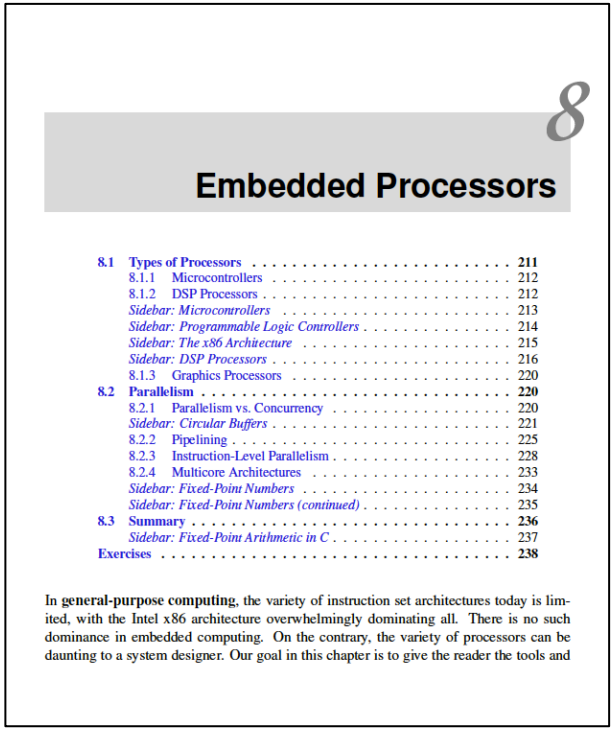
Lecture 4/5 : Embedded Processors

Slides were originally developed by Profs. Edward Lee and Sanjit Seshia, and subsequently updated by Profs. Gavin Buskes and Iman Shames.

Outline

- Type of processors
  - DSP Processors
  - Microcontrollers
    - Basic architecture
    - ATMEGA16 architecture
- Parallelism
  - Parallelism vs concurrency
  - Pipelining
  - Types of parallelism

Focus on concurrency and control over timing.



8

Embedded Processors

In **general-purpose computing**, the variety of instruction set architectures today is limited, with the Intel x86 architecture overwhelmingly dominating all. There is no such dominance in embedded computing. On the contrary, the variety of processors can be daunting to a system designer. Our goal in this chapter is to give the reader the tools and

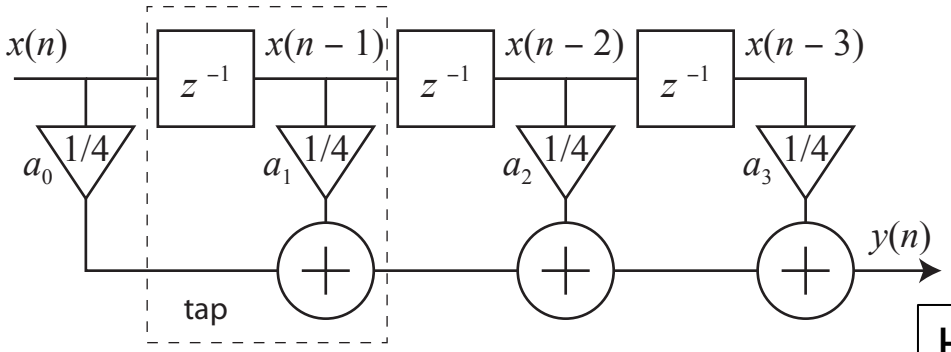
Embedded processors

- In *general-purpose* computing, the variety of instruction set architectures (ISA) today is limited, with the Intel x86 architecture overwhelmingly dominating all.
  - There is no such dominance in embedded computing.
- When deployed in a product, embedded processors typically have a dedicated function.
  - Control a car engine or measure ice thickness in the Arctic.
- Making them more specialised can bring enormous benefits :
  - consume far less energy, and consequently be usable with small batteries for long periods of time.
  - include specialised hardware to perform operations that would be costly to perform on general-purpose hardware, such as image analysis.

Digital Signal Processors (DSPs)

- Many embedded applications involve signal processing.
  - DSPs are designed specifically to support numerically intensive signal processing applications.
- The structure of such processors is highly specialised depending on the application
  - E.g. Finite Impulse Response (FIR) filter

