AVR Programming I

Tools : IDE

- Integrated Development Environment
  - Chip/manufacture specific
  - Editing
  - Compilation and linking
  - Debugging (software and/or hardware)
  - Uploading
  - Documentation
- Otherwise
  - Text editor
  - Command line compilation
  - Debug as a mind game

AVR Studio

- Coding and compilation
- Debugging
  - Code memory view
  - Data memory view
  - Register view
  - I/O view
- Documentation
  - Assembly language help
  - Hardware and software help

Assembly vs. C

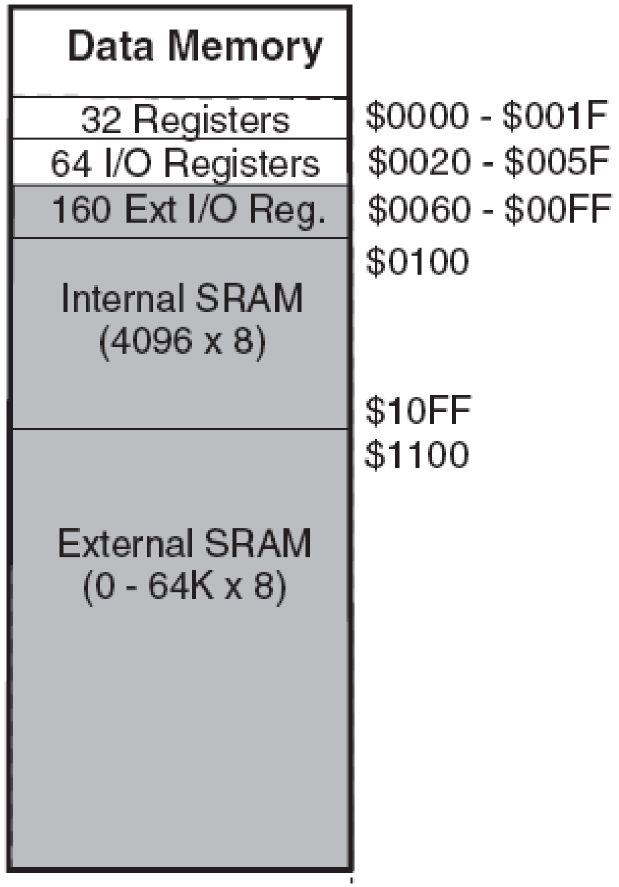
- Difference
  - No variable
  - Any value is either a constant or in a register or at a memory address
  - Any function is a code address
  - Hardware-specific routines must be followed
- Relation via compiler
  - Variable
  - Function

Assembly

- Hardware-specific
  - Register structure
  - I/O structure
  - Addressing
  - Code memory layout
  - Data memory layout
  - Key registers
- Language-specific
  - Directives
  - Expressions
  - ISA

Register Structure

- Also for I/O structure
- 32 general registers
  - Mem address : 0x00-0x1F
- 64 I/O registers
  - Mem address : 0x20-0x5F
  - I/O address : 0x00-0x3F
- 160 Extended I/O registers
  - Mem address : 0x60-0xFF

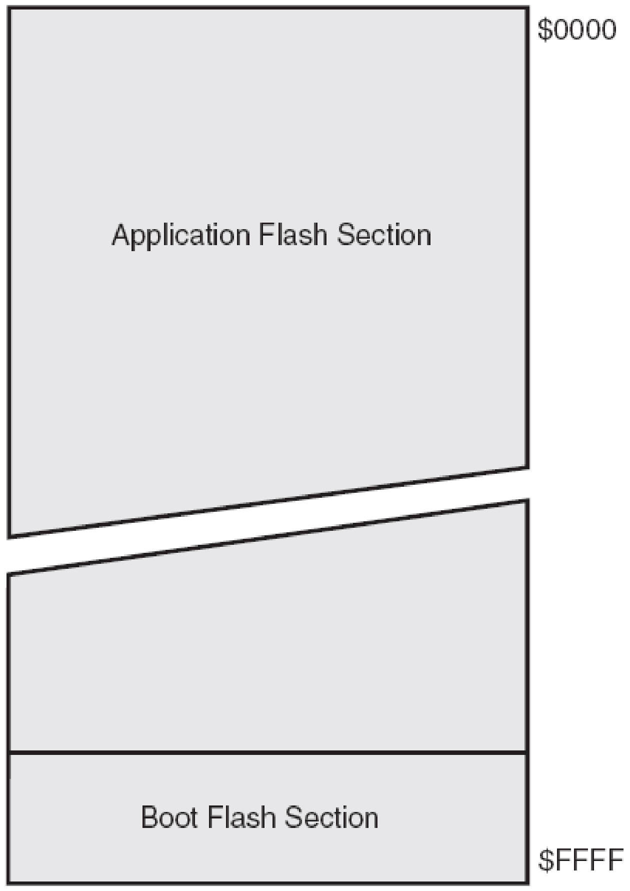


Data Memory Layout

- 4Kx1 Internal SRAM
  - Mem address : 0x100-0x10FF
- 64Kx1 Extended SRAM
  - Mem address : 0x1100-0xFFFF
- Layout
  - Structured by the application
  - BSS for global variables
  - Heap for dynamic allocated variables
  - Stack for local variables

Code Memory Layout

- Boot flash section
  - For boot loader
- Application flash section
  - Interrupt vector
  - Configuration data
    - Peripheral initialization
    - Data memory initialization
  - Application code



Key Registers

- Status register (SREG) : 1 byte
  - I/O address 0x3F, mem address 0x5F
- Stack pointer (SP) : 2 bytes
  - I/O address 0x3D-3E, mem address 0x5D-5E
- X register : R27:R26
- Y register : R29:R28
- Z register : R31:R30
- R0 : temporary register
- R1 : zero register

Assembly Code

- Mnemonic instructions
- Labels
  - [label:] directive [operands] [Comment]
  - [label:] instruction [operands] [Comment]
- Comments
  - ; [Text]
- Directives
- Expressions and functions

Directives

- Directives are NOT opcodes
- Directives are used to
  - Adjust the location of the program in memory
  - Define macros
  - Set data in memory
  - ...
- Directives are used for
  - Convenience
  - Compilation

Directives

- Code memory
  - .CSEG : define the start of a code Segment
  - .DB : define constant byte(s)
  - .DW : define constant word(s)
- Data memory
  - .DSEG : define the start of a data Segment
  - .BYTE : reserve byte(s) to a variable
- Others
  - .SET : set a symbol equal to an expression
  - .DEF : set a symbolic name on a register

Expressions and Functions

- Expressions and functions are constant and evaluated before compilation.
- Expressions can consist of operands, operators and functions.
- Functions
  - LOW(expression): the low byte
  - HIGH(expression): the second byte
  - EXP2(expression): 2 to the power
  - LOG2(expression): the integer part of log2

Example: case9.directive

- Move configuration data from code memory to a table in data memory
  - How to use labels, comments, directives, expressions, and functions.
  - How to watch code and data memory
  - How to watch registers
  - How to debug code