

# Processors



## ⌘ Execute programs

- ⊞ Serial execution of instructions
- ⊞ Simple, Universal

## ⌘ Instruction execution engine: fetch/execute cycle

- ⊞ flow of control determined by modifications to program counter
- ⊞ instruction classes:
  - ⊞ data: move, arithmetic and logical operations
  - ⊞ control: branch, loop, subroutine call
  - ⊞ interface: load, store from external memory

## ⌘ Traditional architecture goal: Performance

- ⊞ Caches
- ⊞ Branch prediction
- ⊞ Multiple/OOO issue

# Embedded Processors



- ⌘ Processor is a Universal computing engine
  - ☒ Program can compute arbitrary functions
  - ☒ Use a processor for simple/specific tasks
- ⌘ Advantages:
  - ☒ High-level language
  - ☒ Compilers/Debuggers
  - ☒ arbitrary control structures
  - ☒ arbitrary data structures
- ⌘ Disadvantages:
  - ☒ Cost/Size

# Embedded Processor (Microcontroller)

## ⌘ Processor optimized for low cost

- ☒ No cache
- ☒ Small memory
- ☒ No disks
- ☒ 4 bit/8 bit/16 bit
- ☒ No FP
- ☒ No complicated datapath
- ☒ Multicycle instruction interpretation
- ☒ Simple/no operating system
- ☒ Programs are static

## ⌘ Low performance

- ☒ 1 MIPS is enough if 1 ms is the time scale

## ⌘ Integrate on a single chip

# General-purpose processor



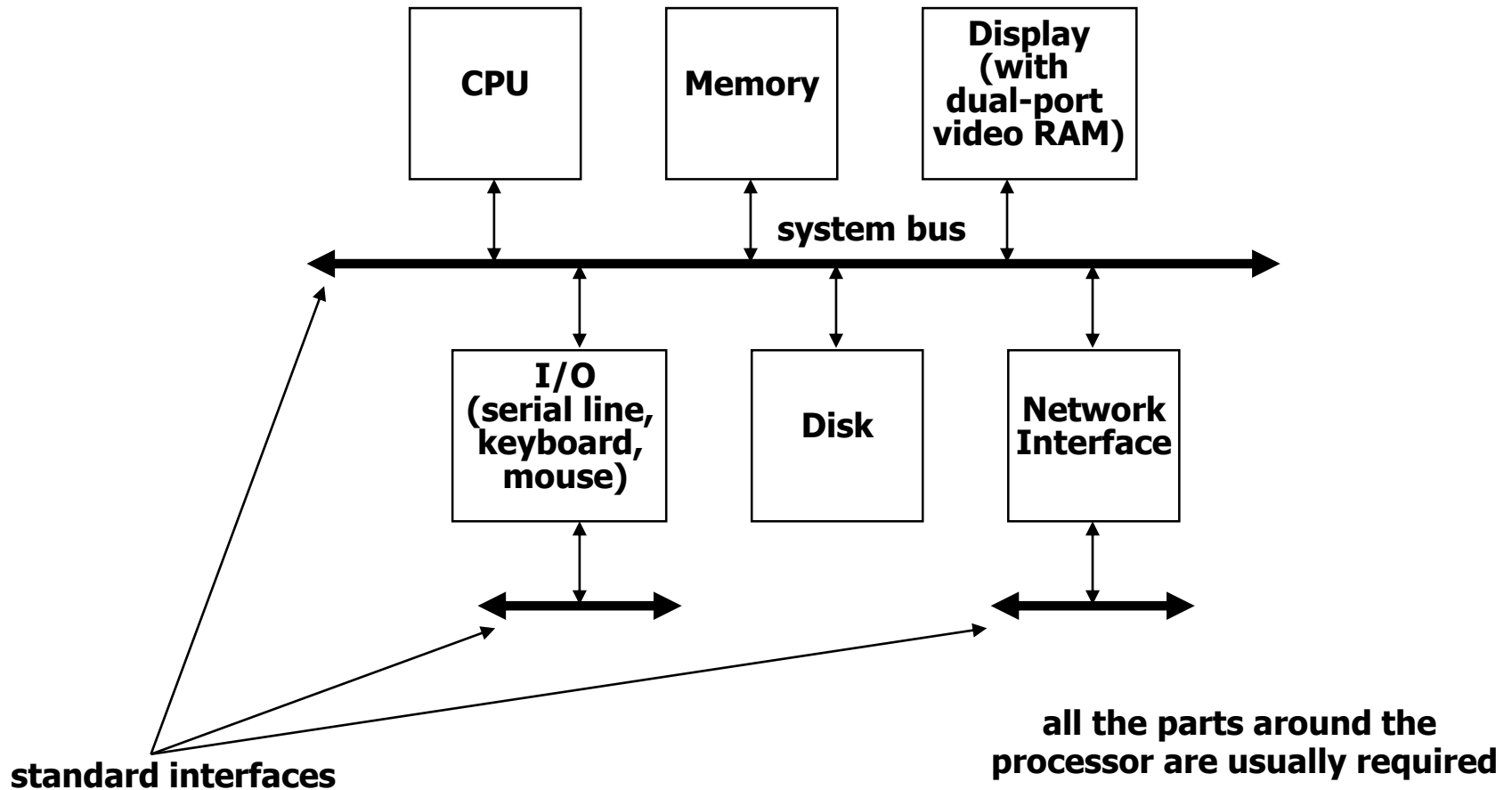
- ⌘ Programmed by user
- ⌘ New applications are developed routinely
- ⌘ General-purpose
  - ☒ must handle a wide ranging variety of applications
- ⌘ Interacts with environment through memory
  - ☒ all devices communicate through memory
  - ☒ DMA operations between disk and I/O devices
  - ☒ dual-ported memory (as for display screen)
  - ☒ oblivious to passage of time (takes all the time it needs)

