Embedded Computing Architecture – ISAs – Assembly Language

Baris Aksanli
Department of Electrical and Computer Engineering
San Diego State University

Outline

- Microcontroller vs. microprocessor
- Von Neumann vs. Harvard architectures
- AVR examples
- Instruction sets
- Assembly instructions
- Character representations
- ASCII

Microprocessor vs. Microcontroller

- Microcontroller chip includes
  - Central processor
  - Program memory
  - Data memory
  - I/O
  - Highly integrated, low cost
- Microprocessor chip includes
  - Central processor only
  - Highest performance
  - Highest cost

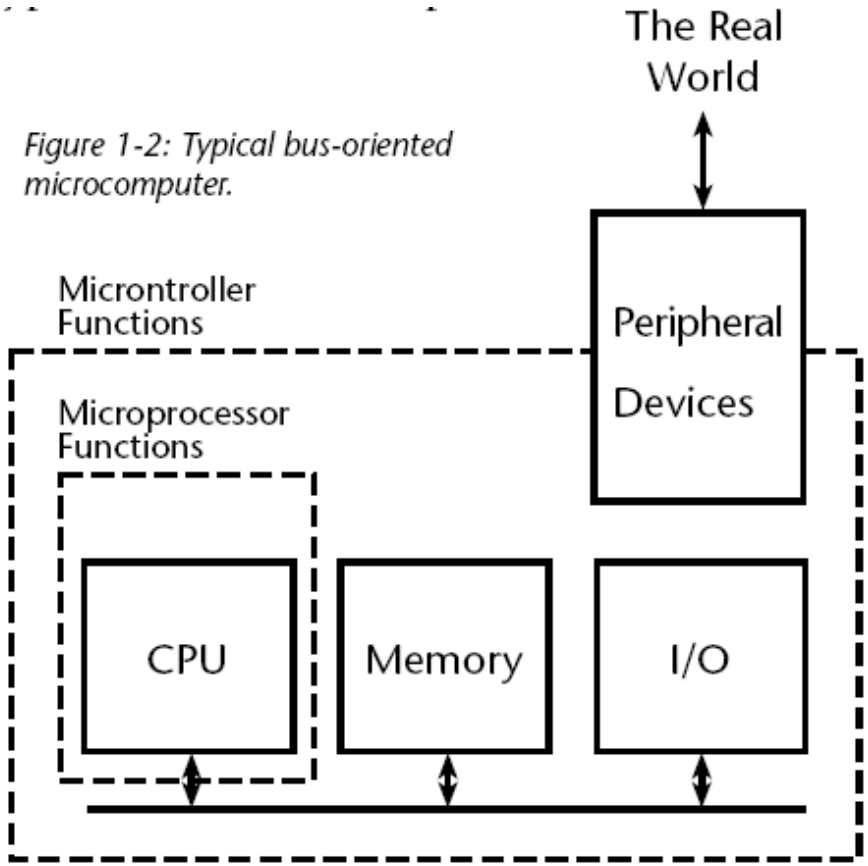


Figure 1-2: Typical bus-oriented microcomputer.

Microcontroller chips

- Advantages:
  - Fewer chips required
  - Lower cost and smaller
  - Lower power (can be power optimized)
  - Fewer connections
  - More I/O pins
  - Higher reliability
- Disadvantages:
  - Reduced flexibility
  - Expansion is limited
  - Limited performance
  - Limited I/O
  - Design compromised to fit everything on one chip

What is AVR ?

