
Cyber-Physical Systems

Embedded Architecture



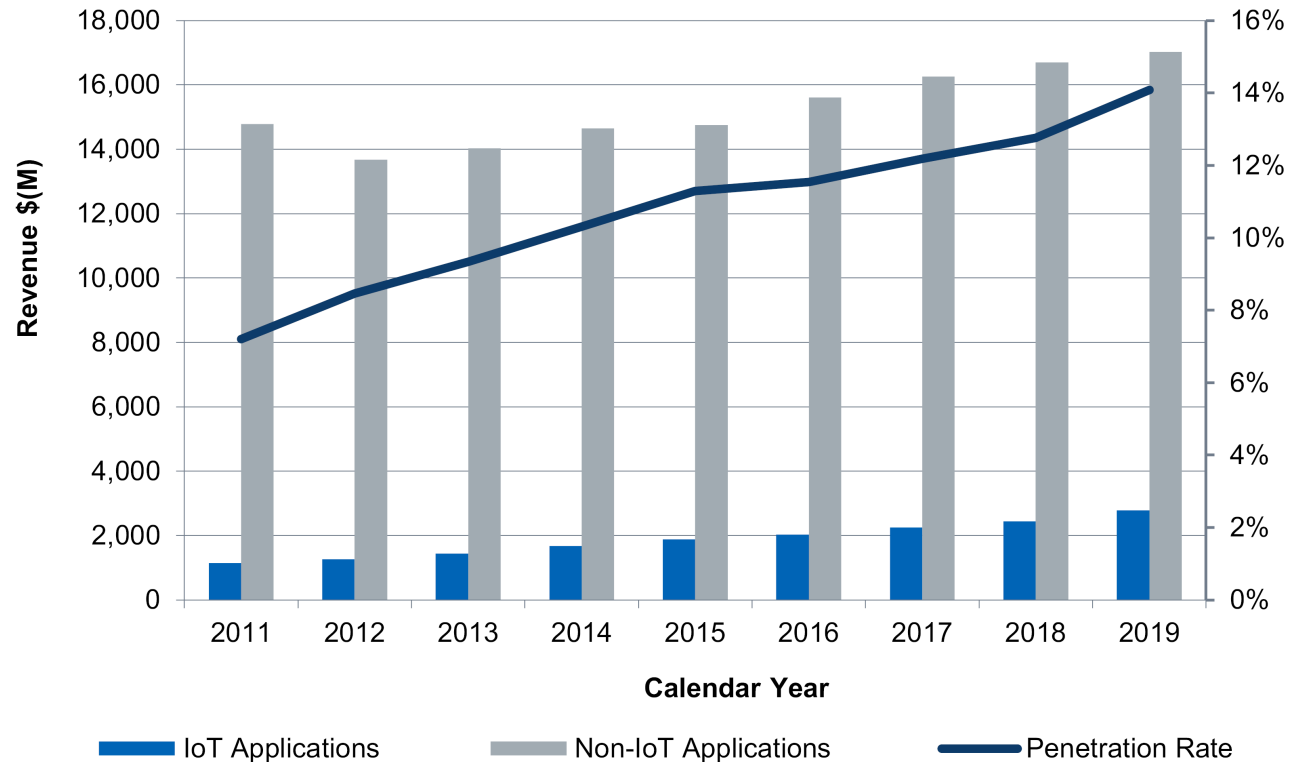
UNIVERSITY
AT ALBANY
State University of New York

IECE 553/453– Fall 2022

Prof. Dola Saha

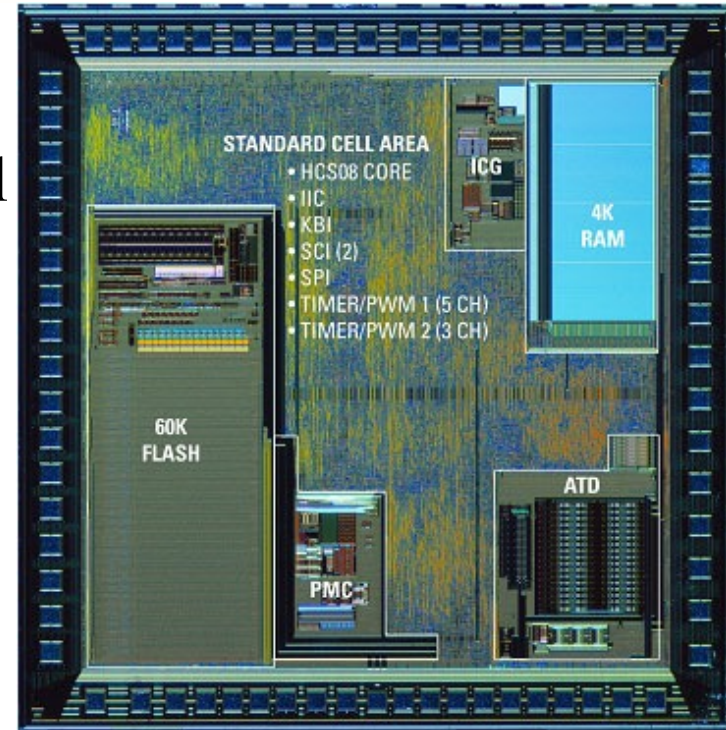
Introduction to Microcontrollers

MCU market in IoT applications compared to markets outside of IoT



Introduction to Microcontrollers

- A microcontroller (MCU) 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.
 - By some accounts, more than half of all CPUs sold worldwide are microcontrollers.
 - Such a claim is hard to substantiate because the difference between microcontrollers and general-purpose processors is indistinct.



Microcontrollers

- An Embedded Computer System on a Chip
 - A CPU
 - Memory (Volatile and Non-Volatile)
 - Timers
 - I/O Devices
- Typically intended for limited energy usage
 - Low power when operating plus sleep modes
- Where might you use a microcontroller?

