\[ \mu C = \text{CPU} + \text{Memory} + \text{Hardware Modules} \]
How is Software connected to Hardware?
Control registers

Control registers are sets of flip flops which are not only connected for reading and writing: their inputs or outputs are wired into other circuits.
How does the CPU interact with the control registers?

Two methods:

**I/O Instructions**
- Instruction set of processor has special I/O commands
- I/O commands control I/O port registers

**Memory Mapped I/O**
- I/O registers have addresses in reserved memory space
- Memory access

In some CPUs a mix of both methods is used.
How does the CPU interact with the control registers?

Two methods:

**I/O Instructions**
- Instruction set of processor has special I/O commands
- I/O commands control I/O port registers

**Memory Mapped I/O**
- I/O registers have addresses in reserved memory space
- Memory access

In some CPUs a mix of both methods is used.
Special Instructions vs. Memory mapping

- Size of address space
- Convenience of access
  - different addressing modes
- Size of instruction set
  - Instruction bits are precious
Special Instructions vs. Memory mapping

- Size of address space
- Convenience of access
  - different addressing modes
- Size of instruction set
  - Instruction bits are precious
- Cache complications!
  - Input registers will change without instructions from the CPU
The C programming language assumes a single address space (von Neumann architecture). Workaround: the intended address space is indicated to the linker by a specific big offset.
AT90USB1286: Address Spaces

Program Memory: 0x00000 → 0x1FFFF
EEPROM: 0x0000 → 0x0FFF

The C programming language assumes a single address space (von Neumann architecture). Workaround: the intended address space is indicated to the linker by a specific big offset.
All hardware modules on the microcontroller are configured by writing to I/O registers.

All communication to and most communication from these modules is facilitated by reading and writing I/O registers.

Note that even the general purpose registers are mapped into memory address space.
All hardware modules on the microcontroller are configured by writing to I/O registers. All communication to and most communication from these modules is facilitated by reading and writing I/O registers.

Note that even the general purpose registers are mapped into memory address space.
- PORTF is mapped to a unique memory address
- DDRF is mapped to a unique memory address
- PINF is mapped to a unique memory address
io.h will include—over several indirections—a set of C-preprocessor `#define` statements that define constant labels (e.g., DDRF) for the correct address of that register in the target chip.
Manipulating register bits

- 8-bit registers $\rightarrow$ unsigned $\Rightarrow$ Type: uint8_t
  - stdint.h: typedef unsigned char uint8_t

- Setting a bit (=1):
  - DDRB |= _BV(PB7)

- Clearing a bit (=0):
  - DDRB &= ~_BV(PB7)
avr-libc: avr/io.h

#include <avr/sfr_defs.h>

#if defined (_AVR_AT94K__)
  # include <avr/ioat94k.h>
#elif defined (_AVR_AT43USB320__)
  # include <avr/io43u32x.h>
#elif defined (_AVR_AT43USB355__)
  # include <avr/io43u35x.h>
#elif defined (_AVR_AT76C711__)
  # include <avr/io76c711.h>
#elif defined (_AVR_AT86RF401__)
  # include <avr/io86r401.h>
#elif defined (_AVR_AT90PWM1__)
  # include <avr/io90PWM1.h>
#elif defined (_AVR_AT90PWM2__)
  # include <avr/io90PWM2.h>
#endif

#define _BV(bit) (1 << (bit))

avr/sfr_defs.h

avr/io.h; see also notes from
avr/sfr_defs.h and avr/portpins.h.