# Embedded processors

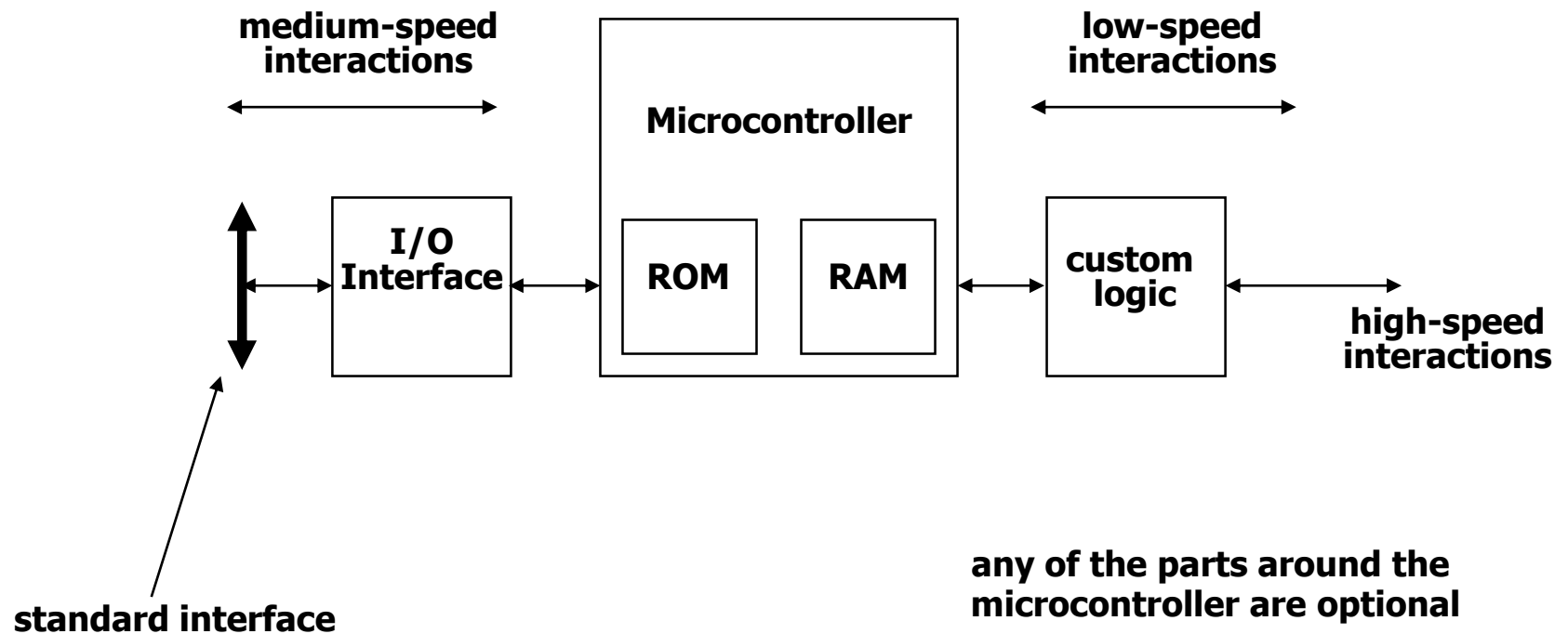


- ⌘ Programmed once by manufacturer of system
- ⌘ Executes a single program (or a limited suite) with few parameters
- ⌘ Task-specific
  - ☒ can be optimized for specific application
- ⌘ Interacts with environment in many ways
  - ☒ direct sensing and control of signal wires
  - ☒ communication protocols to environment and other devices
  - ☒ real-time interactions and constraints