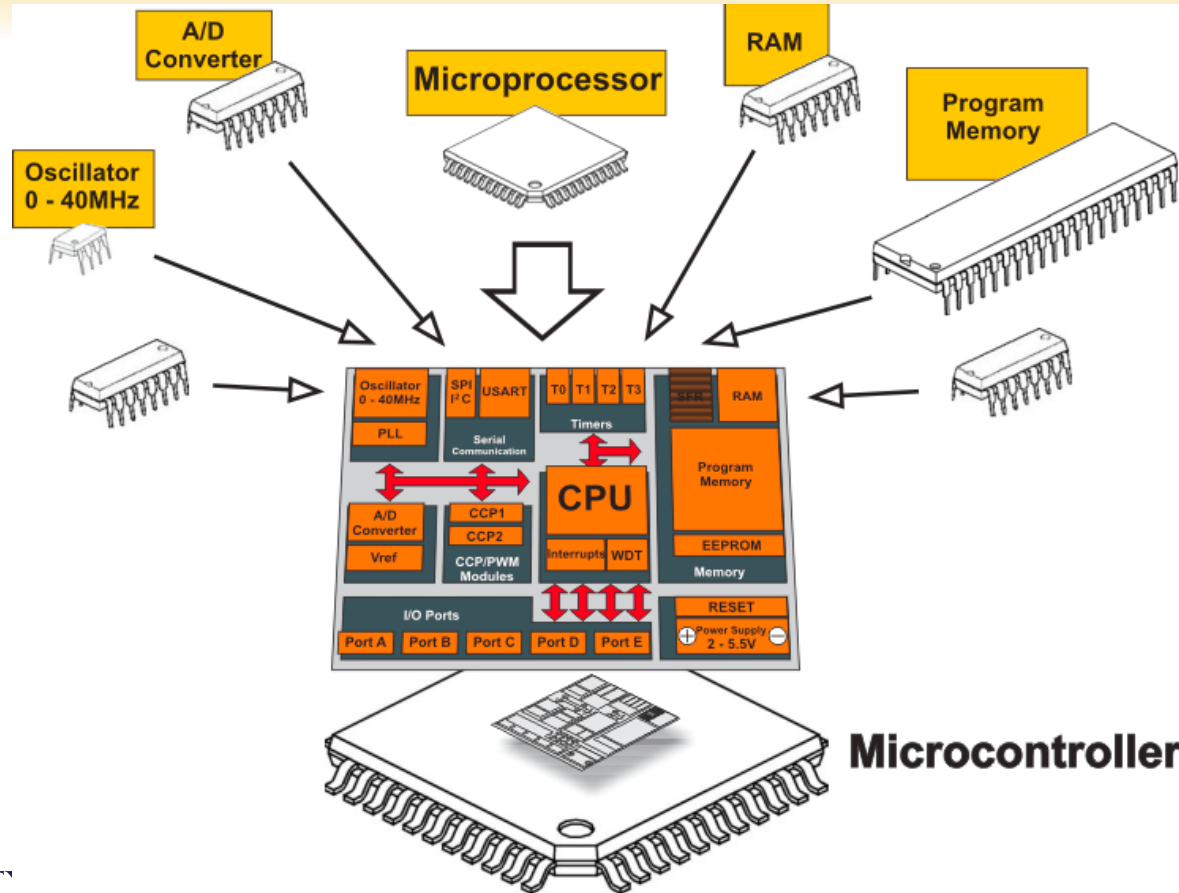
What is Control?

- Sequencing operations
 - Turning switches on and off
- Adjusting continuously (or at least finely) variable quantities to influence a process

Microcontroller vs Microprocessor

- A microcontroller is a small computer on a single integrated circuit containing a processor core, memory, and programmable input/output peripherals.
- A microprocessor incorporates the functions of a computer's central processing unit (CPU) on a single integrated circuit.

Microcontroller vs Microprocessor



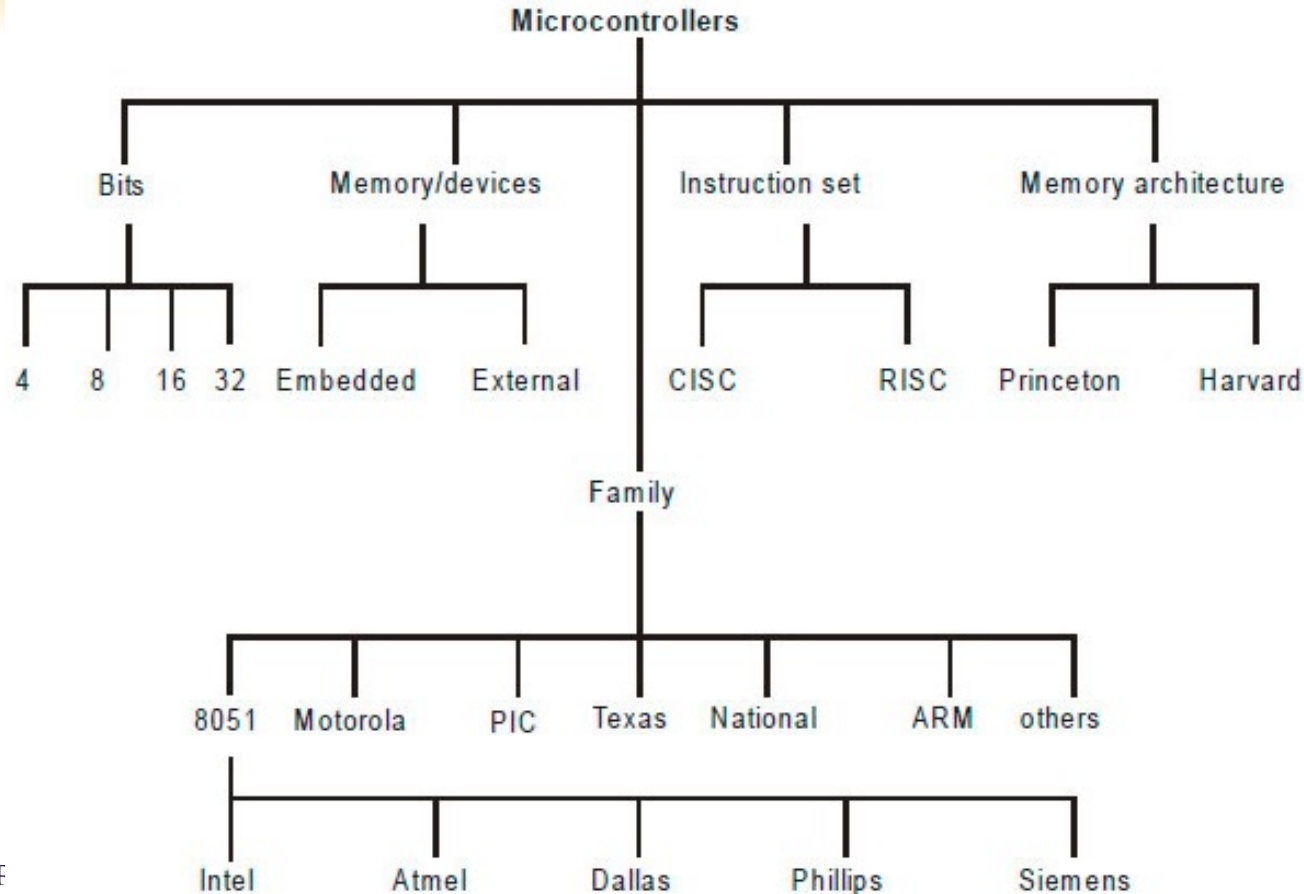
Types of Processors

- 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.
- Do you want same microprocessor for your watch, autonomous vehicle, industrial sensor?

How to choose?

- How to choose micro-processors/controllers?
- Things that matter
 - Peripherals
 - Concurrency & Timing
 - Clock Rates
 - Memory sizes (SRAM & flash)
 - Package sizes

Types of Microcontrollers



DSP Processors

- Processors designed specifically to support numerically intensive signal processing applications are called DSP processors, or DSPs (digital signal processors).
- Signal Processing Applications: interactive games; radar, sonar, and LIDAR (light detection and ranging) imaging systems; video analytics (the extraction of information from video, for example for surveillance); driver-assist systems for cars; medical electronics; and scientific instrumentation.

