logic creates all routing signals after decoding the fetched instruction

Arithmetic Logic Unit (ALU) performs arithmetic and logical manipulation of data and addresses

Registers (i.e., general purpose registers) store intermediate results of calculations

Status Register holds status flags and condition codes

"Memory" (i.e. "main memory") stores data and instructions

Stack stores addresses (or processor status) for returning from program-calls (or interrupts)

Input/Output ("I/O") often addressed as memory (i.e., memory-mapped I/O)

Communications within Personal Computer

- Match speed of CPU, RAM, and Motherboard **F**ront-**S**ide-**B**us (**FSB**) which connects CPU and RAM
- Socket to plug in CPU (i.e. Intel or AMD)
- **Chip-set** to handle CPU and RAM (and video card and other I/O)
  - **Northbridge** for RAM and video card control, and restricts overclocking
  - **Southbridge** for power, clock, and other I/O control
- **Dual-Channel** (to handle two banks of RAM concurrently)
- Expansion Slots
  - **ISA** (almost obsolete)
  - **PCI**
  - **AGP**
  - **PCIexpess**
  - *ISA, PCI, and AGP use PARALLEL memory-mapped I/O Bus protocol; But, PCIexpress uses a packetizing **SERIAL** protocol like that used for Ethernet TCP/IP*

Communication between PCand Devices

- **DVI** (**D**igital **V**isual **I**nterface)
- **Audio** jacks
- **VGA** video jack
- Old **Mouse** and **Keyboard** DIN 6-pin (now USB)

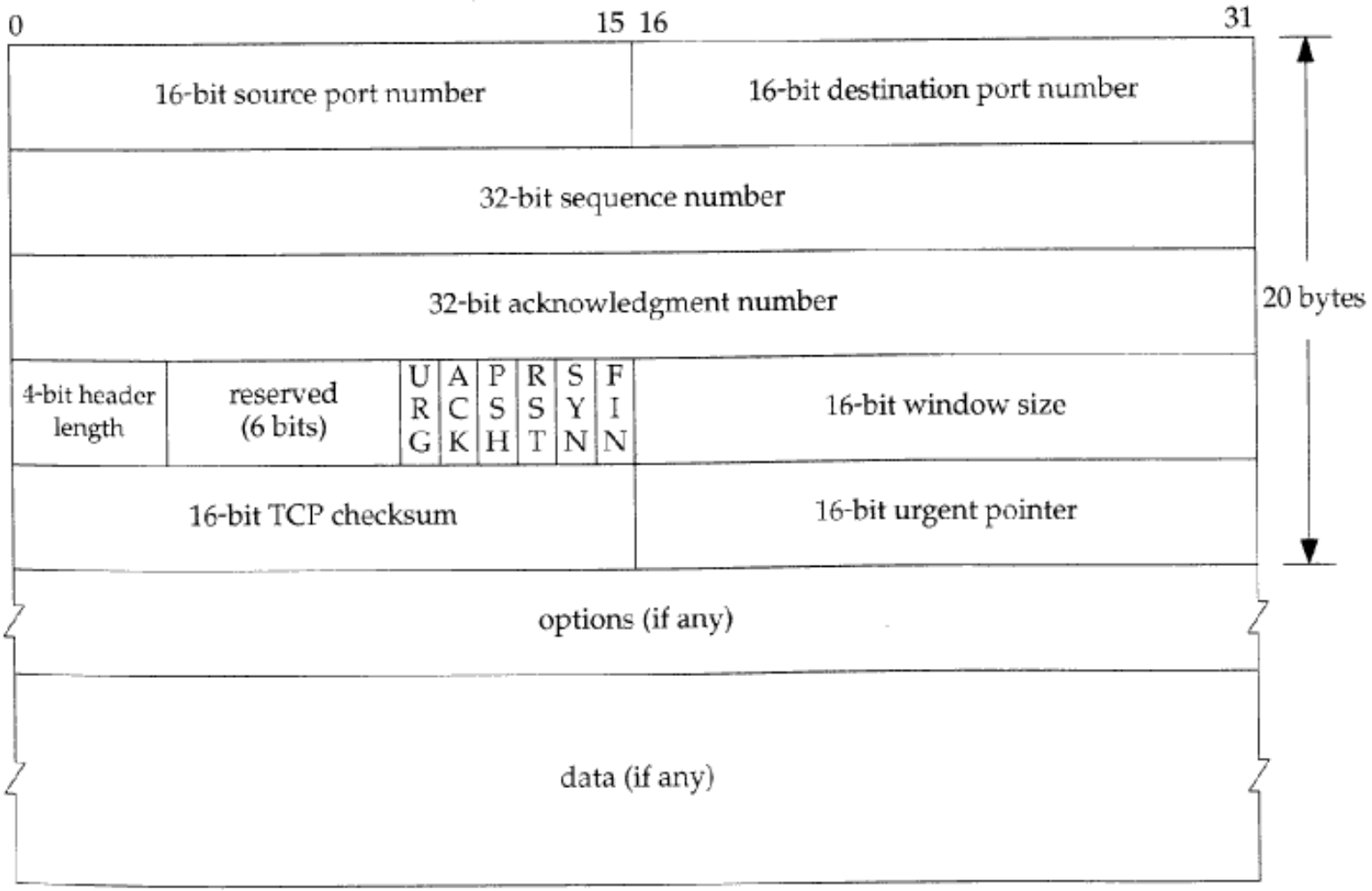
- **Parallel** (multiple **bits** side by side), “LPT1,” “LPT2”
  - Old printer connections
- **Serial** (one **bit** at a time per channel), “Com1,” “Com2”
  - **USB** (**U**niversal **S**erial **B**us)
    - Replaced most Parallel and Serial
    - Up to 127 peripherals simultaneously (including Flash memory cards)
    - Hot insertion and removal
  - **FireWire** for cameras and portable storage
  - **Network** jack “Rj-45” *packetizing **SERIAL** protocol for Ethernet TCP/IP*