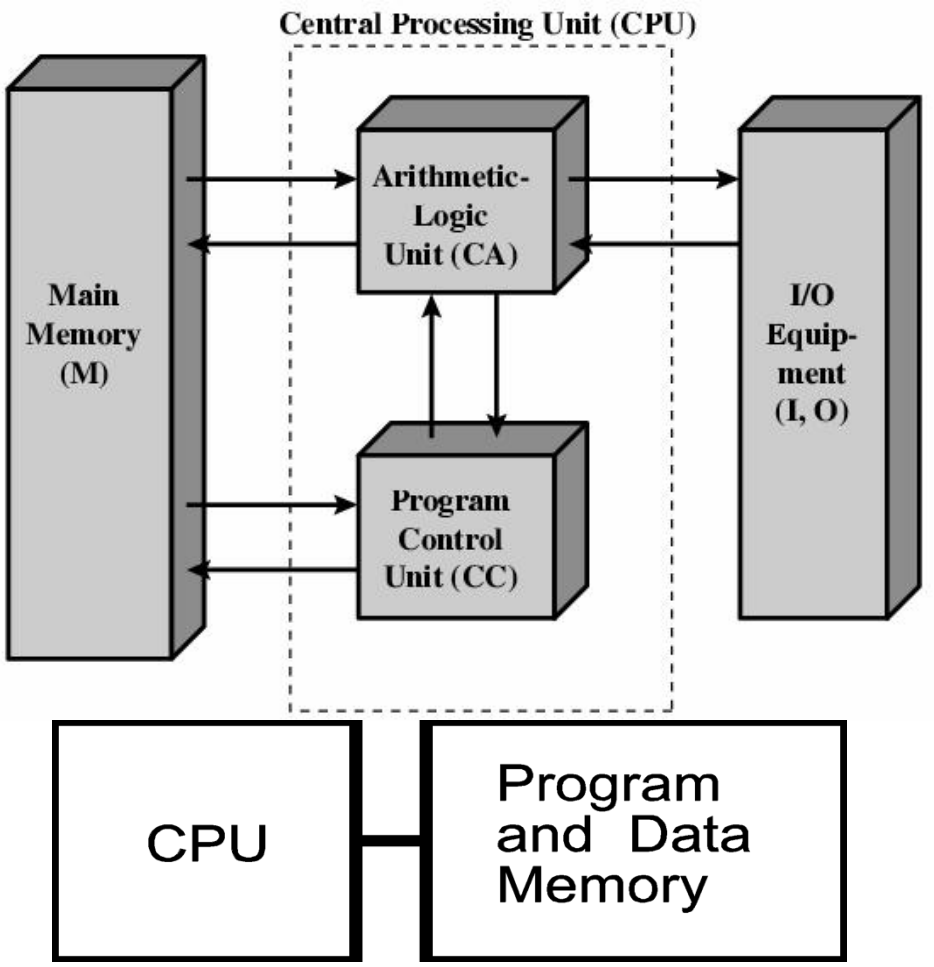
- Modified Harvard architecture 8-bit RISC single chip microcontroller
- Complete System-on-a-chip
  - On Board Memory (FLASH, SRAM & EEPROM)
  - On Board Peripherals
- Advanced (for 8 bit processors) technology
- Developed by Atmel in 1996
- First In-house CPU design by Atmel

AVR Family

- 8 Bit tinyAVR
  - Small package – as small as 6 pins
- 8 Bit megaAVR
  - Wide variety of configurations and packages
- 8 / 16 Bit AVR XMEGA
  - Second Generation Technology
- 32 Bit AVR UC3
  - Higher computational throughput