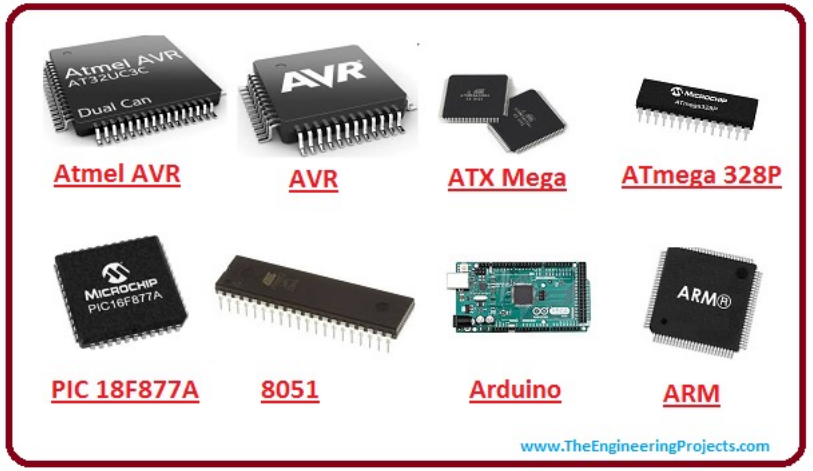
$$y(n) = \sum_{i=0}^{N-1} a_i x(n-i)$$



Here the taps are all 1/4

Microcontrollers

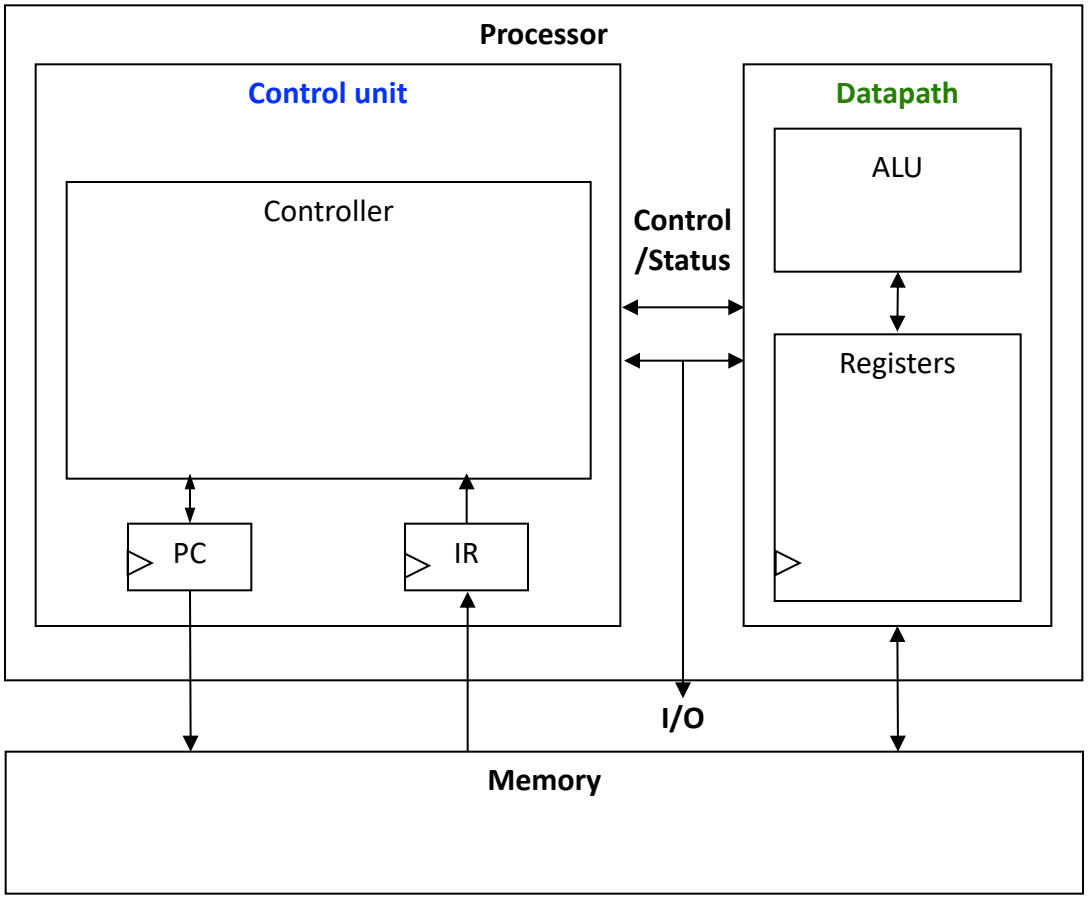
- A microcontroller (uC, μC) is a small computer on a single integrated circuit consisting of a relatively simple central processing unit (CPU) combined with peripheral devices such as memories, I/O devices, and timers.



- They may consume extremely small amounts of energy, and often include a sleep mode that reduces the power consumption to nanowatts.

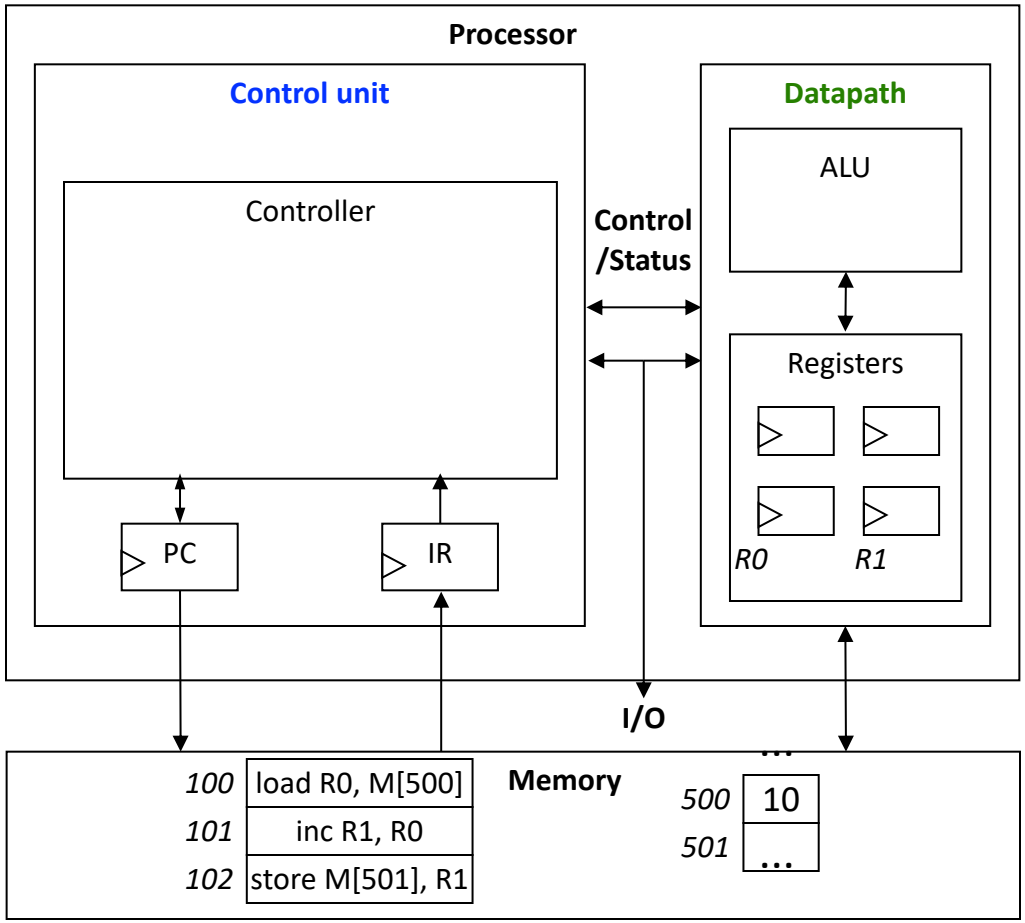
Basic uC Architecture

- Control unit and datapath
  - Note similarity to single-purpose processor
- Key differences
  - Datapath is general
  - Control unit doesn't store the algorithm – the algorithm is "programmed" into the memory