DATAGRAM (i.e., a "Packet")

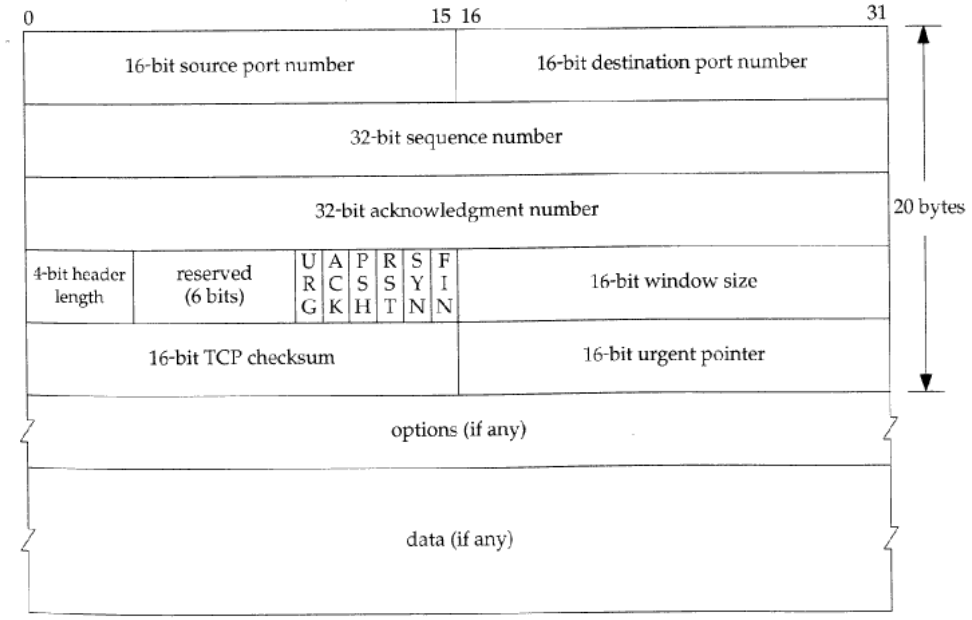
for TCP/IP (Transmission Control Protocol/Internet Protocol)

TCP HEADER

TCP data encapsulated in IP datagram. Figure shows format of TCP header. Its normal size is 20 bytes :



Source: http://www.ssfnet.org/Exchange/tcp/tcpTutorialNotes.html



SrcPort and **DstPort** fields identify source and destination ports. These plus source and destination IP addresses combine to identify each TCP connection.

sequence number identifies byte in data stream from sending TCP to receiving TCP that the first byte of data in this segment represents.

acknowledgement number is next sequence number that sender of acknowledgement expects to receive. i.e., sequence number plus 1 of last successfully received byte of data. This field is valid only if ACK flag is on. Once a connection is established Ack flag is always on.

Acknowledgement, **SequenceNum**, and **AdvertisedWindow** involved in TCP's sliding window algorithm.The Acknowledgement and AdvertisedWindow field carry info about flow of data going in other direction. In TCP's sliding window algorithm receiver advertises a window size to sender using the AdvertisedWindow field. The sender is then limited to having no more than a value of AdvertisedWindow bytes of unacknowledged data at any given time. The receiver sets a suitable value for the AdvertisedWindow based on the amount of memory allocated to the connection for the purpose of buffering data.

header length (in 32-bit words) Required because length of options field is variable.

6-bit Flags field used to relay control info between TCP peers. SYN and Fin flags for establishing and terminating a TCP connection, ACK flag is set any time Acknowledgement field is valid, implying that the receiver should pay attention to it. URG flag signifies this segment contains urgent data. When set, UrgPtr indicates where non-urgent data in this segment begins. PUSH flag signifies sender invoked push operation, which indicates to receiving side of TCP that it should notify the receiving process of this. RESET flag signifies receiver has become confused and so wants to abort connection.

Checksum (FOR ERROR DETECTION) is a mandatory field calculated by sender, then verified by receiver.

Option field is maximum segment size option, called MSS. Each end of connection normally specifies this option on first segment exchanged. It specifies maximum sized segment sender wants to receive.

Data portion of TCP segment (optional, but it's the actual data you are most likely trying to send!) i.e., everything else is communication overhead !!

Wunderbot 4 Wireless Communication
by Jeremy Crouse (advisor: J. Wunderlich)