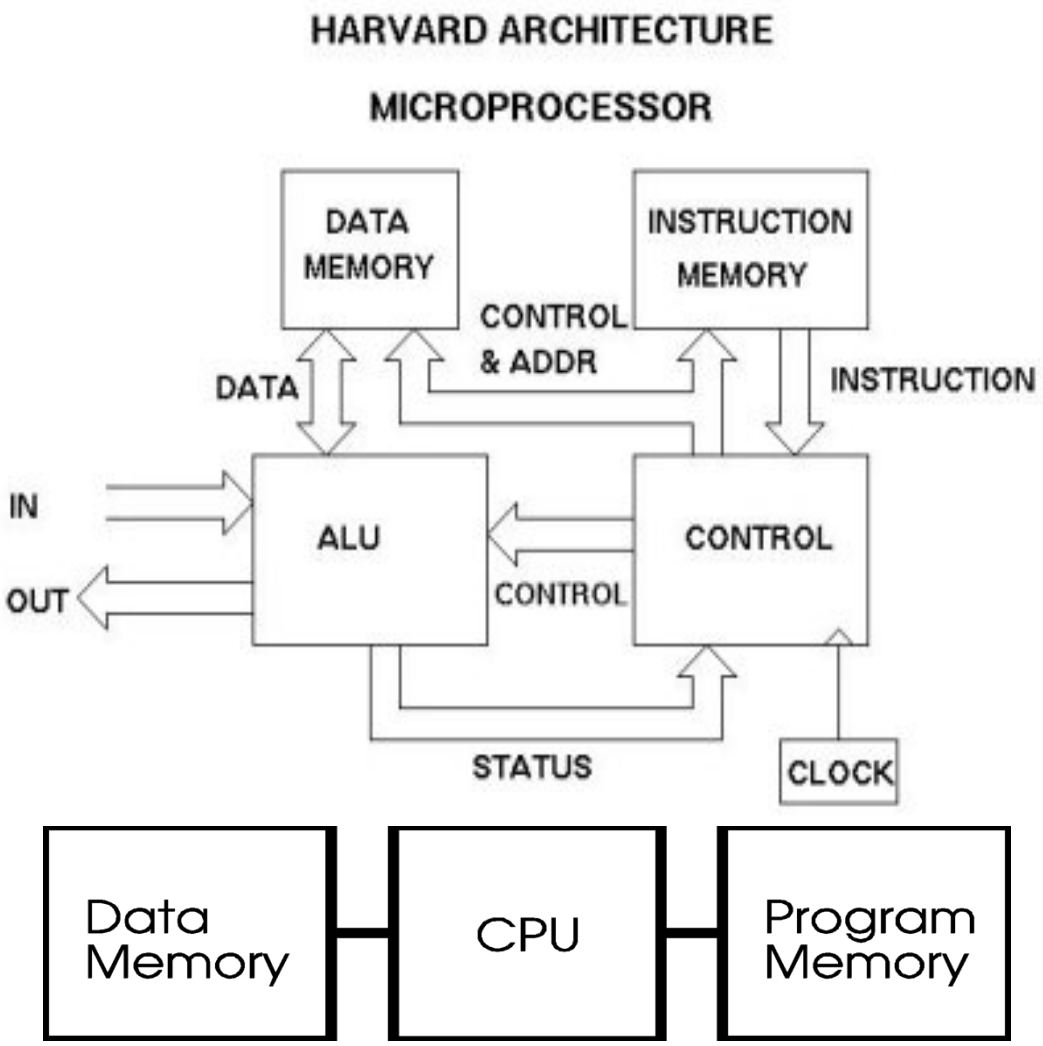
Von Neumann Architecture

- Single memory for:
  - Programs
  - Data
- Familiar
- Most flexible
- Used in PCs
- Speed bottleneck:
  - Memory Interface



