Lecture 7: Embedded Processors

Types of Processors

- ✓ Microprocessors and Microcontrollers
- ✓ DSP Processors
- ✓ Graphics Processors

Parallelism

- ✓ Parallelism vs Concurrency
- ✓ Pipelining
- ✓ Instruction-Level Parallelism
- ✓ Multicore Architectures

Dr. Suneel Kommuri

Suneel.Kommuri@xjtlu.edu.cn

Syllabus & Teaching Plan

Week Number and/or Date	Lecture/Seminar/ Field Trip/Other	Topic/Theme/Title	Pre-reading
Week 1	*Lecture 1*	*Introduction to Embedded Systems*	
Week 2-3	Lecture and Labs 2-3	C Programming	A programming guide with exercises will be provided
Week 4-6	Lecture and Labs	Embedded C programming	A guide to embedded C will be provided
Week 7	Tutorial	Improving programming for CW submission	
Week 8	Lecture 7	Types of Processors	
Week 9	Lecture 8	Memory Architectures	
Week 10	Lecture 9	Interfacing with the physical world	
Week 11	Lecture 10	Practical problems with programming	

Introduction

- In general-purpose computing the variety of instruction set architectures is limited
  - Intel x86 architecture is dominant
- There is no such dominance in embedded computing
- When deployed in a product embedded processors have a dedicated function
  - Making them more specialized has many benefits
    - Lower power consumption
    - Include specialist hardware to perform operations too costly on a general purpose computer (image analysis)
    - Small size
    - Overall reduction in cost

Types of Processors

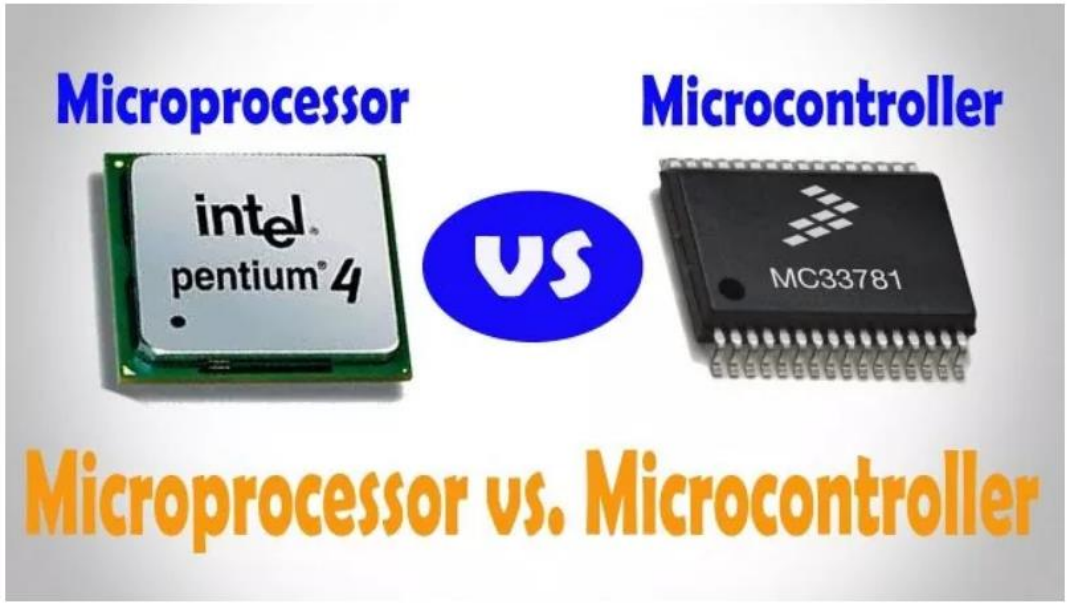
- As a consequence of huge variety of embedded applications, huge variety of processors are in use.
- They range from small, slow, inexpensive, low power devices to high performance, specialist devices.
- **Microcontrollers**
- **DSP Processors**
- **Graphic Processors**

Microcontrollers

- A microcontroller (µC) is a small computer on a single integrated circuit.
- It consists of a relatively simple CPU combined with peripheral devices (memories, I/O devices, timers, etc).
  - More than half the CPUs sold in the world are microcontrollers.
- A microcontroller is a small and low-cost computer built for the purpose of dealing with specific tasks.
- Microcontrollers are mainly used in products that require a degree of control to be exerted by the user.