Robot Navigation

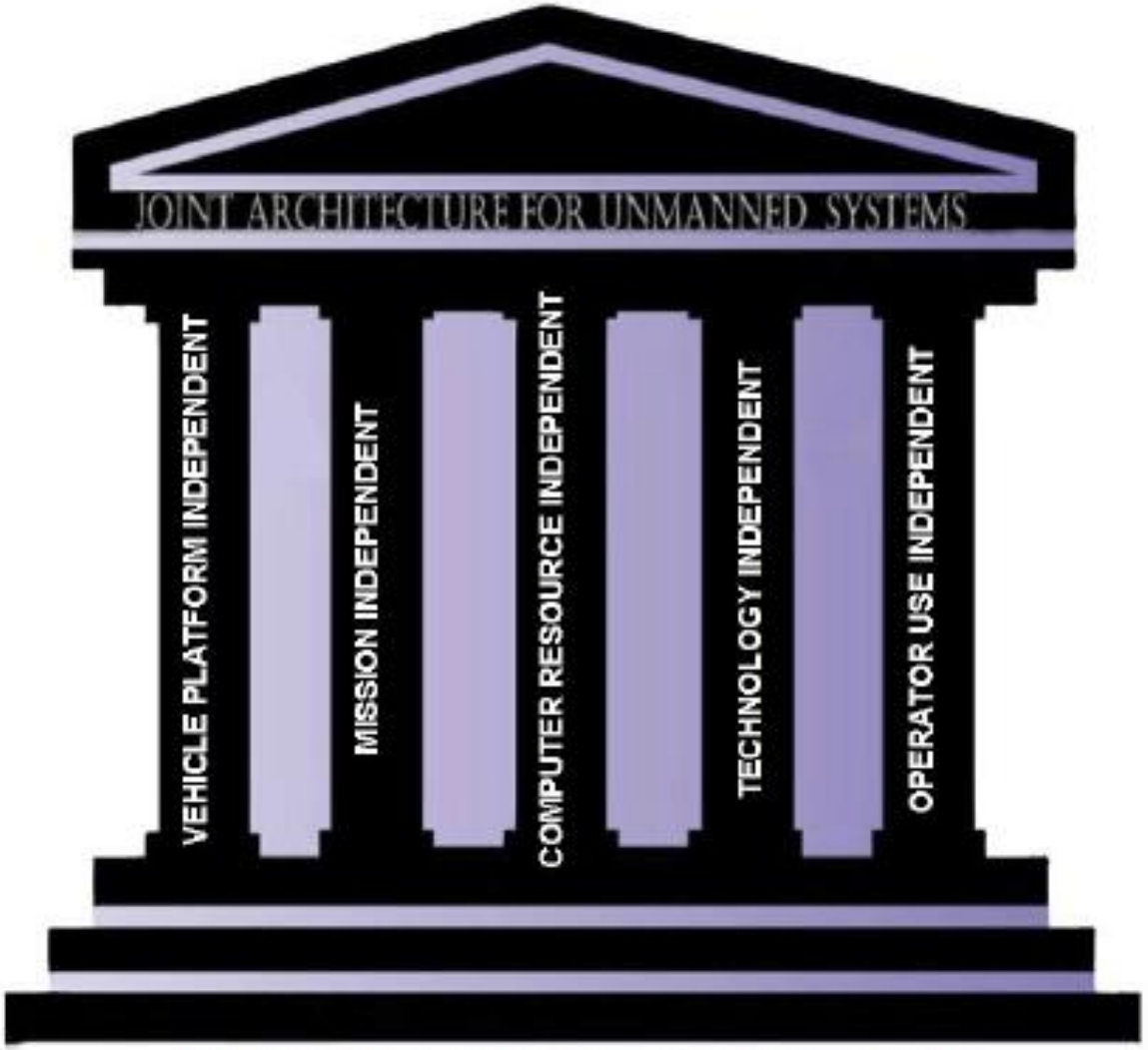


Figure 1: JAUS Independence requirements [3]

Although Wunderbots are fully autonomous, the IGVC awards those who can respond to "JAUS"

Wunderbot 4 was one of only a few to do this (of many Universities)

SOURCE: : Crouse, J. (2008). **The Joint Architecture for Unmanned Systems (JAUS): a subsystem of the wunderbot 4**. *Elizabethtown College research report.*

Wunderbot 4 Wireless **JAUS** Communication
is like TCP/IP , but different. Used for military comm