A Common Signal Processing Algorithm

- finite impulse response (FIR) filtering
- N is the length of the filter
- a_i are tap values
- $x(n)$ is the input

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

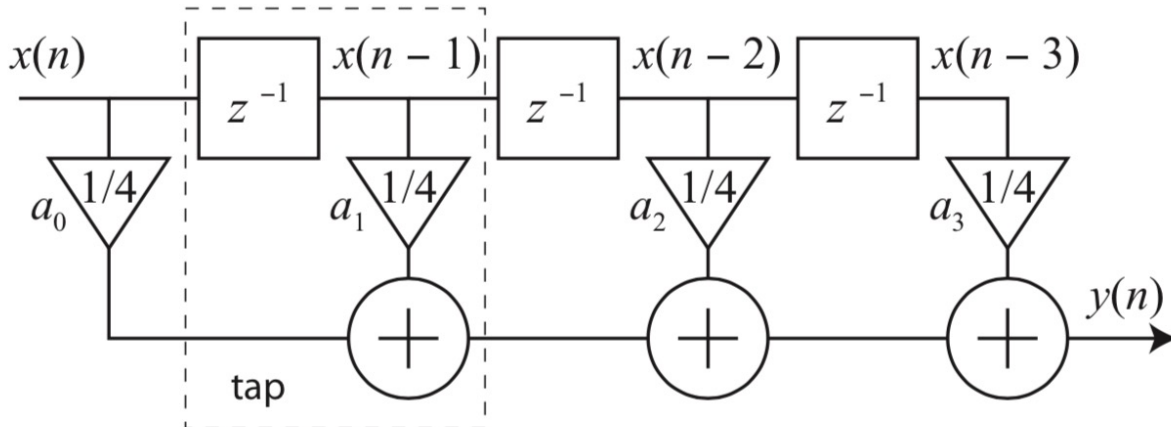
FIR Filter Formula

FIR Filter Implementation

- z^{-1} is unit delay
- Suppose $N = 4$ and $a_0 = a_1 = a_2 = a_3 = 1/4$.
- Then for all $n \in \mathbb{N}$,

$$y(n) = (x(n) + x(n-1) + x(n-2) + x(n-3))/4 .$$

- **Multiply-Accumulate**

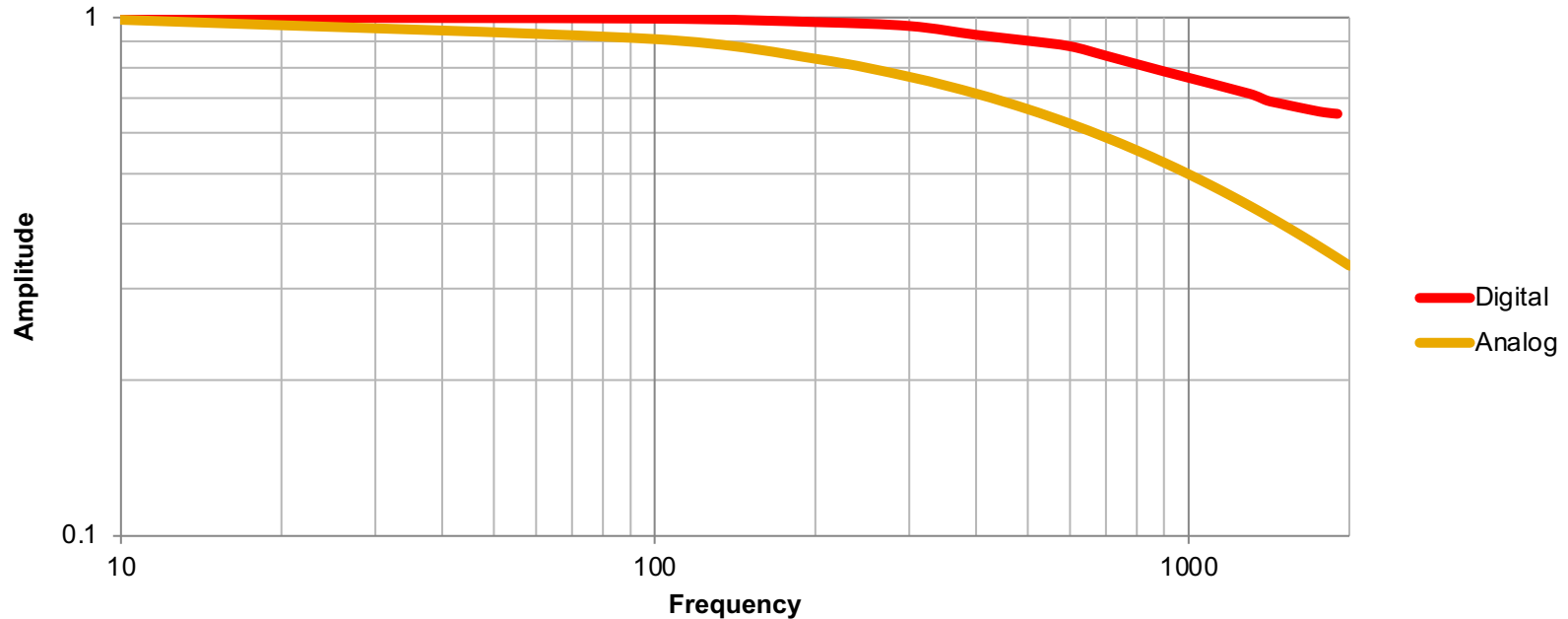


Multiply-Accumulate Instructions

- Digital Signal Processors provide a fast and efficient multiply-accumulate (MAC) instruction
 - Typically including a relatively large accumulator
- They also typically use a Harvard memory access architecture
- They may include auto-increment addressing modes
- They may support circular buffer addressing
 - Efficient implementation of delay lines
- They may support zero-overhead loops