Harvard Architecture

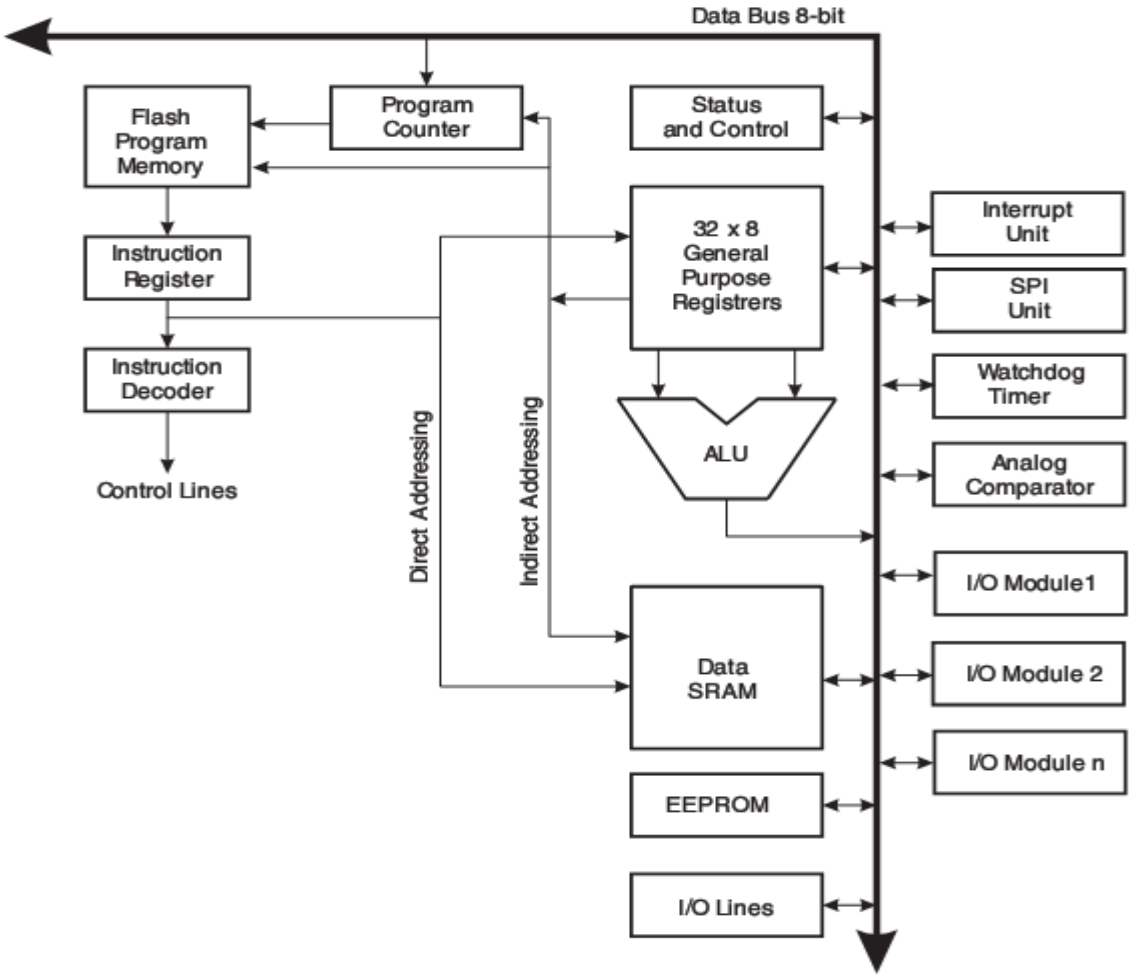
- Separate memory for:
  - Programs
  - Data
- Advantages:
  - Faster
  - Overlap transfers
    - Instruction Fetch
    - Data Transfer
  - Can't execute data!



Harvard Architecture Advantages

- Separate instruction and data paths
- Simultaneous accesses to instructions & data
- Hardware can be optimized for access type and bus width

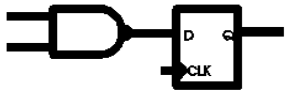
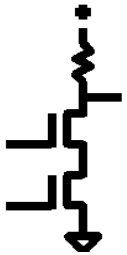
AVR Architecture