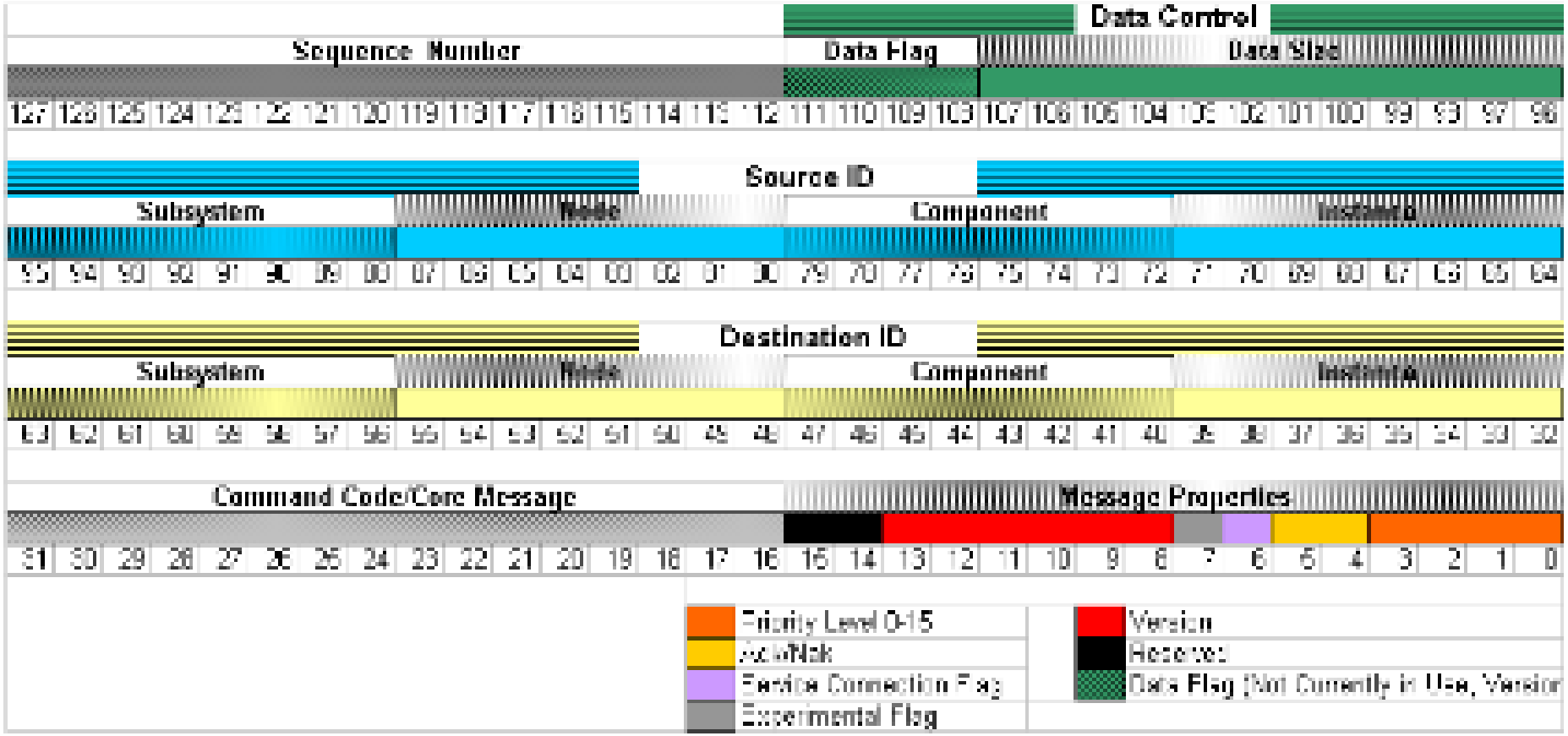


Figure 7: JAUS message header detailed structure [6]

III.JAUS & IGVC

Navigation

Most recent Wunderbot systems:

Wunderbot - Main VI Labview Tutorial
Wunderbot - GPS Subsystem Labview Tutorial
Wunderbot - LADAR Subsystem Labview Tutorial
Wunderbot - JAUS Subsystem Labview Tutorial
Wunderbot - Vision Subsystem Labview Tutorial
Wunderbot - Motor Control Subsystem Labview Tutorial
Wunderbot - Digital Compass Subsystem Labview Tutorial
Wunderbot - MCglobal08 Subsystem Labview Tutorial
NanoLC Robot Simulation