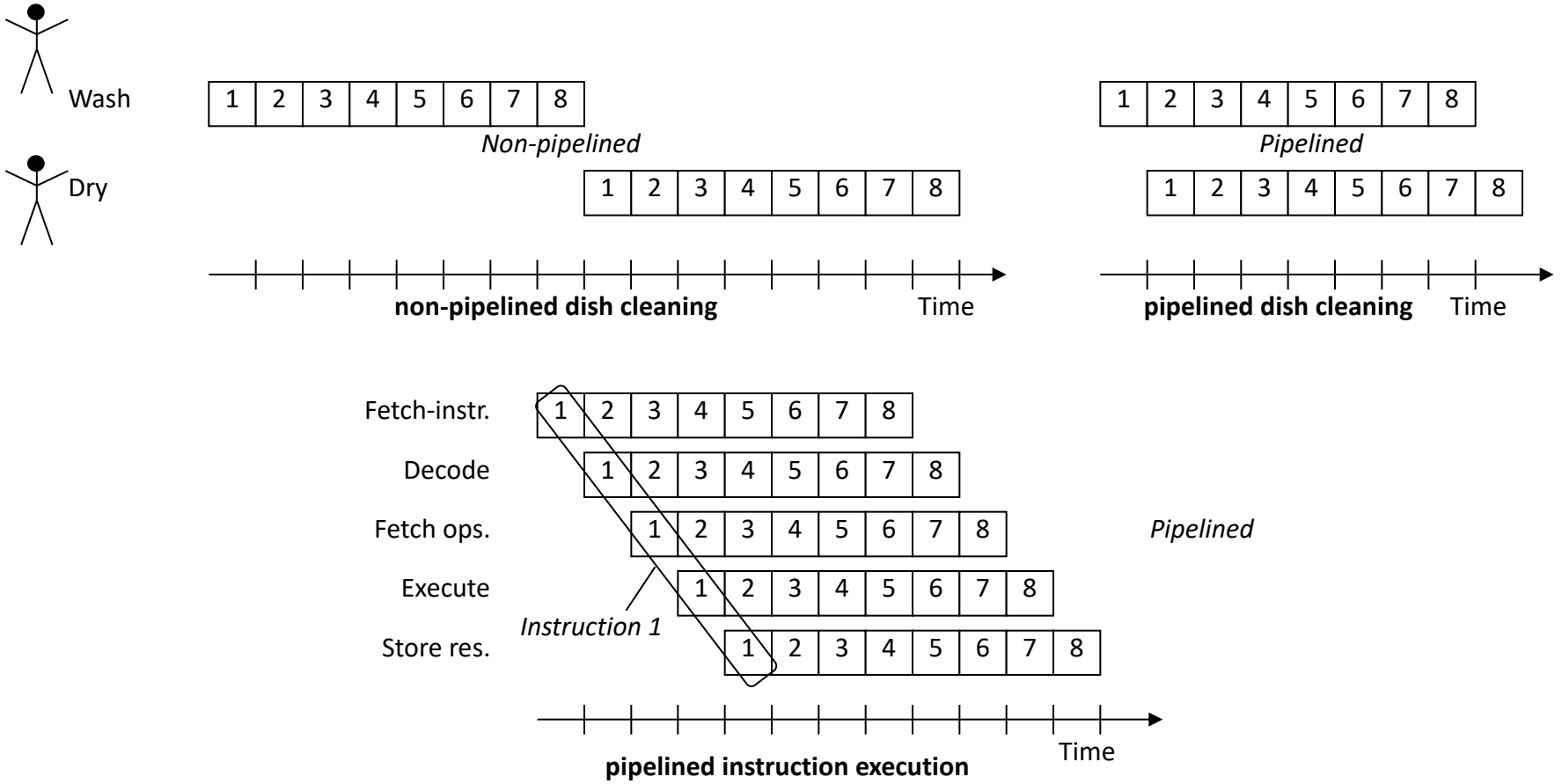


Control Unit

- Control unit: configures the datapath operations
  - Sequence of desired operations ("instructions") stored in memory
- Instruction cycle – broken into several sub-operations, each one clock cycle, e.g.:
  - **Fetch**: Get next instruction into IR
  - **Decode**: Determine what the instruction means
  - **Fetch operands**: Move data from memory to datapath register
  - **Execute**: Move data through the ALU
  - **Store results**: Write data from register to memory