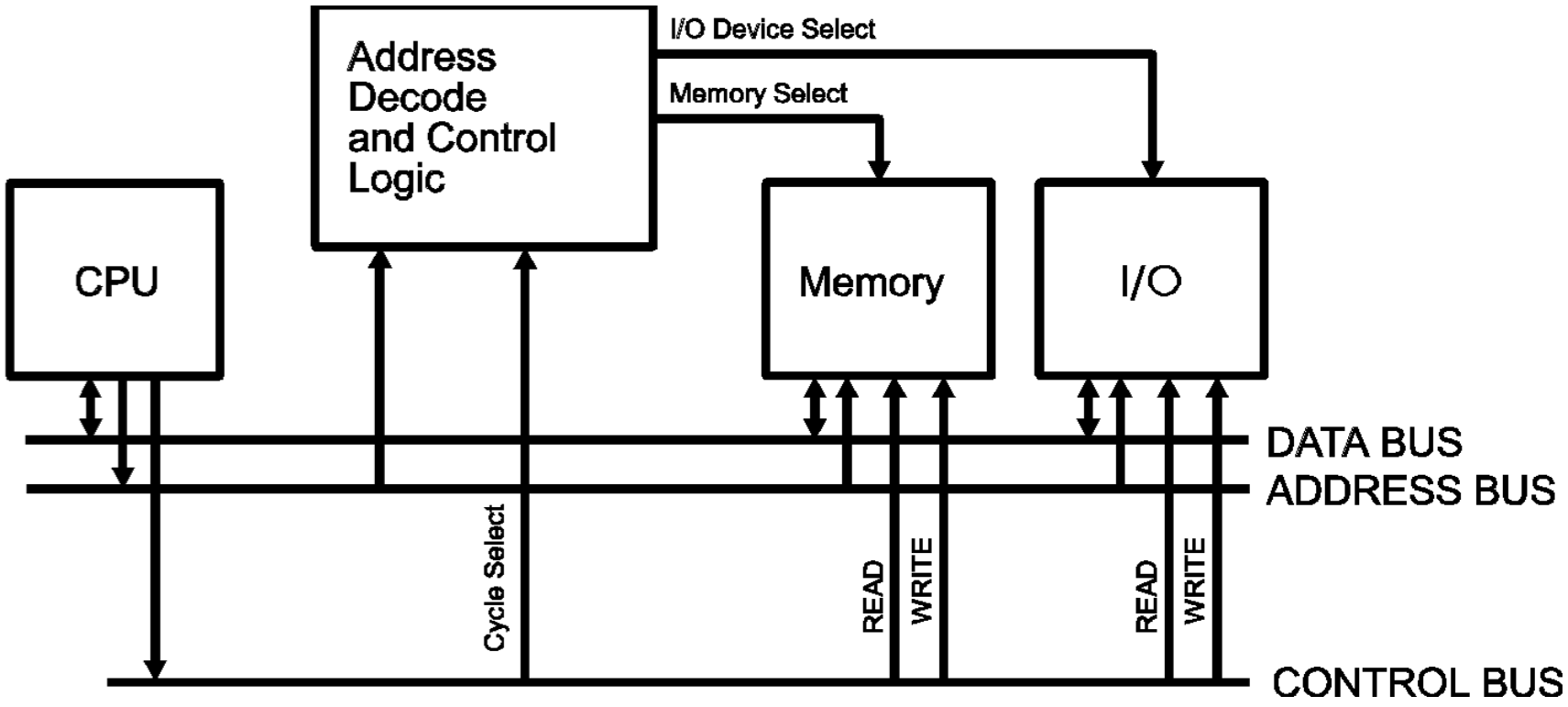
Modified Harvard Architecture

- Special instructions can access data from program space
- Data memory is more expensive than program memory
- Don't waste data memory for non-volatile data

Hierarchy of Computer Design

High Level	Sum := Sum + 1
Assembly	MOV BX,SUM INC (BX)
Machine	1101010100001100 0010001101110101 1111100011001101
Register Transfer	Fetch Instruction, Increment PC, Load ALU with SUM ...
Gate	
Circuit	

Typical Block Diagram



Computer Architecture

- ISA == Instruction Set Architecture
- Common ISAs:
  - CISC, Complex: x86
  - RISC, Reduced Instruction Set:
    - 8 bit: AVR
    - 32 bit: ARM, MIPS
  - WISC: PIC, 8051…
    - WISC == Writeable instruction set computer
  - VLIW = very long instruction word

What is RISC?