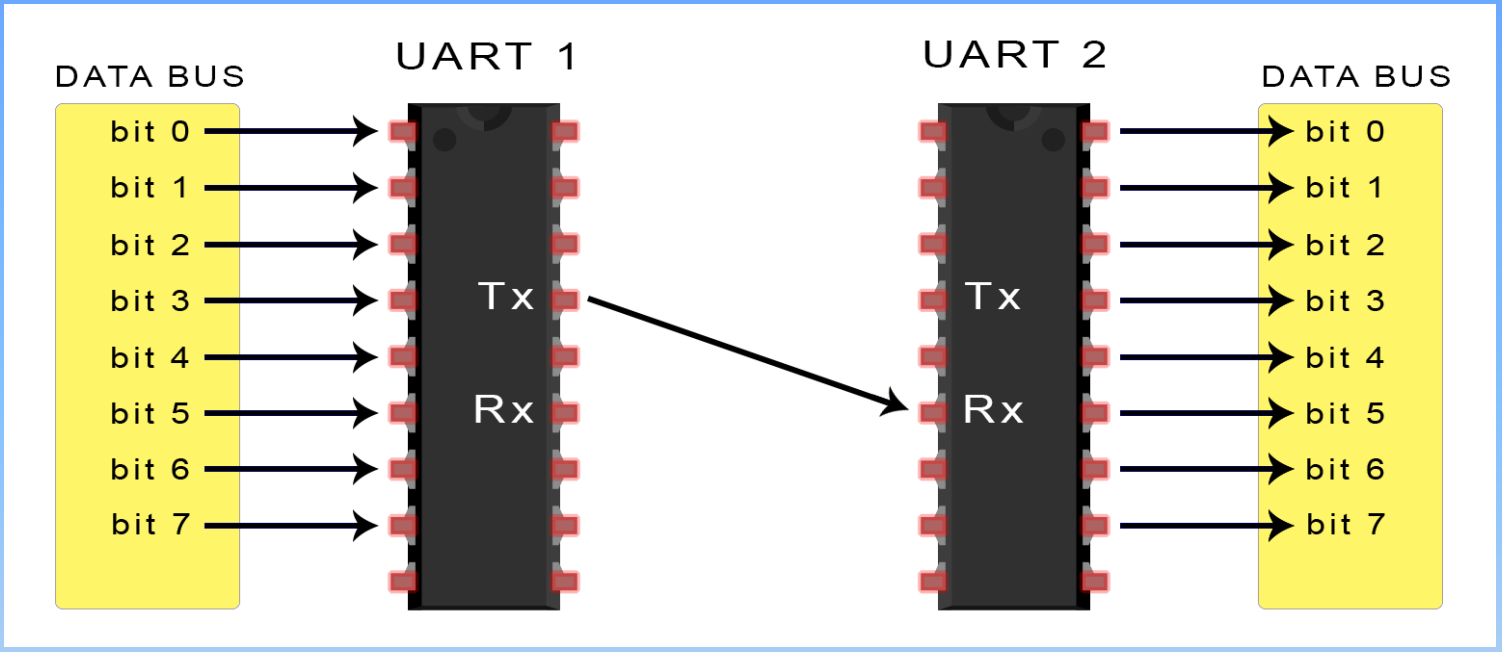
Wunderbot 4

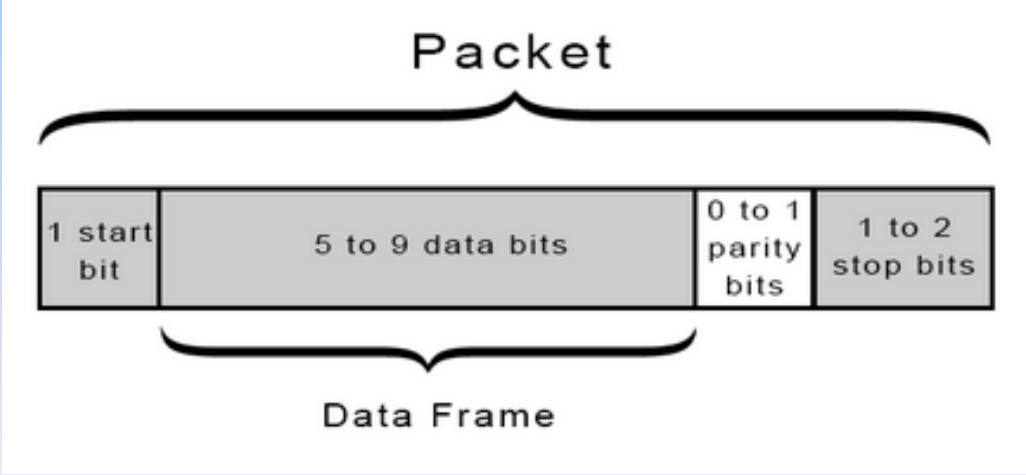
Theory and design:

1. Painter, J. and Wunderlich, J.T. (2008). Wunderbot IV: autonomous robot for international competition. In *Proceedings of the 12th World Multi-Conference on Systemics, Cybernetics and Informatics: WMSCI 2008, Orlando, FL*: (pp. 62-67). And HERE
2. Coleman, D. and Wunderlich, J.T. (2008). O^3: an optimal and opportunistic path planner (with obstacle avoidance) using voronoi polygons. In *Proceedings of IEEE the 10th international Workshop on Advanced Motion Control, Trento, Italy.* vol. 1, (pp. 371-376). IEEE Press.
3. **JAUS wireless packetized communication by Jeremy Crouse**

UART (Universal Asynchronous Receiver Transmitter)

- Translates between Parallel and Serial Communication

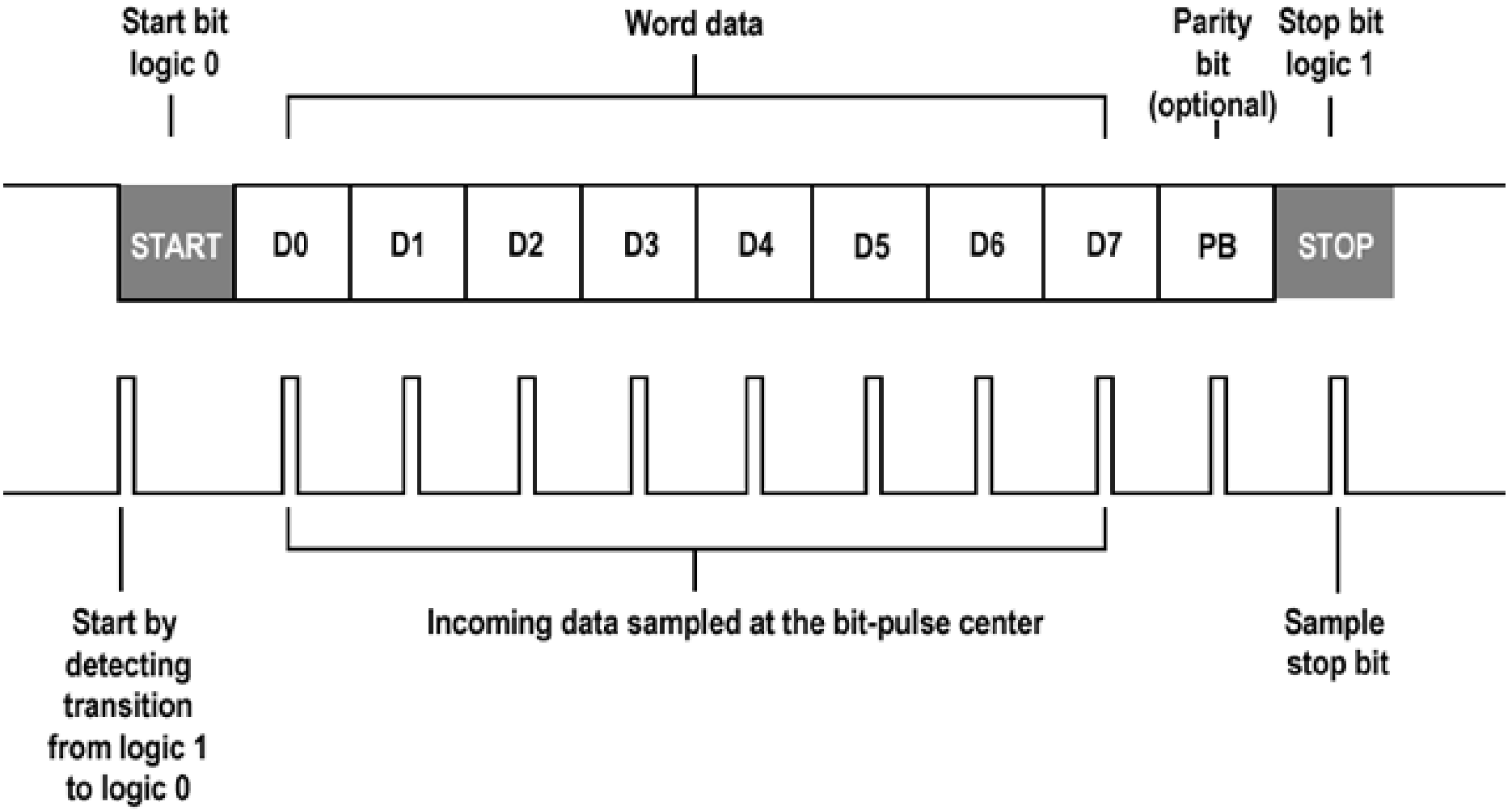




From http://www.circuitbasics.com/basics-uart-communication/

UART (Universal Asynchronous Receiver Transmitter)