Comparison

Frequency Response Comparison

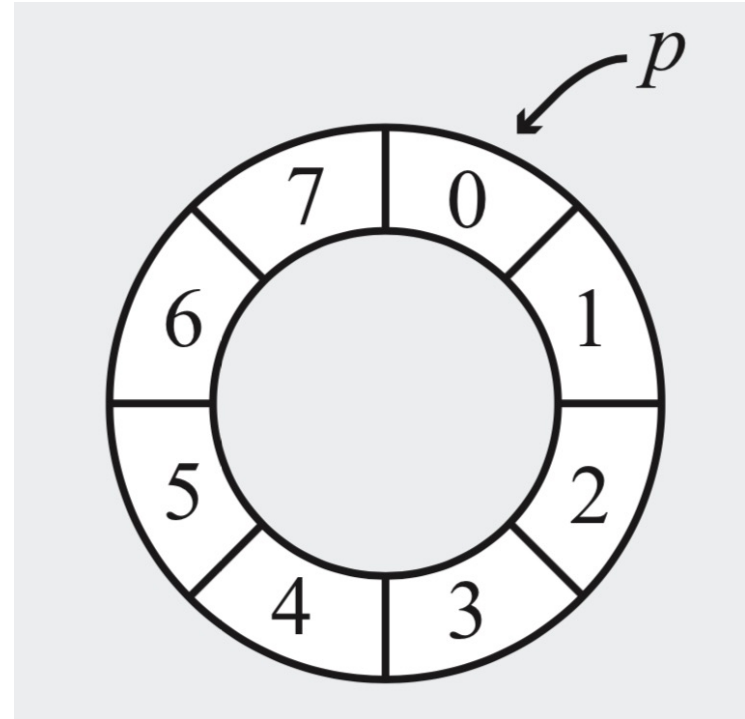


Digital Filter Critique

- The filter pole is at about $\frac{1}{4}$ of the sampling rate
 - We have only 4 samples of the impulse response
 - This makes the FIR filter simple: only 4 taps
 - This also degrades the filter performance
- We may be able to improve the filter performance some by using a different design technique
 - The filter coefficients would differ
- A higher sampling rate with respect to the filter corner frequency could also help

FIR Filter Delay Implementation

➤ Circular Buffer



Programmable Logic Controller (PLC)

- A microcontroller system for industrial automation
 - Continuous operation
 - Hostile environments
 - originated as replacements for control circuits using electrical relays to control machinery

- PLCs are frequently programmed using ladder logic
 - This notation was developed to specify logic constructed with relays and switches

GPUs

- A graphics processing unit (GPU) is a specialized processor designed especially to perform the calculations required in graphics rendering.
- Most used for Gaming (earlier days)
- Common programming language: CUDA

Parallelism vs Concurrency

- Embedded computing applications typically do more than one thing “at a time.”
- Tasks are said to be “concurrent” if they conceptually execute simultaneously
- Tasks are said to be “parallel” if they physically execute simultaneously
 - Typically multiple servers at the same time

Imperative Language

- Non-concurrent programs specify a *sequence* of instructions to execute.
- Imperative Language: expresses a computation as a sequence of operations
 - Example: C, Java
- How to write concurrent programs in imperative language?
 - Thread Library

Dependency – Sequential Consistency

- No dependency between lines 3 and 4

```
double pi, piSquared, piCubed;  
pi = 3.14159;  
piSquared = pi * pi ;  
piCubed = pi * pi * pi;
```

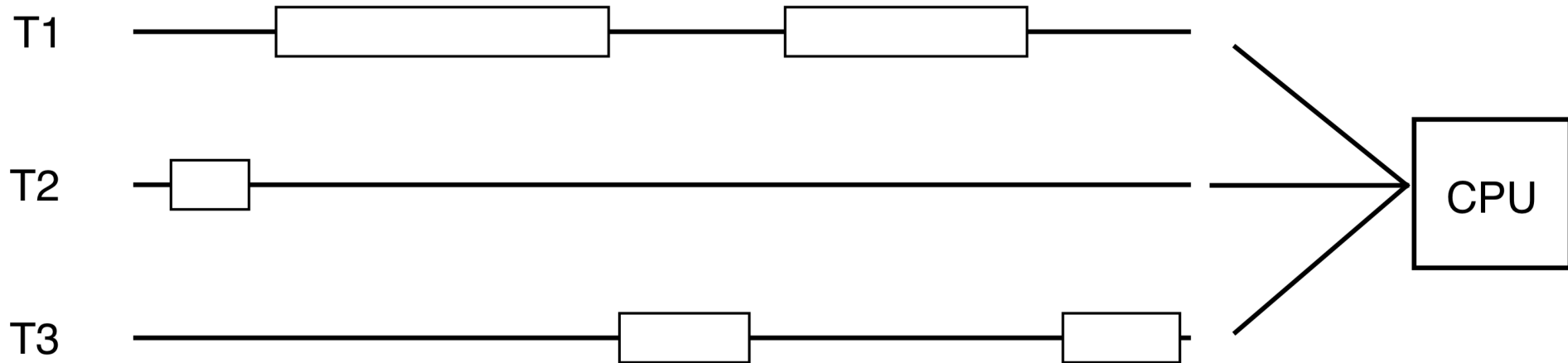
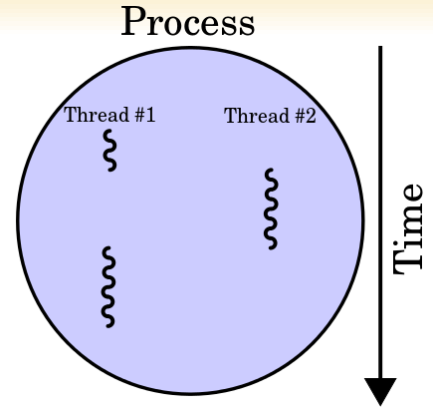
- Line 4 is dependent on Line 3

```
double pi, piSquared, piCubed;  
pi = 3.14159;  
piSquared = pi * pi ;  
piCubed = piSquared * pi;
```

Thread Mapping on Processor

➤ OS Dependent Scheduler

- Static Mapping
- Basic Lowest Load (fill in Round Robin fashion)
- Extended Lowest Load



Performance Improvement

- Various current architectures seek to improve performance by finding and exploiting potentials for parallel execution
 - This frequently improves processing throughput
 - It does not always improve processing latency
 - It frequently makes processing time less predictable
- Many embedded applications rely on results being produced at predictable regular rates
 - Embedded results must be available at the right time

Parallelism

- Temporal Parallelism – Pipelining
- Spatial Parallelism –
 - Superscalar (instruction and data level parallelism)
 - VLIW
 - Multicore