- Reduced Instruction Set Computer
- As compared to Complex Instruction Set Computers, i.e. x86
- Assumption: simpler instructions execute faster
- Optimized most used instructions
- RISC machines: AVR, ARM, PowerPC, SPARC
- Became popular in mid 1990s

Characteristics of RISC Processors

- Faster clock rates
- Single cycle instructions (20 MIPS @ 20 MHz)
- Better compiler optimization
- Typically no divide instruction in core

AVR: ALU – Arithmetic Logic Unit

- Directly connected to all 32 general purpose registers
- Operations between registers executed within a single clock cycle
- Supports arithmetic, logic and bit functions
- On-chip 2-cycle Multiplier

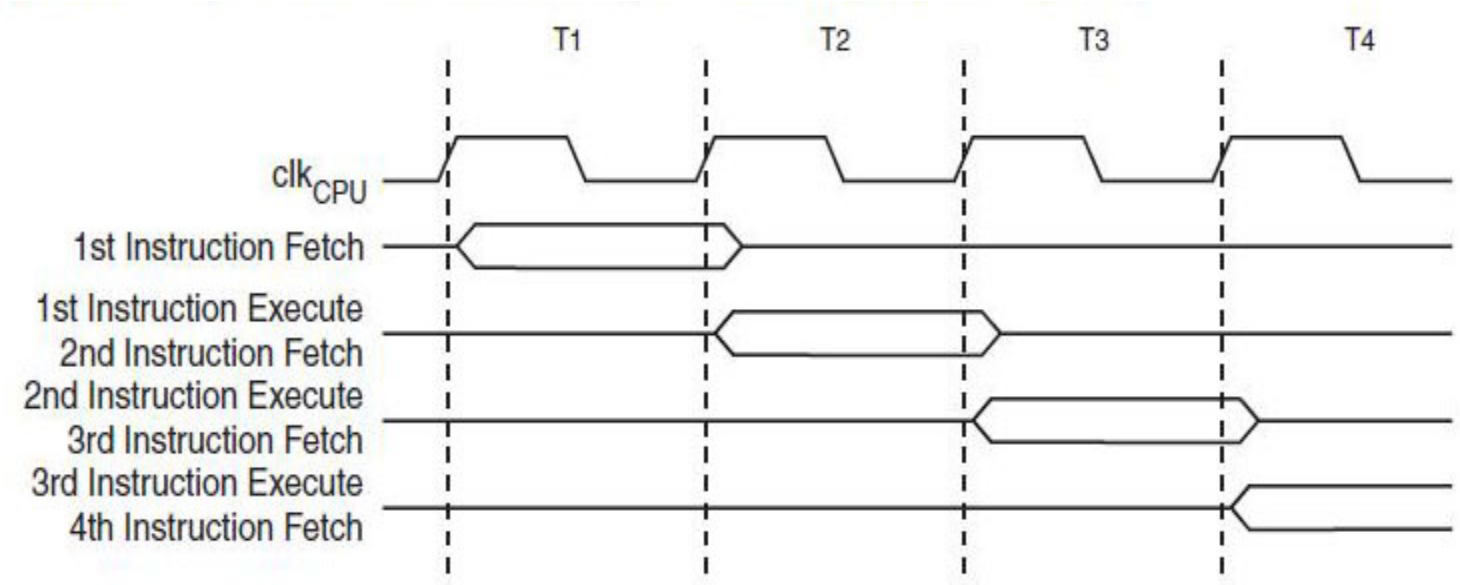
AVR: Instruction Set

- 131 instructions
  - Arithmetic & Logic
  - Branch
  - Bit set/clear/test
  - Data transfer
  - MCU control