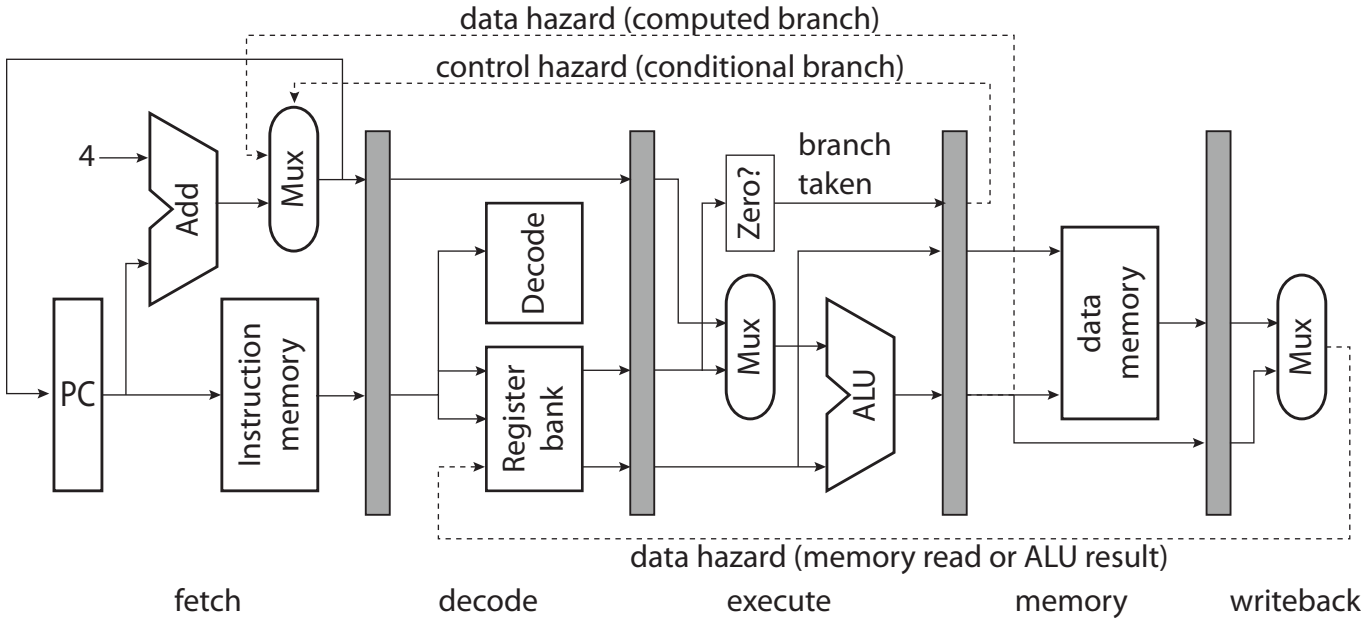


Pipelining: Increasing Instruction *Throughput*



Pipelining

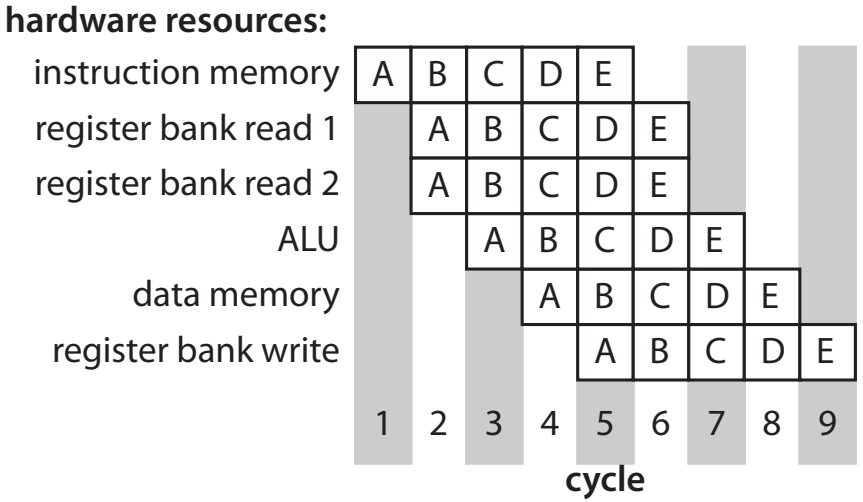
- Most modern processors are pipelined. A simple five-stage pipeline for a 32-bit machine



- Shaded rectangles are latches, clocked at the processor clock rate.
- On each edge of the clock, the value at the input is stored in the latch register.
- The output is then held constant until the next edge of the clock, allowing the circuits between the latches to settle.

Pipelining – data hazards

- A reservation table table shows hardware resources that may be simultaneously used.
- In this case, the register bank appears three times considering the pipeline from the previous slide and assumes that two reads and write of the register file can occur in each cycle.



The write by A occurs in cycle 5, but the read by B occurs in cycle 3. Thus, the value that B reads will not be the value that A writes => data hazard!

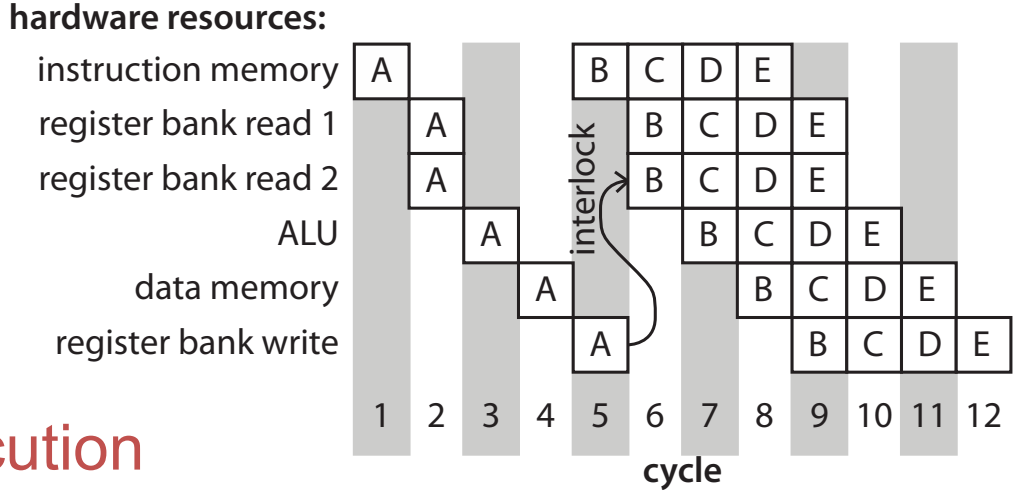
Pipelining – dealing with data hazards

- Computer architects have tackled the problem of pipeline hazards in a variety of ways:
  - Explicit pipeline
  - Interlocks
  - Out-of-order execution

- Explicit pipeline
  - Pipeline hazard is documented, and programmer (or compiler) must deal with it, e.g. if B reads a register written by A, the compiler inserts three **no-op instructions** between A and B to ensure that the write occurs before the read.

Pipelining – dealing with data hazards

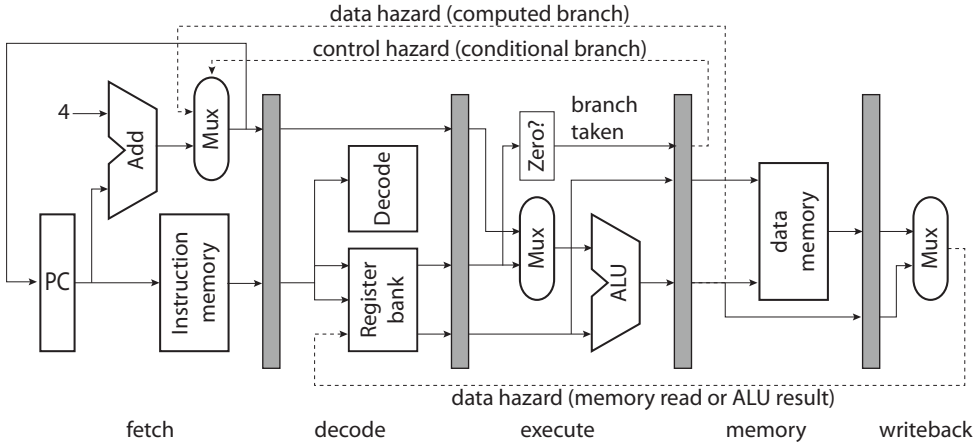
- Interlocks
  - Instruction decode hardware, upon encountering instruction B that reads a register written by A, will detect the hazard and delay the execution of B until A has completed the writeback stage.