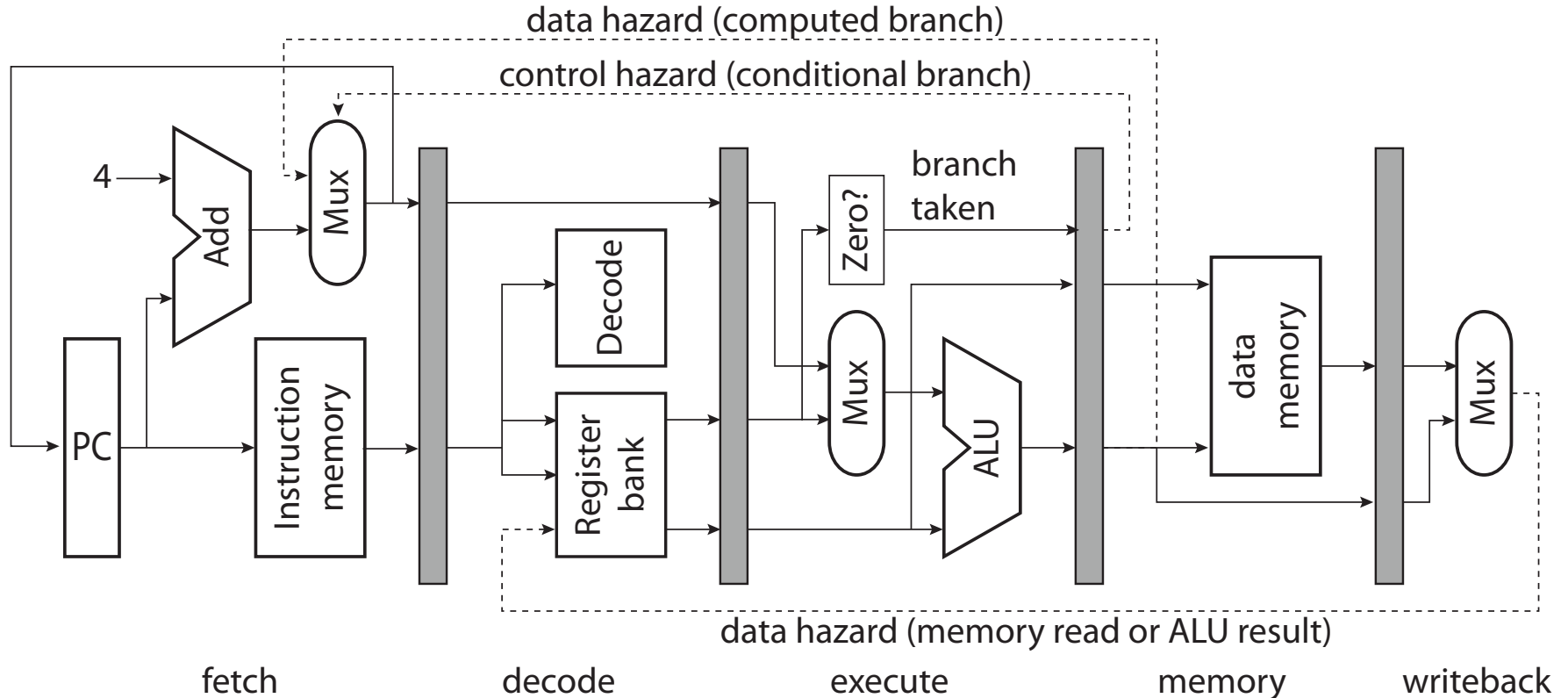
RISC and CISC Architectures

- CISC – Complex Instruction Set Computer
 - Multi-clock complex instructions
- RISC – Reduced Instruction Set Computer
 - Simple instructions that can be executed within one cycle

5 Cycles of RISC Instruction Set

- Instruction fetch cycle (IF)
 - Fetch instruction from memory pointed by PC, then increment PC
- Instruction decode/register fetch cycle (ID)
 - Decode the instruction
- Execution/effective address cycle (EX)
 - ALU operates on the operands
- Memory access (MEM)
 - Load/Store instructions
- Write-back cycle (WB)
 - Register-Register ALU instruction

Pipelining in RISC



Simple RISC Pipeline

| | Clock number | | | | | | | | |
|---------------------|--------------|----|----|-----|-----|-----|-----|-----|----|
| Instruction number | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| Instruction i | IF | ID | EX | MEM | WB | | | | |
| Instruction $i + 1$ | | IF | ID | EX | MEM | WB | | | |
| Instruction $i + 2$ | | | IF | ID | EX | MEM | WB | | |
| Instruction $i + 3$ | | | | IF | ID | EX | MEM | WB | |
| Instruction $i + 4$ | | | | | IF | ID | EX | MEM | WB |

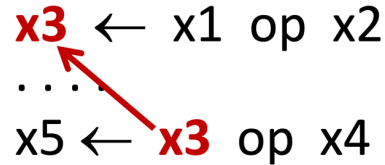
Pipelining Hazard

- Data Hazard (RAW (*read after write*) , WAW (*write after write*) , WAR (*write after read*))
 - Pipeline bubble (no op)
 - Interlock
 - Out-of-order Execution
- Control Hazard
 - Out-of-order Execution
 - Speculative Execution

Data Hazard

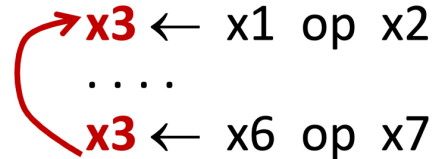
- RAW (*read after write*)— j tries to read a source before i writes it, so j incorrectly gets the *old* value.

x3 ← x1 op x2
...
x5 ← **x3** op x4



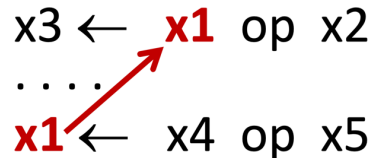
- WAW (*write after write*)— j tries to write an operand before it is written by i .

→ **x3** ← x1 op x2
...
x3 ← x6 op x7



- WAR (*write after read*)— j tries to write a destination before it is read by i , so i incorrectly gets the *new* value.

x3 ← **x1** op x2
...
x1 ← x4 op x5



Data Hazard

RAW

store R1, 0(SP)
load R4, 0(SP)

WAR

load R4, 0(SP)
store R1, 0(SP)

WAW

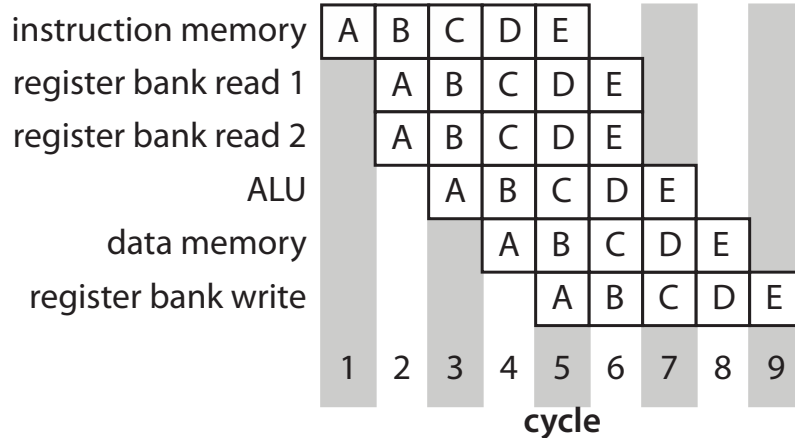
store R1, 0(SP)
store R4, 0(SP)

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|-----------------|---|---|---|---|---|---|---|---|---|
| store R1, 0(SP) | F | D | X | M | W | | | | |
| load R4, 0(SP) | | F | D | X | M | W | | | |