AVR: Instruction Timing

- Register ↔ register in 1 cycle
- Register ↔ memory in 2 cycles
- Branch instruction 1-2 cycles
- Subroutine call & return 3-5 cycles
- Some operations may take longer for external memory

AVR: Pipelined Execution



Math Instructions

- Basic instructions:
  - INC, DEC
  - ADD, ADDC, SUBB
- Multi-byte/word Math for 8 bit CPU
  - Byte by byte: Use carry and overflow
  - Software has to do it one byte at a time!
  - (0x1FFF)+(0x0201) = (0x2200)
  - Is this a signed or unsigned example?

Data Path: Arithmetic 1

- INC
  - Increment
  - operands: registers, memory, +??
- DEC
  - Decrement
  - operands: registers, memory, +??

Data Path: Arithmetic 2

- ADD a b
  - Add a to b
  - Set carry bit if carry result
  - No dependence on carry bit in
- ADC a b
  - Add a + b *plus carry bit*
  - Set carry bit if carry result
  - Result depends on carry bit in

MOV Instructions

- MOV destination, source?
- MOV source, destination?
- It depends, even with the same ISA…

Load/Store Instructions

- Load register with memory contents
  - Load register from memory address
- Store register at memory location
  - Store register at memory address
- x86 MOV instruction is really Load/Store

MOV Instructions

- Move between different code spaces
  - Harvard architecture requires special moves
  - Code memory is usually non-volatile
    - But it is also used for lookup tables
  - Data memory is usually volatile
    - SRAM, DRAM
  - Secondary memory:
    - Solid state: Flash, EEPROM,
    - Rotating disk: magnetic, optical

Control Transfer

- JMP/Branch addr
  - Go to address
- Conditional
- JZ jump if zero
  - if Z status bit = 0
- JNZ jump if NOT zero
  - if Z status bit = 0
- Same for Carry, Sign…

- CALL/BL addr
  - Save PC on stack
  - Go to subroutine address
- RET return from sub
  - Pop PC from stack
- Special: RETI
  - Return from ISR
  - Allows for Interrupts
  - "Hardware" Call

Logical and Bit Operations

- AND – Used to mask/clear specific bits
- OR – Used to set specific bits
- XOR – Used to toggle specific bits
- Typical Instructions (bitwise and bit oriented)
  - ANL, ORL, XRL, CLR, SET, CPL
- C language equivalents
  - & bitwise AND, | bitwise OR, ^ bitwise XOR
  - Don't confuse with logical operations: &&, ||

Character Representations

- ASCII
- ISO Latin 1
- Unicode
- Baudot
- EBCDIC

ASCII

- American Standard Code for Information Interchange
- ***7 bit code:*** Values 128..255 are undefined!
- Used extensively with serial communications
- Formally defined by ANSI X3.4.
- Subset of ISO Latin 1, which also defines characters for European languages.
- Other extended codes: 8+bit supersets of ASCII

ASCII Printable Characters

- Digits, Alphabet, Specials
  - Digits are 30h..39h
  - Alphabet
    - 'A'..'Z' are 41h..5Ah
    - 'a'..'z' are 61h..7Ah
  - Specials are '+', <space>, '>', <bell>
- See ASCII tables:
  - https://en.wikipedia.org/wiki/ASCII

ASCII Null, Newline

- Null or \o == oxoo
- Newline \n or End-of-line (EOL)
- May be single control character: Unix uses oxA (line feed)
- May be multiple control characters: MS uses oxD, oxA (carriage return+line feed)
- Transmitter and receiver must agree
- Unix vs. MS

Other Data Encoding

- BCD (Binary Coded Decimal)
- Fixed point
- Floating point
- Error Detection and Correction Coding
  - Parity
  - Hamming Codes
  - Block Error Coding