# Typical general-purpose architecture



# Typical task-specific architecture



# How does this change things?



## ⌘ Sense and control of environment

- ☒ processor must be able to “read” and “write” individual signal wires
- ☒ controls I/O devices directly

## ⌘ Program has to do everything

- ☒ sense input bits
- ☒ change output bits
- ☒ do computation
- ☒ measure time

☒ many applications require precise spacing of events

## ⌘ Problems with this

- ☒ Precise timing?
- ☒ Too slow?



# Connecting to inputs/outputs

⌘ Map external wire to a bit of a variable (memory or register)

⊞ if (a & 1) ...

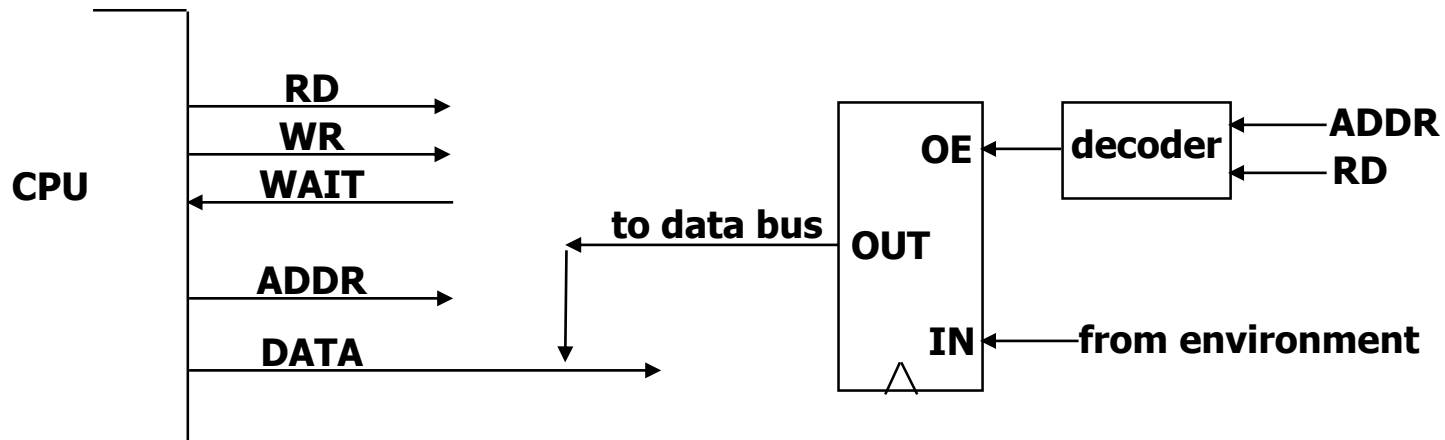
⊞ X = 2;

# Example Problem: Accelerometer Interface

- ⌘ Accelerometer output has one wire!
  - ⊞ Acceleration coded as the duty cycle
    - ⊞ pulse-width/cycle-length
      - 20% = -128
      - 80% = +127
    - ⊞ cycle time = 10ms
- ⌘ Write a C program that measures the acceleration
  - ⊞ Input is low-order bit of variable X
  - ⊞ Assign result to variable Z
  - ⊞ Make up whatever you need