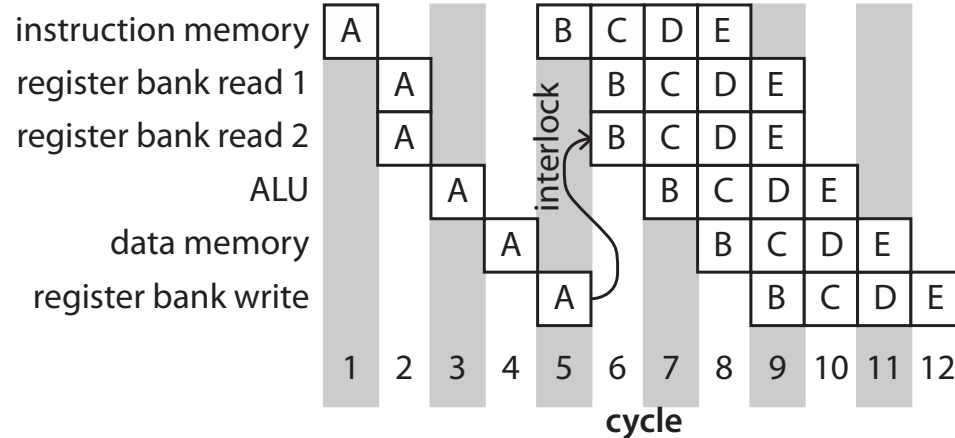
Interlocks

instruction B reads a register written by instruction A

hardware resources:



hardware resources:



Reservation Table

Reservation Table with Interlocks

CISC

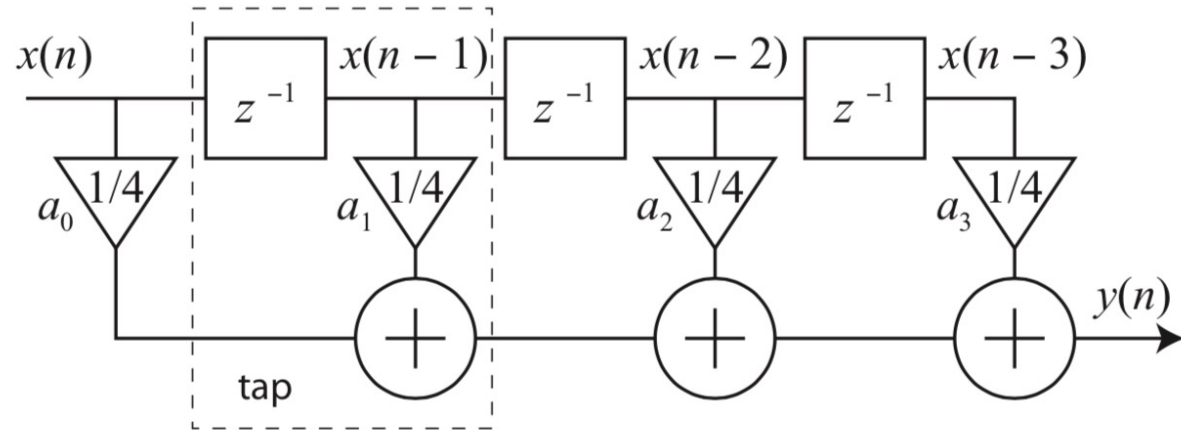
- DSPs are typically CISC machines
- Instructions support
 - FIR filtering
 - FFTs
 - Viterbi decoding

FIR Filter Implementation

- z^{-1} is unit delay
- Suppose $N = 4$ and $a_0 = a_1 = a_2 = a_3 = 1/4$.
- Then for all $n \in \mathbb{N}$,

$$y(n) = (x(n) + x(n-1) + x(n-2) + x(n-3))/4 .$$

- **Multiply-Accumulate**



CISC Instruction

- Texas Instruments TMS320c54x family of DSP processors
- Code
 - RPT numberOfTaps - 1
 - MAC *AR2+, *AR3+, A
- RPT: zero overhead loops
- MAC : Multiply accumulate
 - $a := a + x * y$
 - AR2, AR3 are registers
 - A is the Accumulator

Symmetric FIR Filter

- Coefficients of FIR Filter is often symmetric
 - $N = 2, a_i = a_{N-i-1}$

$$y(n) = \sum_{i=0}^{N-1} a_i x(n-i) \longrightarrow y(n) = \sum_{i=0}^{(N/2)-1} a_i (x(n-i) + x(n-N+i+1))$$

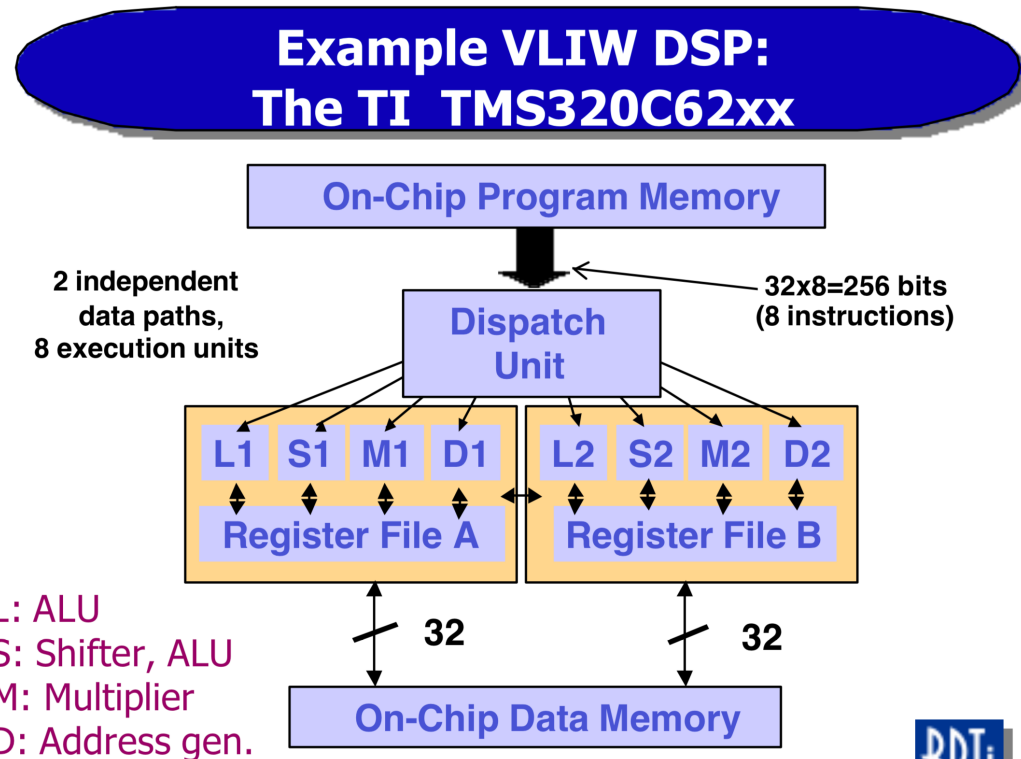
- If hardware has two ALUs, it can be used
- Requires half the time

Example DSP Library from TI:

http://processors.wiki.ti.com/index.php/C674x_DSPLIB 37

VLIW Instruction Set

- Used for DSP, other Embedded Applications
- Multiple independent instructions per cycle, packed into single large "instruction word" or "packet"