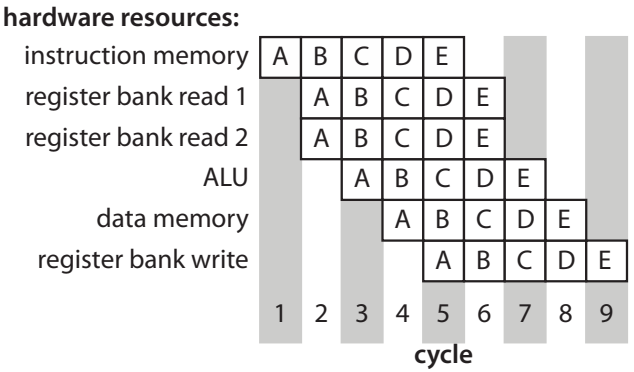


- Out-of-order-execution
  - When hardware is provided that detects hazards, instead of delaying execution of B, proceeds to fetch C, and if C does not read registers written by either A or B, and does not write registers read by B, then proceeds to execute C before B. This reduces the number of pipeline bubbles.

Pipelining – control hazards

- Control hazard





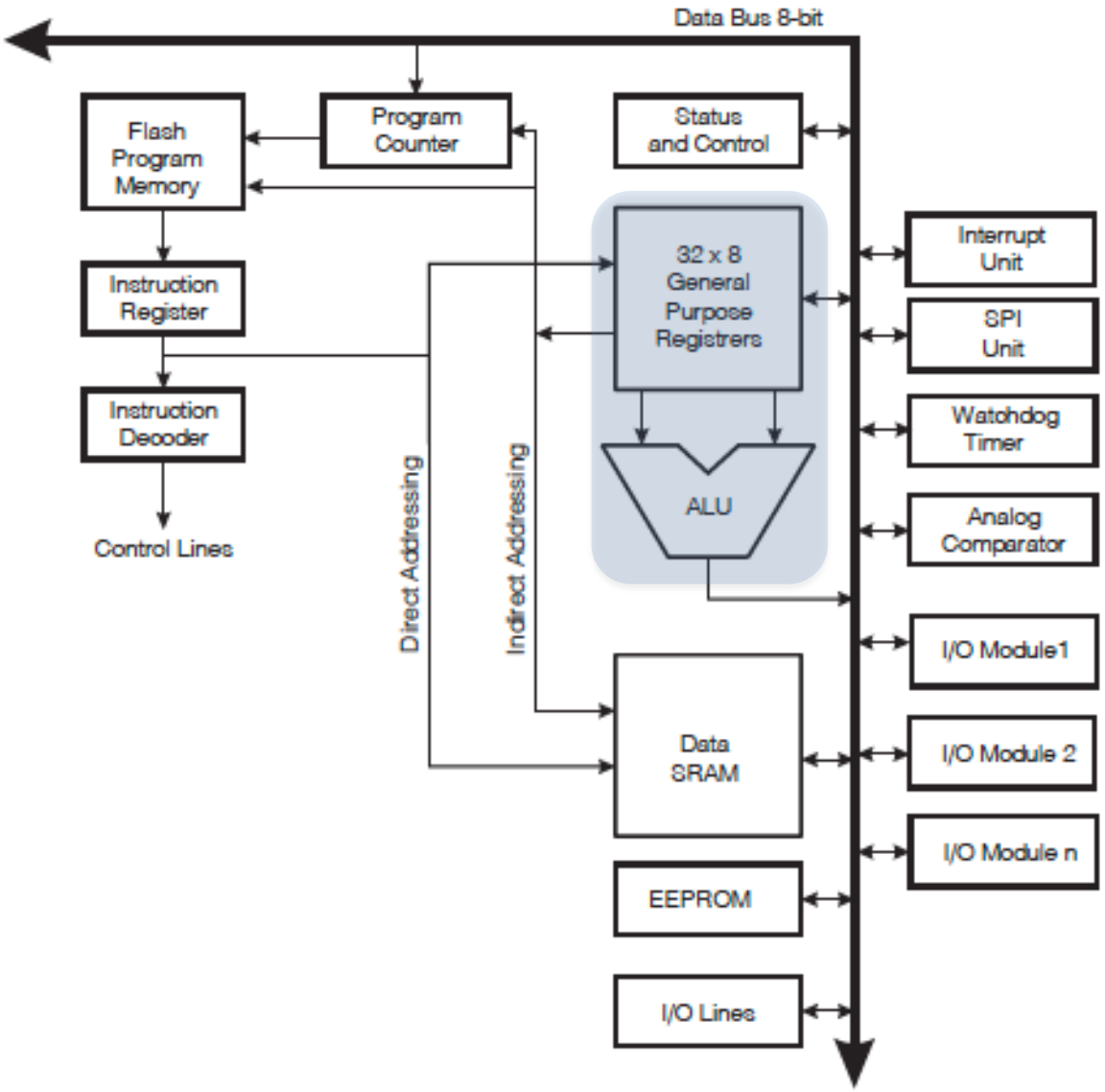
  - e.g. a conditional branch instruction changes the value of the PC if a specified register has value zero.
    - The new value of the PC is provided (optionally) by the result of an ALU operation.
    - In this case, if A is a conditional branch instruction, then A has to have reached the memory stage **before** the PC can be updated.
    - Instructions that follow A in memory will have been fetched and will be at the decode and execute stages already by the time it is determined that those instructions **should not** in fact be executed.

Pipelining – dealing with control hazard

- Delayed branch
  - documents the fact that the branch will be taken some number of cycles after it is encountered, and leaves it up to the programmer (or compiler) to ensure that the instructions that follow the conditional branch instruction are either harmless (like no-ops) or do useful work that does not depend on whether the branch is taken.
- Speculative execution
  - hardware estimates whether the branch is likely to be taken, and begins executing the instructions it expects to execute. If its expectation is not met, then it undoes any side effects (such as register writes) that the speculatively executed instructions caused.
- Except for explicit pipelines and delayed branches, all of these techniques introduce variability in the timing of execution of an instruction sequence.

