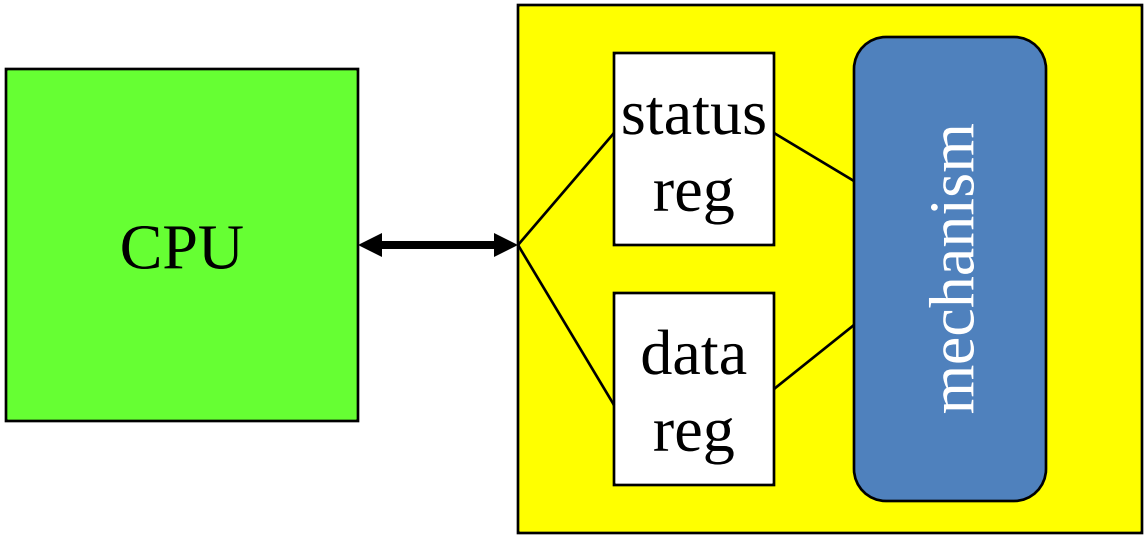
Chapter 3 - CPUs

- Beyond the instruction set:
  - Input and output.
    - Busy-Wait and Interrupt-Driven
  - Supervisor mode, exceptions, traps.
  - Input/output on the LPC2292
    - I/O ports
    - Busy-wait programming
    - Interrupt processing
    - UART

I/O devices

- Often includes some non-digital component.
- Typical digital interface to CPU:

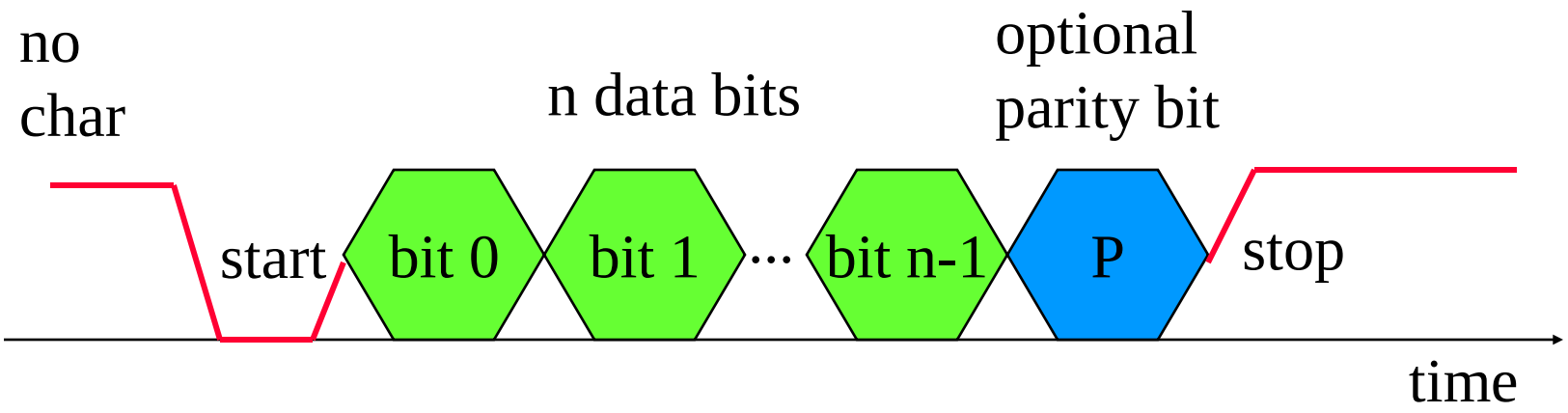


Example: LPC2292 UART

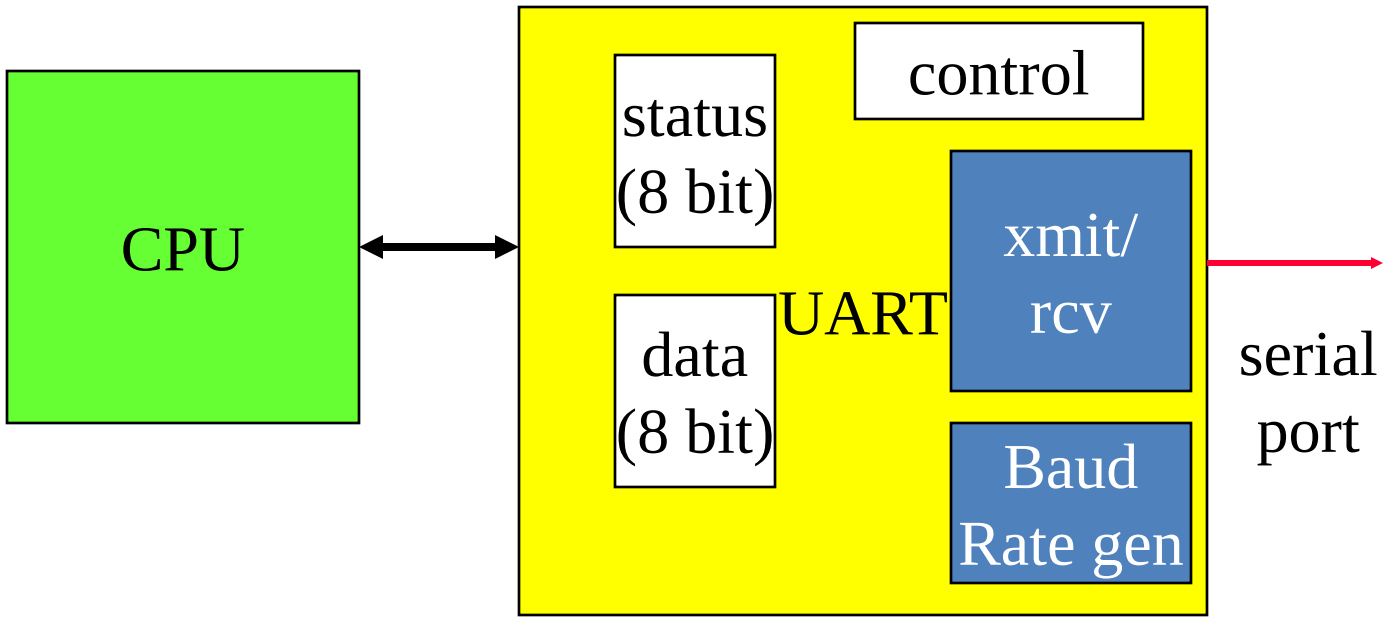
- Universal asynchronous receiver transmitter (UART) : provides serial communication.
- UARTs are integrated into most microcontrollers
  - Two UART modules on LPC2292
- Allows many communication parameters to be programmed.

Asynchronous serial communication

- Characters are transmitted separately:



LPC2292 UART CPU interface



Serial communication parameters set via UART “Line Control Register”

- Number of bits per character (5,6,7,8 bits).
- Enable/disable parity generation/checking.
- Type of parity bit: Even, Odd, Stuck-0, Stuck-1.
- Length of stop bit (1, 2 bits).