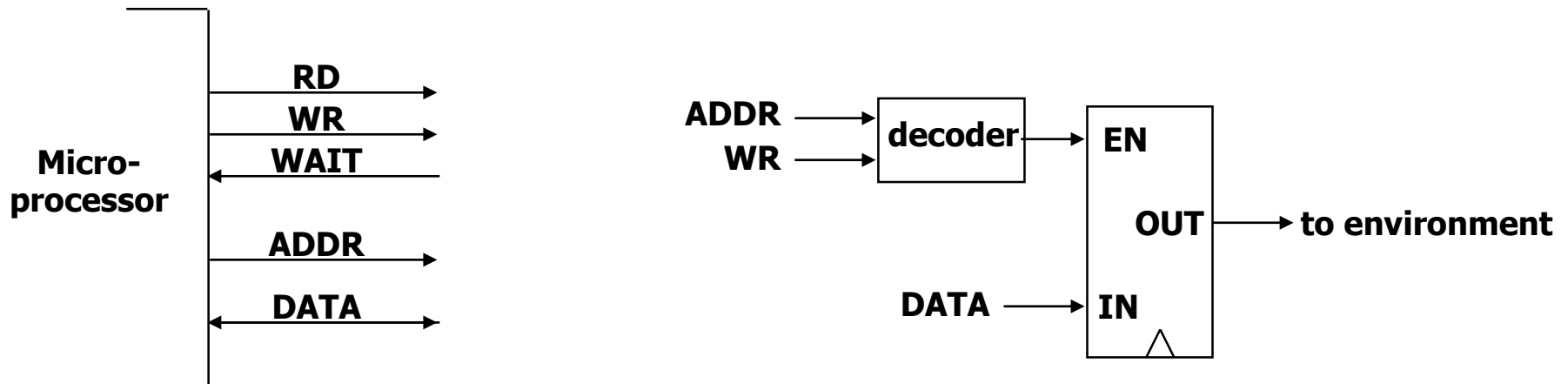
# Memory-mapped inputs

- ⌘ Map external wire to a bit in the address space of the processor
- ⌘ External register buffers values coming from environment
  - ☑ map register into address space
  - ☑ decoder needed to select register for reading
  - ☑ output enable (OE) so that many registers can use the same data bus



# Memory-mapped outputs

- ⌘ Map external wire to a bit in the address space of the processor
- ⌘ Connect output of memory-mapped register to environment
  - ☑ map register into address space
  - ☑ decoder need to select register for writing (holds value indefinitely)
  - ☑ input enable (EN) so that many registers can use the same data bus



# On-chip support for communication

- ⌘ Processor may not be fast enough
- ⌘ Offload standard protocols
- ⌘ Built-in device drivers
  - ☒ for common communication protocols
  - ☒ serial-line protocols most common as they require few pins
- ⌘ e.g. RS-232 serial interface
  - ☒ special registers in memory space for interaction
- ⌘ Increases level of integration
  - ☒ pull external devices on-chip
    - ☒ must be standard
  - ☒ eliminate need for shared memory or system bus

# Measuring Time

- ⌘ Keep track of detailed timing of each instruction's execution
  - ☒ highly dependent on code
  - ☒ hard to use with compilers
  - ☒ not enough control over code generation
  - ☒ interactions with caches/instruction-buffers
- ⌘ Loops to implement delays
  - ☒ keep track of time in counters
  - ☒ keeps processor busy counting and not doing other useful things
- ⌘ Real-time clock
  - ☒ sample at different points in the program
  - ☒ simple difference to measure time delay

# Timers

## ⌘ Separate and parallel counting unit(s)

- ⊞ co-processor to microprocessor
- ⊞ does not require microprocessor intervention
- ⊞ in simple case, like a real-time clock
  - ⊞ set timer/read timer
- ⊞ interrupt generated when expired

## ⌘ More interesting timer units

- ⊞ self reloading timers for regular interrupts
- ⊞ pre-scaling for measuring larger times
- ⊞ started by external events

# Input/output events



## ⌘ Input capture

- ☒ record precise time when input event occurred
- ☒ to be used in later handling of event

## ⌘ Output compare

- ☒ set output to happen at a point in the future
- ☒ reactive outputs - set output to happen a pre-defined time after some input
- ☒ processor can go on to do other things in the meantime



# Example Microcontroller: 8051



- ⌘ Very old, very common and very cheap microcontroller
  - ☒ Lots of variants
- ⌘ Review online documentation
  - ☒ learn how to read documentation
- ⌘ Instruction set
  - ☒ instruction capabilities
  - ☒ timing
- ⌘ Special registers and integrated I/O devices
  - ☒ I/O ports
  - ☒ serial interface
- ⌘ Interrupt organization
- ⌘ Memory space and its allocation
- ⌘ Timers

# Why this choice?

- ⌘ Used on the XESS board we are using
- ⌘ Atmel part with 20K on-chip flash memory
  - ⊞ dispense with external memory for instructions
- ⌘ 8051
  - ⊞ very common microcontroller with simple instruction set
  - ⊞ lots of features
  - ⊞ lots of alternatives
  - ⊞ lots of support and resources
  - ⊞ good tools available: we will use the Keil software
    - ⊞ Assembler
    - ⊞ C compiler
    - ⊞ Debugger
  - ⊞ we will use the C compiler mostly
    - ⊞ requires a good understanding of the 8051 architecture