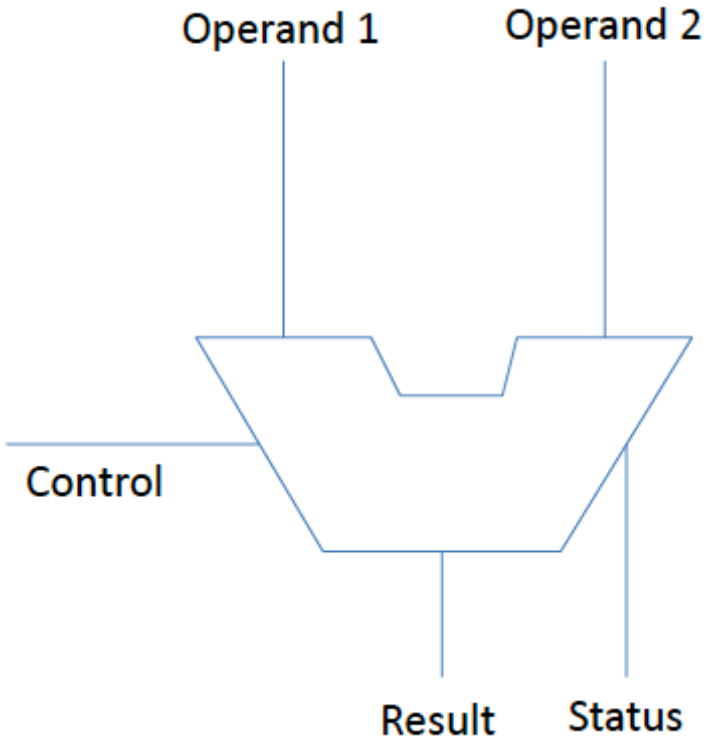
ATMEGA16 Architecture

ATMEL ATMEGA16 Datasheet page 8



Arithmetic Logic Unit (ALU)

- Two data inputs
  - Operands
- Control input
  - What to do with operands, i.e., add, subtract
- One data output
  - Result
- One status output
  - Status register



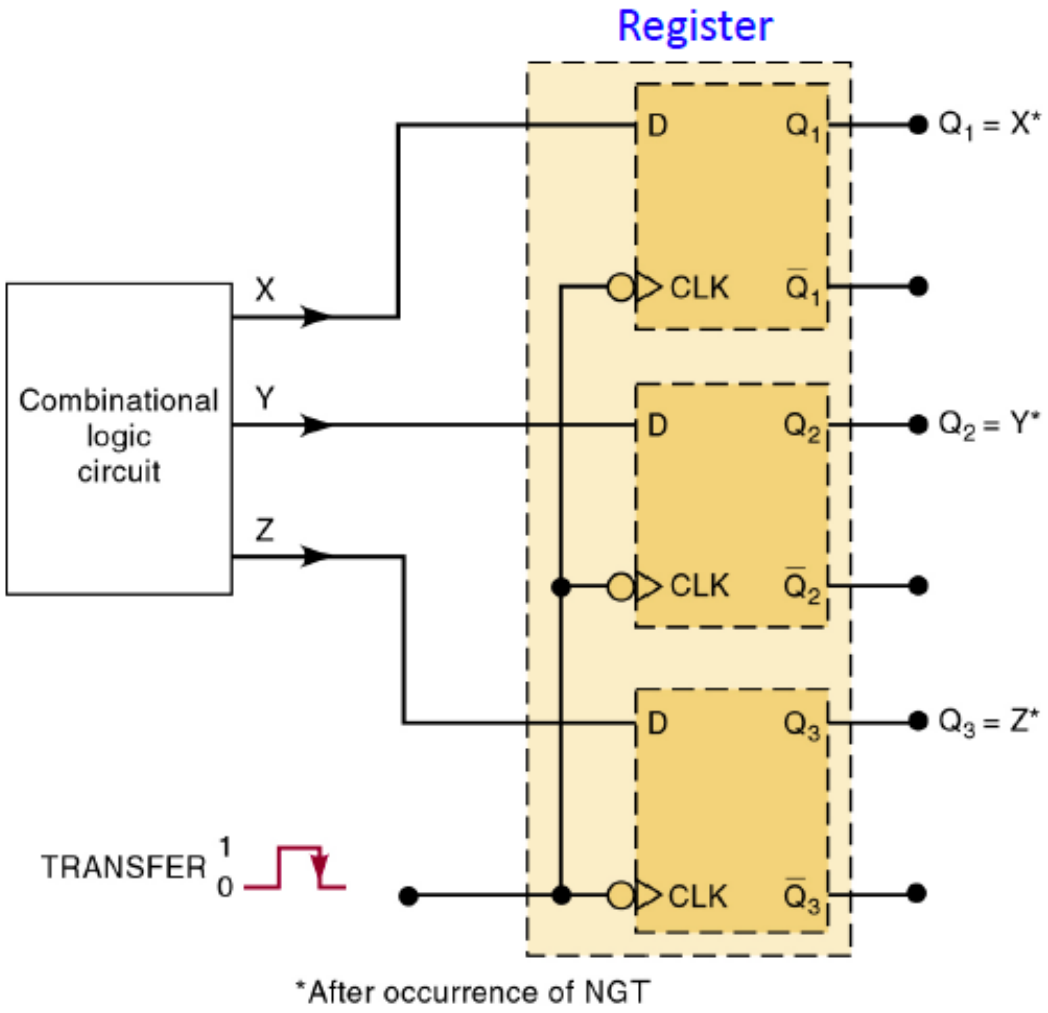
Registers

- Outputs X, Y, Z are to be transferred to multiple flipflops for storage

Using D flip-flops, levels present at X, Y & Z will be transferred to Q1, Q2 & Q3, upon application of a TRANSFER pulse to the common CLK inputs