- Translates between Parallel and Serial Communication

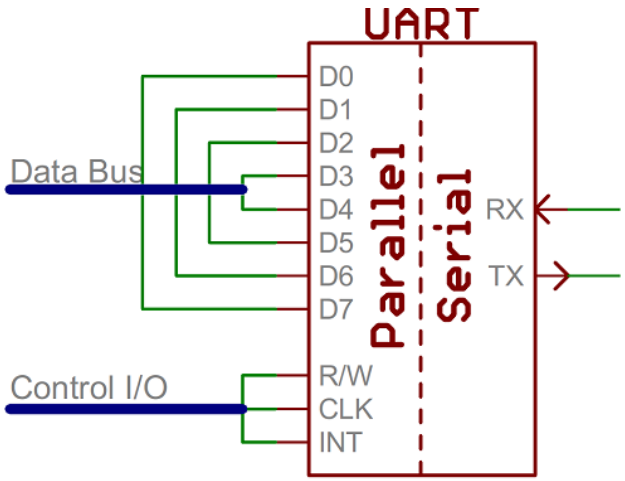


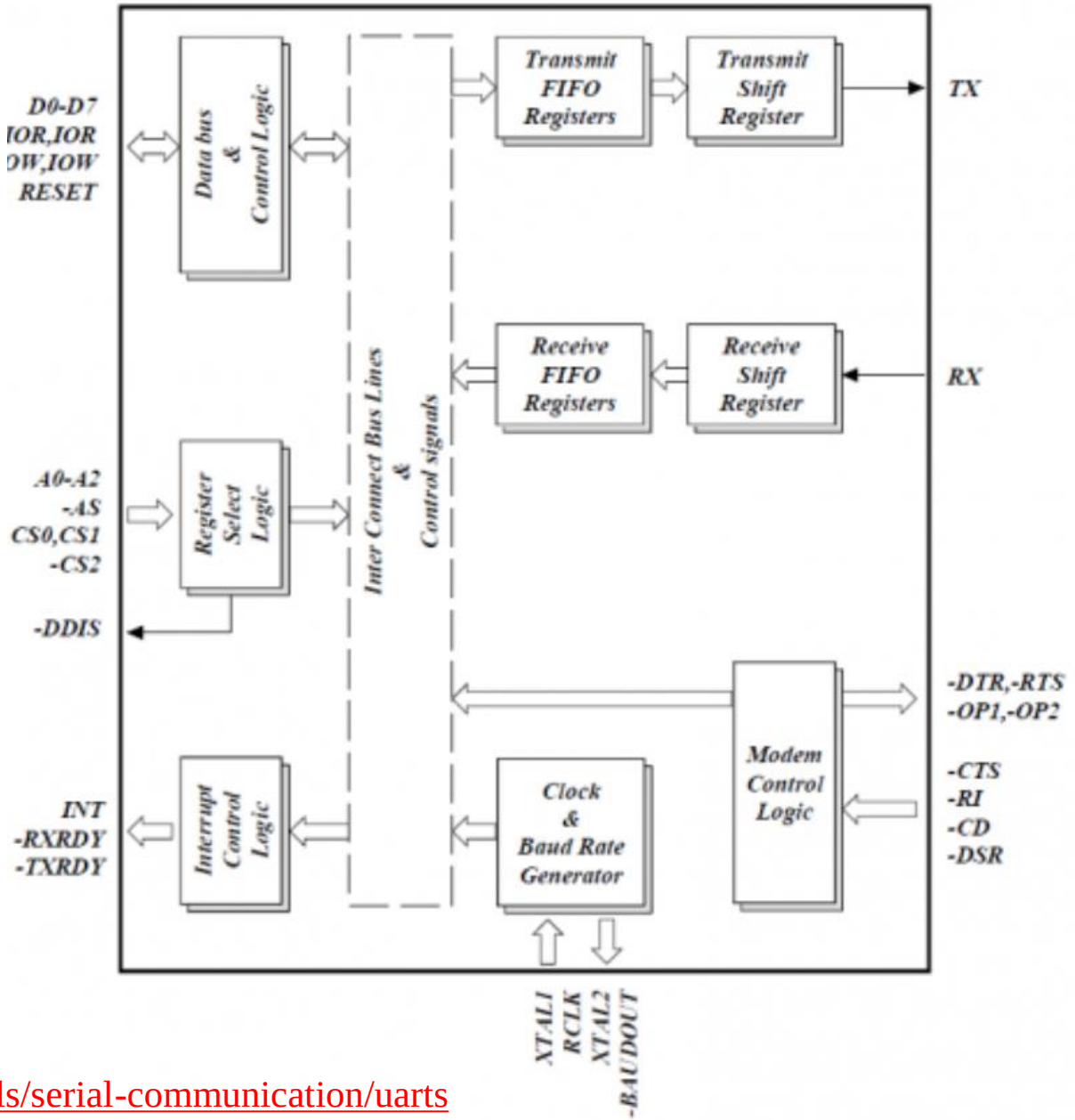
A UART frame

From: https://electricimp.com/docs/resources/uart/

UART (Universal Asynchronous Receiver Transmitter)