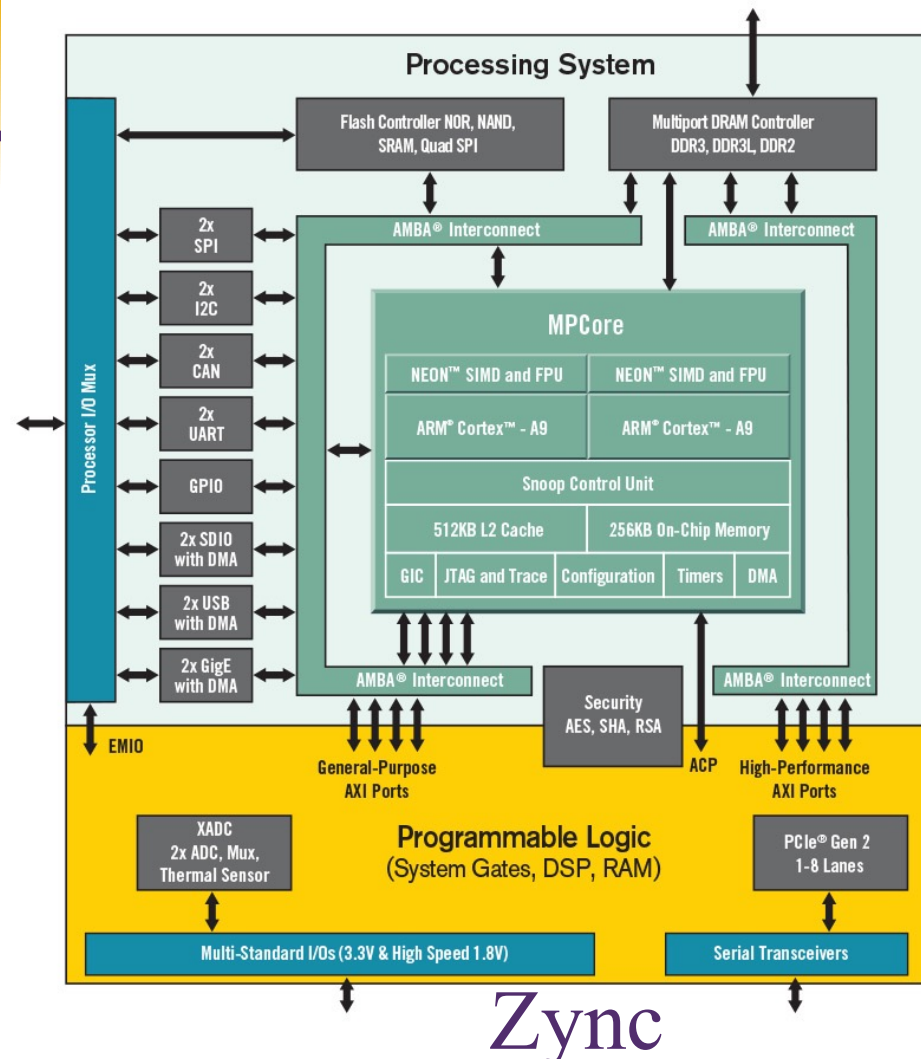
© 1999 Berkeley Design Technology, Inc.

Multicore Architecture

- Combination of several processors in a single chip
- Real-time and Safety critical tasks can have dedicated processors
- Heterogeneous multicore
 - CPU and GPUs together

FPGAs

- Field Programmable Gate Arrays
 - Set of logic gates and RAM blocks
 - Reconfigurable / Programmable
 - Precise timing
- System on Chip design

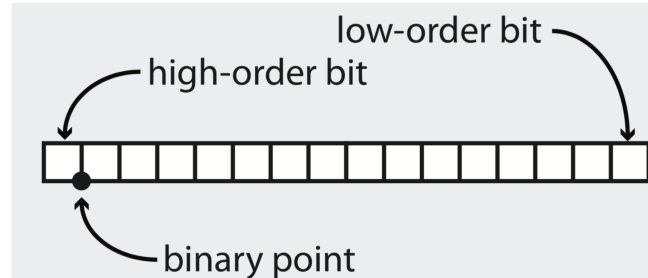


Bits to represent data

- Range and Resolution Tradeoff
 - More bits
 - Better precision
 - More flip-flops
 - Fewer bits
 - Less precision
 - Fewer flip-flops → lower footprint, lower power
- Fixed Point Representation
 - Simulation required for the complete design for dynamic range of parameters

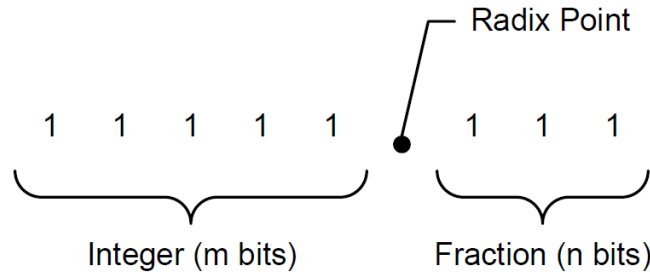
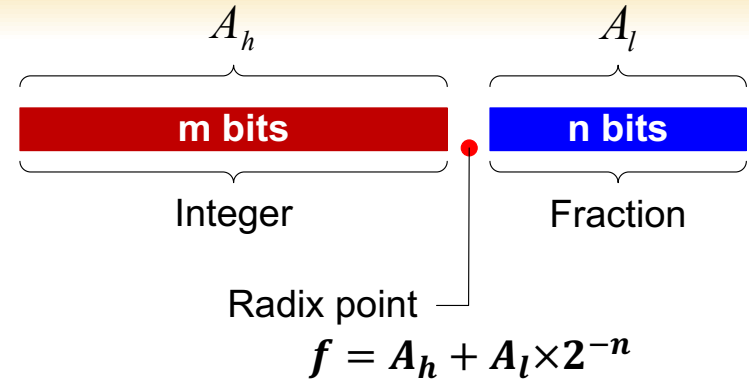
Fixed and Floating Point Numbers

- Programs may use float or double
- Many embedded processors do not have floating point arithmetic hardware
- Conversion required, which makes it slow
- Imaginary Binary Point is considered for computation
 - Binary point separates bits
 - Decimal point separates digits
- Format $x.y$ representation indicates
 - x bits left & y bits right of binary point



Fixed Point Numbers

- **01101.101**₂
- = $1 \times 2^3 + 1 \times 2^2 + 1 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-3}$
- = 13.625



$$\begin{aligned}
 10101.101_2 &= A_h + A_l \times 2^{-3} \\
 &= 21 + 5 \times 2^{-3} \\
 &= 21.625
 \end{aligned}$$

Unsigned Fixed Point Representation

- **Example:** Convert $f = 3.141593$ to unsigned fixed-point UQ4.12 format.
- Calculate $f \times 2^{12} = 12867.964928$
- Round the result to an integer, $round(12867.964928) = 12868$
- Convert the integer to binary: $12868 = 11_0010_0100_0100_2$
- Organize into UQ4.12: $0011.0010_0100_0100_2$
- Final result in Hex: **0x3244**
- Error: $\frac{12868}{2^{12}} - f = -8.5625 \times 10^{-6}$