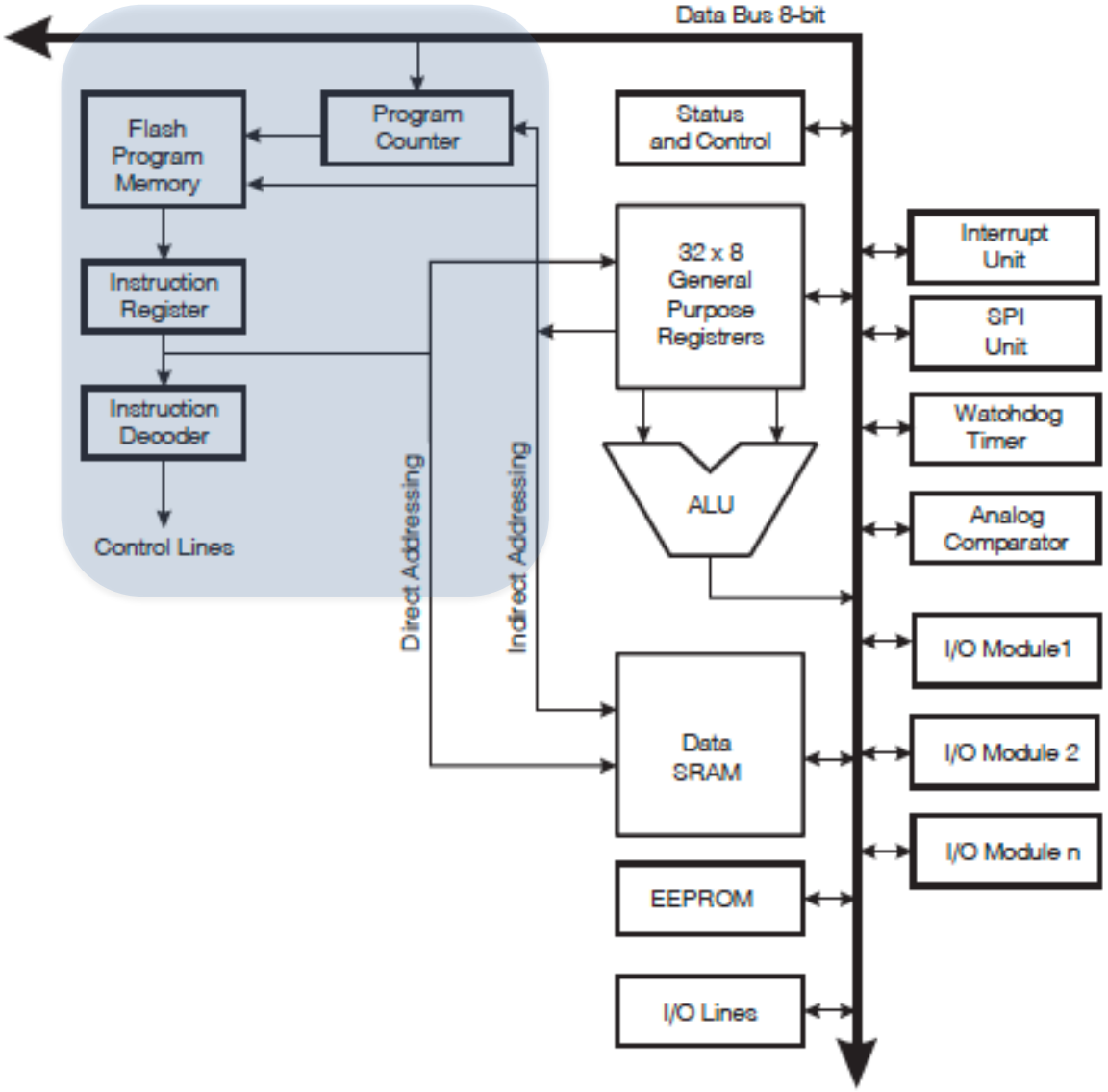
This is an example of parallel data transfer of binary data—the three bits X, Y & Z are transferred simultaneously

The array of flip-flops in parallel is called a register



ATMEGA16 Architecture

ATMEL ATMEGA16 Datasheet page 8



Program Counter

- Program counter is a register that holds the memory address of the next instruction that would be executed.
- Microprocessors usually fetch instructions sequentially from memory, but control transfer instructions change the sequence by placing a new value in the Program Counter (PC). These include branches (sometimes called jumps), subroutine calls, and returns:
  - A branch provides that the next instruction is fetched from elsewhere in memory.
  - A subroutine call not only branches but saves the preceding contents of the PC somewhere.
  - A return retrieves the saved contents of the PC and places it back in the PC, resuming sequential execution with the instruction following the subroutine call.

Example program

```
ldi r1, 0x19
; Load Immediate Register 1 25
ldi r2, 0x24
; Load Immediate Register 2 36
; ADD Register 1 to Register 2
add r1, r2
```

- The program is stored in Flash Program memory

Execution of Line 1

- Program counter points to program address with

```
ldi r1, 0x19
```

- On rising edge of clock 1, the instruction is latched into the instruction register and the program counter is incremented
- The instruction is decoded and the control signals are activated to do the following
  - Output the value 0x19 onto the direct addressing line
  - Input enable register 1
- On clock 2 rising edge 0x19 is latched into register 1

Execution of Line 2

- Program counter points to program address with

```
ldi r2, 0x25
```

- On rising edge of clock 2, the instruction is latched into the instruction register and the program counter is incremented
- The instruction is decoded and the control signals are activated to do the following
  - Output the value 0x25 onto the direct addressing line
  - Input enable register 2
- On clock 3 rising edge 0x25 is latched into register 2

Execution of Line 3

- Program counter points to program address with

```
add r1, r2
```

- On rising edge of clock 3, the instruction is latched into the instruction register and the program counter is incremented
- The instruction is decoded and the control signals are activated to do the following
  - Output the value from register 1 to input 1 of the ALU
  - Output the value from register 2 to input 2 of the ALU
  - The ALU instruction is set to ADD
  - The input of register 1 is enabled
- On clock 4 rising edge output of ALU is latched into register 1

What does the program look like?