- Translates between Parallel and Serial Communication

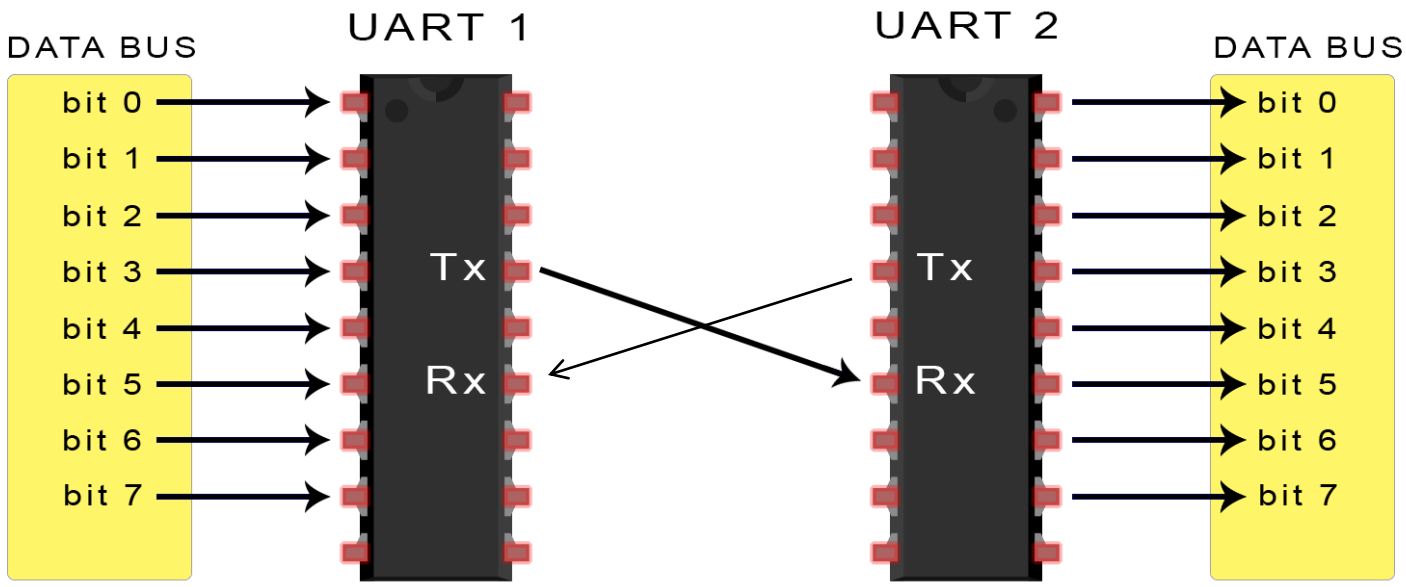




From: https://learn.sparkfun.com/tutorials/serial-communication/uarts

UART (Universal Asynchronous Receiver Transmitter)

- Translates between Parallel and Serial Communication



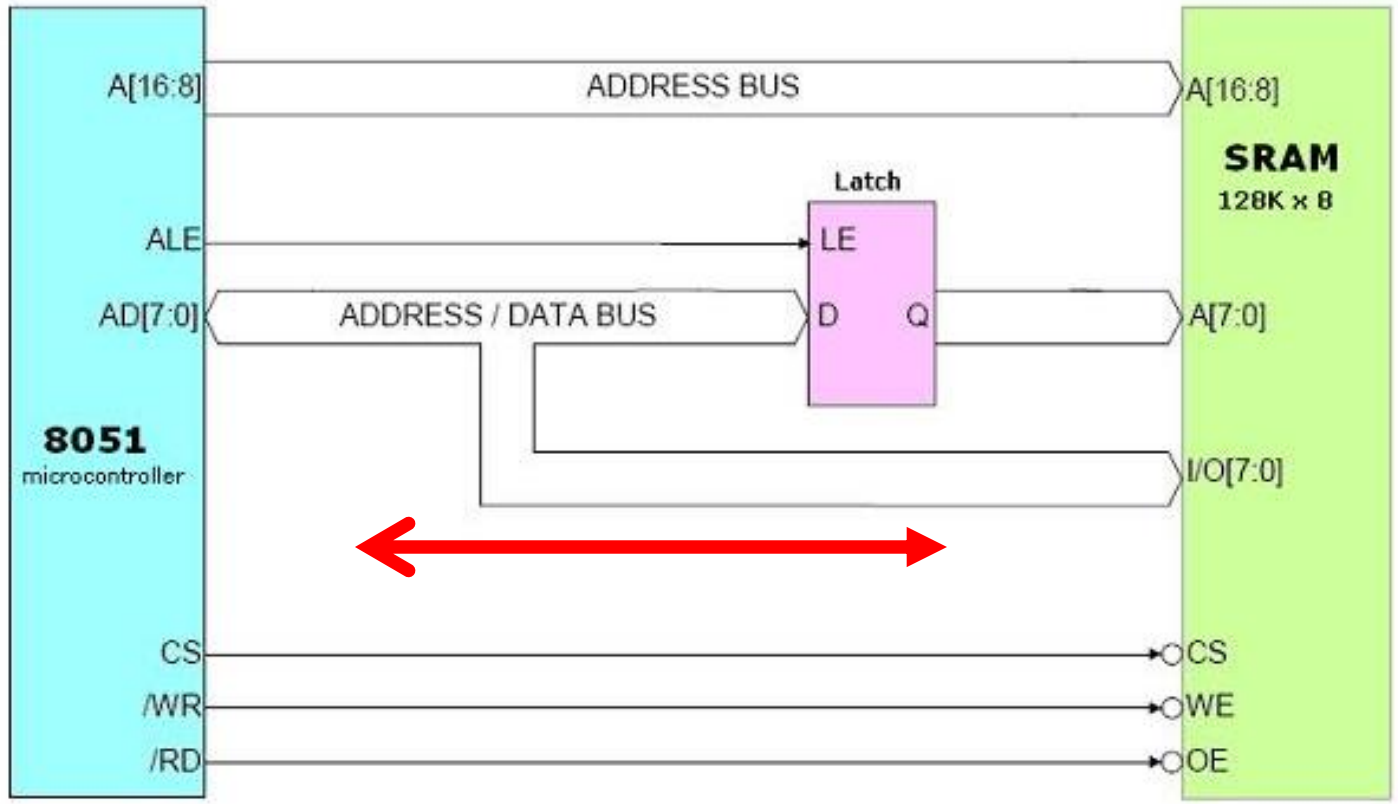
From http://www.circuitbasics.com/basics-uart-communication/

NOTE: This reduces communication wires down to just two

Sometimes the wires (and pins) needed for Communication are reduced by multiplexing pins, like the Address/Data BUS between an Intel 8051 Microcontroller and External RAM . . . . . .