Signed Fixed Point Representation



$$A = -1 \times b_{N-1} \times 2^{N-1} + \sum_{i=0}^{N-2} (b_i \times 2^i)$$

$$f = \frac{A}{2^n}$$

where $N = m + n + 1$

Signed Fixed Point Representation

- **Example:** Convert $f = -3.141593$ to signed fixed-point Q3.12 format.
- Calculate $f \times 2^{12} = -12867.964928$
- Round the result to an integer, $round(-12867.964928) = -12868$
- Convert the absolute integer to binary: $12868 = 11_0010_0100_0100_2$
(Note that the integer is represented in two's complement.)
- Make the result into 16 bits: **0011_0010_0100_0100**₂
- Find the two's complement: **1100_1101_1011_1100**₂
- Final result in Hex: **0xCDBC**
- Error: $-\frac{12868}{2^{12}} - f = 8.5625 \times 10^{-6}$

Range and Resolution

- Range of Unsigned UQm.n (m+n bits)
 - Unsigned integer $\rightarrow [0, 2^{m+n} - 1]$
 - Unsigned fixed point $\rightarrow [0, 2^{m+n} - 1] \times 2^{-n} = [0, 2^m - 2^{-n}]$
- Range of Signed Fixed point Qm.n (m+n+1 bits)
 - Range of signed integers: $[-2^{m+n}, 2^{m+n} - 1]$
 - Range of Signed fixed point number: $[-2^{m+n}, 2^{m+n} - 1] \times 2^{-n} = [-2^m, 2^m - 2^{-n}]$
- Resolution/Precision (UQm.n and Qm.n) = 2^{-n}

Addition and Subtraction

Addition

Assume UQ16.16

$$f_C = f_A + f_B$$

$$\begin{cases} I_A = f_A \times 2^{16} \\ I_B = f_B \times 2^{16} \\ I_C = f_C \times 2^{16} \end{cases} \quad \longrightarrow \quad \begin{cases} f_A = I_A \times 2^{-16} \\ f_B = I_B \times 2^{-16} \\ f_C = I_C \times 2^{-16} \end{cases}$$

$$\begin{aligned} \longrightarrow \quad f_C &= f_A + f_B \\ &= I_A \times 2^{-16} + I_B \times 2^{-16} \\ &= (I_A + I_B) \times 2^{-16} \end{aligned}$$

$$\longrightarrow \quad I_C \times 2^{-16} = (I_A + I_B) \times 2^{-16}$$

$$\longrightarrow \quad I_C = I_A + I_B$$

Subtraction

$$f_C = f_A - f_B$$

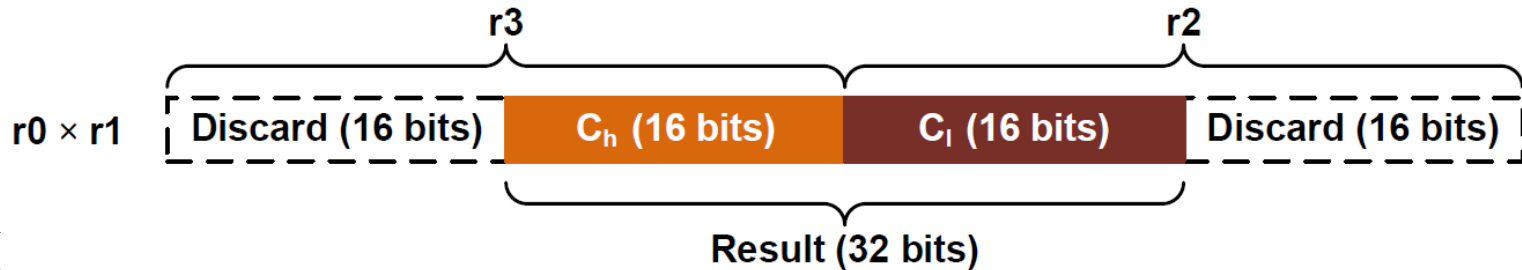
$$I_C = I_A - I_B$$

Multiplication

$$\begin{aligned} f_C &= f_A \times f_B \\ &= (I_A \times 2^{-16}) \times (I_B \times 2^{-16}) \\ &= (I_A \times I_B) \times 2^{-32} \end{aligned}$$

$$I_C = (I_A \times I_B) \times 2^{-16}$$

$$f_C = I_C \times 2^{-16}$$



Law of Conservation of Bits

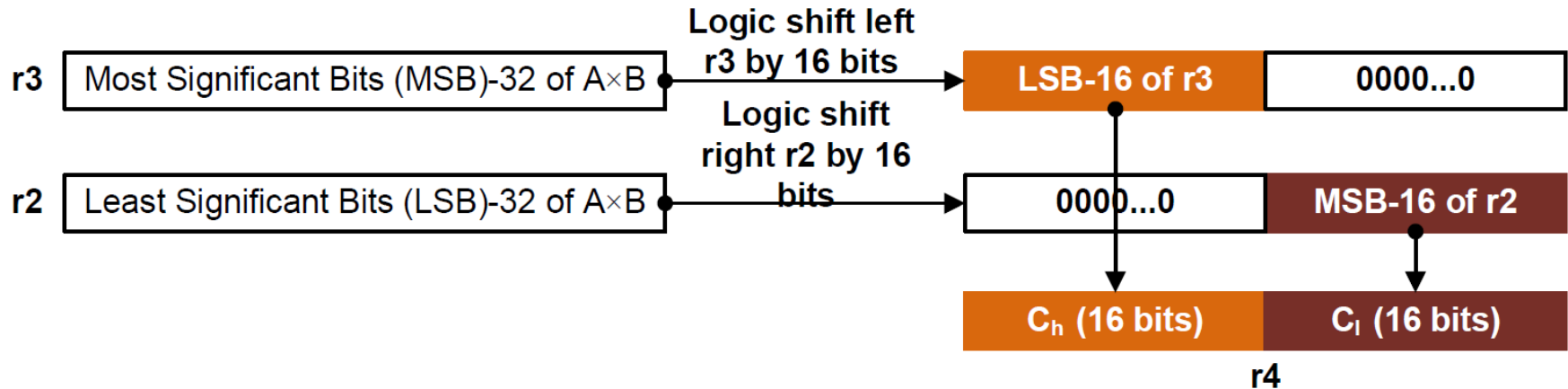
- When multiplying two x -bit numbers with formats $n.m$ and $p.q$, the result has format $(n + p).(m + q)$
- Processors might support full precision multiplications
- Finally need to convert x -bits to data register

Fixed Point Multiplication

$$\begin{aligned}
 f_C &= f_A \times f_B \\
 &= (I_A \times 2^{-16}) \times (I_B \times 2^{-16}) \\
 &= (I_A \times I_B) \times 2^{-32}
 \end{aligned}$$

$$f_C = I_C \times 2^{-16}$$

$$I_C = (I_A \times I_B) \times 2^{-16}$$



Programmers need to guard

- Overflow – since higher order bits are discarded
- Underflow – due to lower order bits being discarded
- Truncation –if bits are chosen before operation
- Rounding – rounds to nearest full precision after operation