- If we look at the flash address, we will see the following code:

```
1110 0001 0001 1001
1110 0010 0010 0101
0000 1100 0001 0010
```

- To work out what this means you need to refer to the AVR instruction set

- Time consuming, best done by compiler

Parallelism

- Concurrency is central to embedded systems.
  - A computer program is said to be **concurrent** if different parts of the program conceptually execute simultaneously.
  - A program is said to be **parallel** if different parts of the program physically execute simultaneously on distinct hardware (such as on multicore machines, on servers in a server farm, or on distinct microprocessors)
- Parallelism can take two broad forms
  - Multicore architectures
  - Instruction-Level Parallelism (ILP)
    - CISC Instructions
    - Subword Parallelism
    - Superscalar
    - VLIW

Instruction-Level Parallelism

- Complex Instruction Set Computer (CISC) Instructions
  - A processor with complex (and typically, rather specialised) instructions is called a CISC machine.
  - DSPs are typically CISC machines, and include instructions specifically supporting FIR filtering (and often other algorithms such as FFTs (fast Fourier transforms) and Viterbi decoding).
  - CISC instruction sets have their disadvantages
    - e.g. extremely challenging for a compiler to make optimal use of such an instruction set. As a consequence, DSP processors are commonly used with code libraries written and optimised in assembly language.

CISC is the opposite of RISC

II

Reduced Instruction Set Computer

Subword Parallelism

- Some processors support subword parallelism, where a wide ALU is divided into narrower slices enabling simultaneous arithmetic or logical operations on smaller words.
  - E.g. Intel MMX technology

- A vector processor is one where the instruction set includes operations on **multiple** data elements simultaneously. Subword parallelism is a particular form of vector processing.

Superscalar processors

- Superscalar processors use hardware that can simultaneously dispatch multiple instructions to distinct hardware units when it detects that such simultaneous dispatch will not change the behaviour of the program.
  - Execution of the program is identical to what it would have been if it had been executed in sequence. Such processors even support out-of-order execution, where instructions later in the stream are executed before earlier instructions.
- Superscalar processors have a significant disadvantage for embedded systems, which is that execution times may be extremely difficult to predict, and in the context of multitasking (interrupts and threads), may not even be repeatable

Very Large Instruction Word (VLIW)

- Processors intended for embedded applications often use VLIW architectures instead of superscalar in order to get more repeatable and predictable timing.
- VLIW processors include multiple functional units, like superscalar processors, but instead of dynamically determining which instructions can be executed simultaneously, **each instruction** specifies what **each functional unit** should do in a particular cycle.
  - A VLIW instruction set combines multiple independent operations into a single instruction.

Multicore architectures

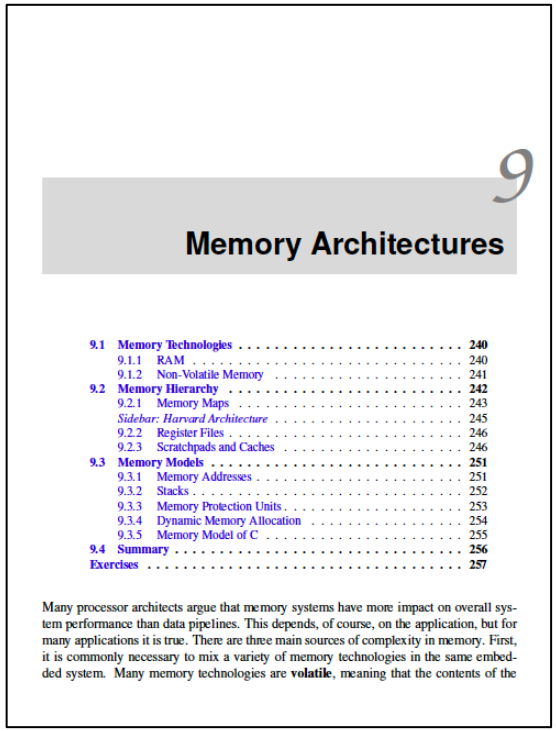
- Multicore machine - combination of several processors on a single chip.
- For embedded applications, multicore architectures have a significant potential advantage over single-core architectures because real-time and safety-critical tasks can have a dedicated processor.
  - e.g. mobile phones, since the radio and speech processing functions are hard real-time functions with considerable computational load.
  - In such architectures, user applications cannot interfere with real-time functions.

Multicore architectures

- Field-Programmable Gate Arrays (FPGAs)
  - Different type of multicore architecture, used in embedded applications .
  - Use one or more soft cores (processors implemented on FPGAs) together with custom hardware.
  - Hardware function programmable using hardware design tools.
  - Advantage of soft cores is that they can be tightly coupled to custom hardware more easily than off-the-shelf processors.

Things to do …

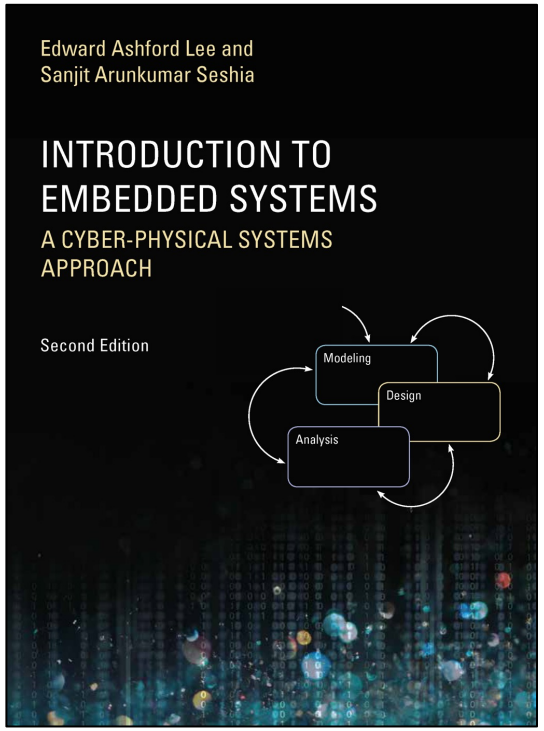
- Download the textbook and read Chapter 9
- Sign up to a group in your workshop class
- MUST do the HSW Quiz to attend on-campus workshop
- Read over Workshop 1 and do the pre-workshop work



9

Memory Architectures

Many processor architects argue that memory systems have more impact on overall system performance than data pipelines. This depends, of course, on the application, but for many applications it is true. There are three main sources of complexity in memory. First, it is commonly necessary to mix a variety of memory technologies in the same embedded system. Many memory technologies are **volatile**, meaning that the contents of the



Next Lecture

- Memory architectures