UART status register

- Receiver data ready
  - Newly-received data in RBR
- Transmitter Holding empty
  - THR ready to accept new data
- Transmitter Empty
  - All data has been transmitted
- FE, OE, PE – framing/overrun/parity error in received data

Programming I/O

- Two types of instructions can support I/O:
  - special-purpose/isolated I/O instructions;
  - memory-mapped load/store instructions.
- Intel x86 provides `in`, `out` instructions ("isolated I/O").
- Most CPUs use memory-mapped I/O.
- Having special I/O instructions do not preclude memory-mapped I/O.

68HC11A8 – Memory address space (Von Neumann)

- All addresses mapped into a 64K-byte memory address space
  - RAM: 0000-00FF (can re-map to 4K page)
  - I/O Registers: 1000-103F (can re-map)
  - EEPROM: B600-B7FF
  - BOOT ROM: BF40-BFFF
  - USER ROM: E000-FFFF

Intel 8051 On-chip address spaces (Harvard architecture)

- Program storage: 0000-0FFF
- Data address space:
  - RAM: 00-7F
    - low 32 bytes in 4 banks of 8 registers R0-R7
  - Special function registers: 80-FF
    - includes "ports" P0-P3
- Special I/O instructions for ports P0-P3

Peek and poke

- Traditional HLL interfaces for memory and memory-mapped I/O (compiler-dependent):

```
/* return contents of memory location */
int peek(char *location) {
  return *location; }

/* modify contents of memory location */
void poke(char *location, char newval) {
  (*location) = newval; }
```

(Not used in Keil ARM C compiler)

CPUs with "isolated" I/O addresses

- Separate address spaces for memory and I/O

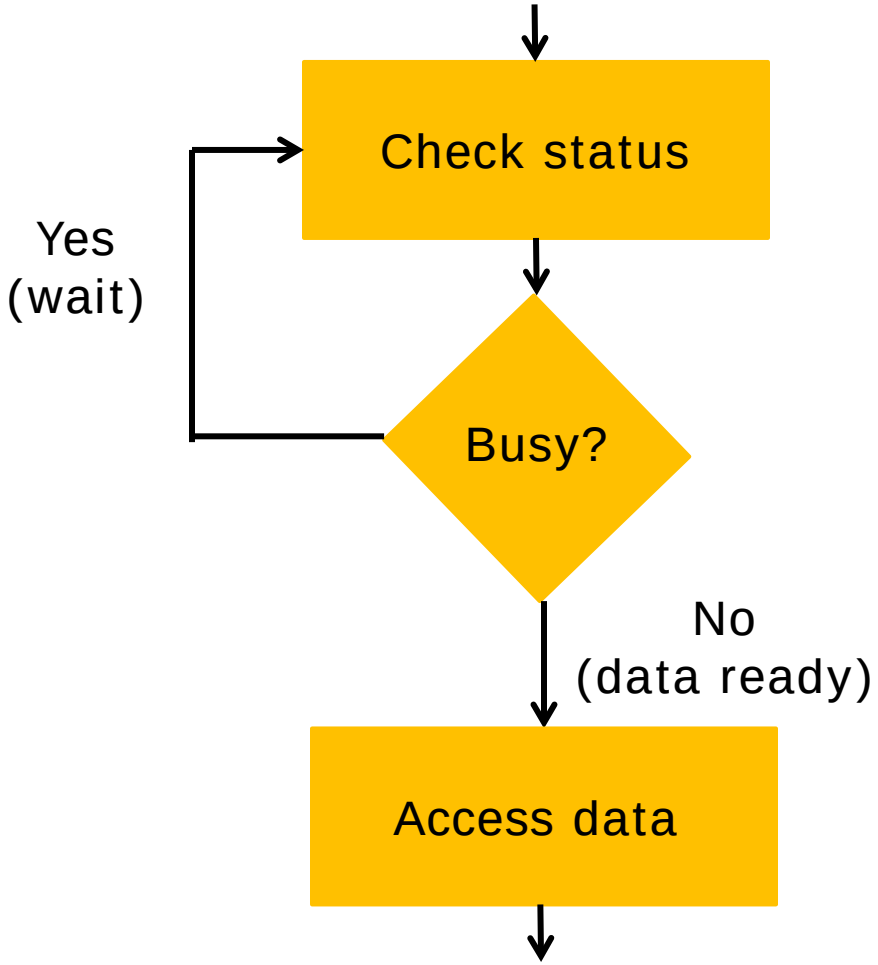
- Compiler-dependent functions for isolated I/O on Intel CPUs:

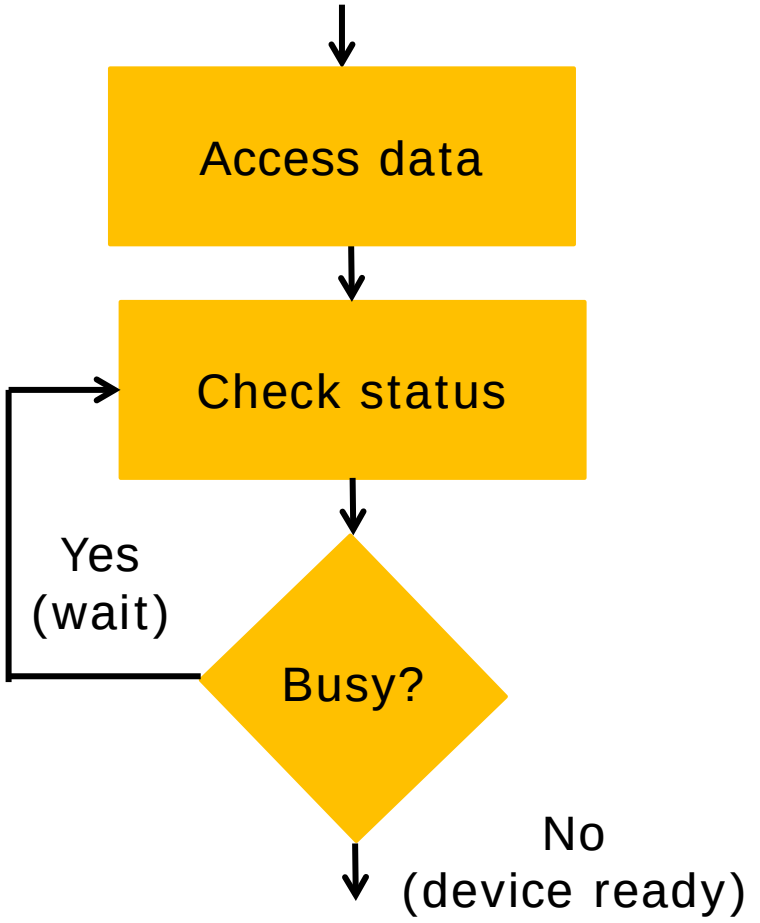
```
/* return contents of input port */
int inp(char *location) {
  return *location; }

/* modify contents of output port */
void outp(char *location, char newval) {
  (*location) = newval; }
```

(Not used in ARM)

Busy-wait I/O ("program-controlled")





Busy/wait output example

- Simplest way to program device.
  - Instructions test for device ready.
  - OUT_CHAR and OUT_STATUS are device addresses

```
/* send a character string */
current_char = mystring;
while (*current_char != '\0') {
  OUT_CHAR = *current_char;
  while (OUT_STATUS != 0);  //wait while busy
  current_char++;
}
```

Busy/wait output (ARM assy.lang.)

```
;output character from r0
#define OUT_STATUS  0x1000
#define OUT_CHAR    0x1004

   ldr  r1,=OUT_STATUS ;point to status
w  ldrb r2,[r1]        ;read status reg
   ands r2,r2,#1       ;check ready bit
   beq w               ;repeat until 1
   ldr r2,=OUT_CHAR    ;point to char
   strb r0,[r2]        ;send char to reg
```

Simultaneous busy/wait input and output

```
while (TRUE) {
  /* read */
  while (IN_STATUS == 0);  //wait until ready
  achar = IN_DATA;
  /* write */
  OUT_DATA = achar;
  while (OUT_STATUS != 0); //wait until ready
  }
```

NOTE:
Assumes all 8 bits of IN_STATUS = 0
May need to test a single bit