EXTERNAL BIDIRECTIONAL Parallel Address / Data BUS

between an Intel 8051 Microcontroller and External RAM



Wires (and pins) needed for Comm reduced by multiplexing pins, like the EXTERNAL BIDIRECTIONAL Parallel Address/Data BUS between an Intel 8051 Microcontroller and External RAM (if used)

From http://www.eeherald.com/section/design-guide/esmod15.html

EXTERNAL BIDIRECTIONAL SERIAL BUS LAB

Assignment: LAB #4 "Bus Communication between two CircuitTrainers (TTL) and Programmable Logic Controller (PLC)"

COURSE: EGR/CS333 **"DIGITAL DESIGN & INTERFACING"** (Digital Design II, Assembly Language, and Interfacing)

SYLLABUS: http://users.etown.edu/w/wunderjt/syllabi/CS333%20Wunderlich,Joseph.htm

INSTRUCTOR: J. Wunderlich PhD

- Make NanolineLC Programmable Logic Controller (PLC) implement some of the functionality of one of the circuit trainers of Lab #3; specifically, shifting in a nibble; And, shifting out a nibble (can be defined simply as a constant within the PLC), Extra points if you can shift in, **store, and then back out** of PLC. Extra points for dealing with parity of bits coming into, or leaving PLC.
- **Using tri-state buffers and appropriate relays** between devices (Phoenix Contact, Arduino, or others located in lab), and needed control lines communicating within and between devices, create a **BIDIRECTIONAL SINGLE WIRE SERIAL** communication **DATA BUS** shared by the two circuit trainers (with circuits from Lab #3) and the NanolineLC Programmable Logic Controller (PLC). You can have as many control lines as you wish, but only one shared bidirectional serial single-wire data bus.
- Demonstrate **bidirectional** serial communication with parity check between each trainer TTL circuit. (as part of testing, create a data error in the data transmission of your nibble so that the sending and receiving parity don't match).
- Demonstrate **bidirectional** serial communication (parity check optional) between each trainer TTL circuit and the PLC.
- Use high-voltage light bulb controlled by the PLC to communicate to the user each data bit received or transmitted to or from the PLC, Use terminal blocks and wire nuts for safe hi-voltage connections, and keep high voltage wires as removed as possible from TTL and low-voltage controls of the PLC.
- Use your circuit trainer clock generators to see how fast you can communicate **to** the PLC (i.e., if you can communicate as fast as the fastest clock generator available)
- Create a complete simulation in Logisim of everything above including forced error in data transmission, clock generators, and a logic facsimile of what your PLC is actually doing (i.e., translate your PLC logical decisions and actions into equivalent logic gates in Logisim)

IPC (Inter-Processor/Core Communication) bottleneck

- IPC has been a limiting factor in the SCALABILITY of multi-processor/core computing
  - Because of the law of diminishing returns , IPC overhead grows disproportionately to the speedup gained by increasing the number of Processors/Cores
- However the opposite is somewhat true when considering the increase of nodes on the internet
  - Where the routing benefits of decongesting packet traffic mitigates the IPC growth penalties, and in fact, overall network speed increases with the number of nodes, but only up until a limit.
- This methodology has now found its way into CPU design
  - Read ***"Breaking the Multicore Bottleneck" Oct, 2016, IEEE Spectrum***
    http://spectrum.ieee.org/semiconductors/processors/breaking-the-multicore-bottleneck
    - » **This is NOT better than the Ideal Case but *may* be an improvement on the upper bound of Amdahl's Law**

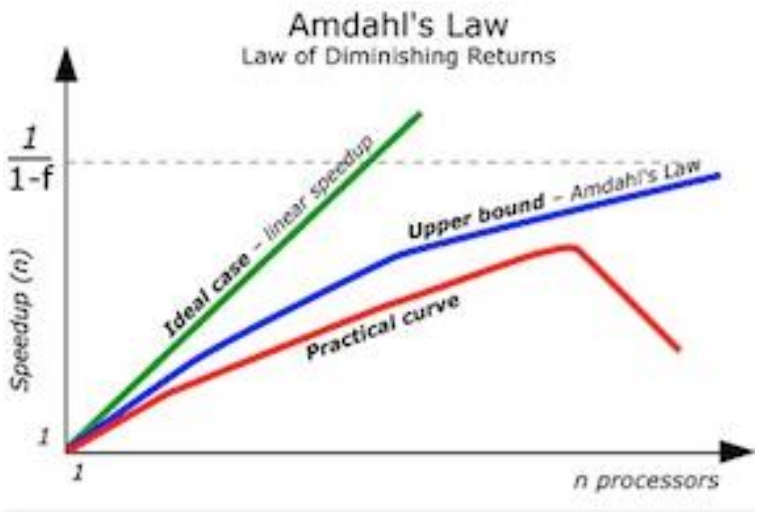


Image from: https://www.javacodegeeks.com/wp-content/uploads/2013/02/amdahl.jpg