Microprocessor vs microcontroller

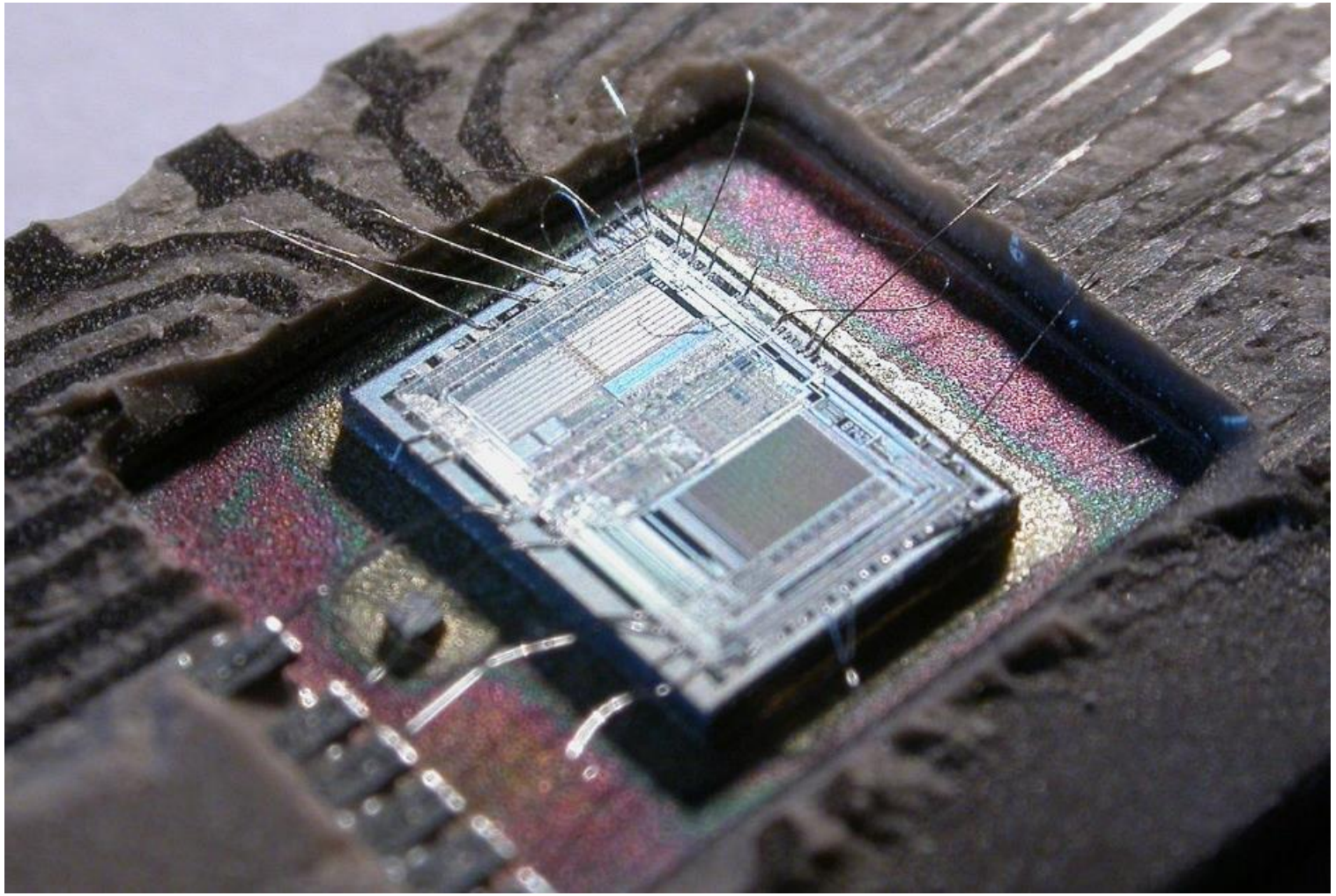
- Microprocessor is an IC which has only the CPU inside them
  - i.e. only the processing powers such as Intel's Pentium 1,2,3,4, core 2 duo, i3, i5 etc.
  - These microprocessors don't have RAM, ROM, and other peripheral on the chip.
  - A system designer has to add them externally to make them functional.
  - Applications of microprocessor includes Desktop PC's, Laptops, notepads etc.
- But this is not the case with Microcontrollers.
- A Microcontroller has a CPU, in addition with a fixed amount of RAM, ROM and other peripherals all embedded on a single chip.
  - At times it is also termed as a mini computer or a computer on a single chip.
  - Today different manufacturers produce microcontrollers with a wide range of features available in different versions.



Microprocessor	Microcontroller
• CPU is stand-alone, RAM, ROM, I/O, timer are separate	• CPU, RAM, ROM, I/O and timer are all on a single chip
• Designer can decide on the amount of ROM, RAM and I/O ports.	• Fix amount of on-chip ROM, RAM, I/O ports
• Expansive	• For applications in which cost, power and space are critical
• Versatility	• Not Expansive
• General-purpose	• Single-purpose

Can they be differentiated on cost?

- Comparing microcontroller and microprocessor in terms of cost is not justified.
  - Undoubtedly a microcontroller is far cheaper than a microprocessor.
  - However, a microcontroller cannot be used in place of microprocessor and using a microprocessor is not advised in place of a microcontroller as it makes the application quite costly.
  - Microprocessor cannot be used stand alone.
    - They need other peripherals like RAM, ROM, buffer, I/O ports etc and hence a system designed around a microprocessor is quite costly.



The die from an Intel 8742, 8-bit microcontroller that includes a CPU running at 12 MHz, 128 bytes of RAM, 2048 bytes of EPROM, and I/O in the same chip.

DSP Processors

- Many embedded applications may be required to do a lot of signal processing
- Signals of physical systems are collected at a sample rate
  - A motion control application may read position or location information from sensors at sample rates range from a few Hz to 100's of Hz.
  - Audio signals 8 kHz (telephony) to 44 kHz (audio CD's)
  - Ultrasonics (medical imaging) may sample sound signals at higher rates
  - Video (25 or 30 Hz) for consumer devices (frames and pixels)
  - Software-defined radio applications – 100 kHz to several GHz

DSP Characteristics

- Signal processing applications all share certain characteristics
  - Deal with large amounts of data
  - The data may represent a signal in time (wireless radio signal), space (images) or both (video and radar).
  - Typically, perform sophisticated mathematical operations (filtering, machine learning)
- Processors that are designed to specifically support numerically intensive signal processing applications are called Digital Signal Processors (DSPs).

FIR filtering

- To gain some insight into the structure of a DSP we shall consider the structure of a typical signal processing algorithm.

- A canonical signal processing algorithm used in most applications is **finite impulse response (FIR)** filtering.

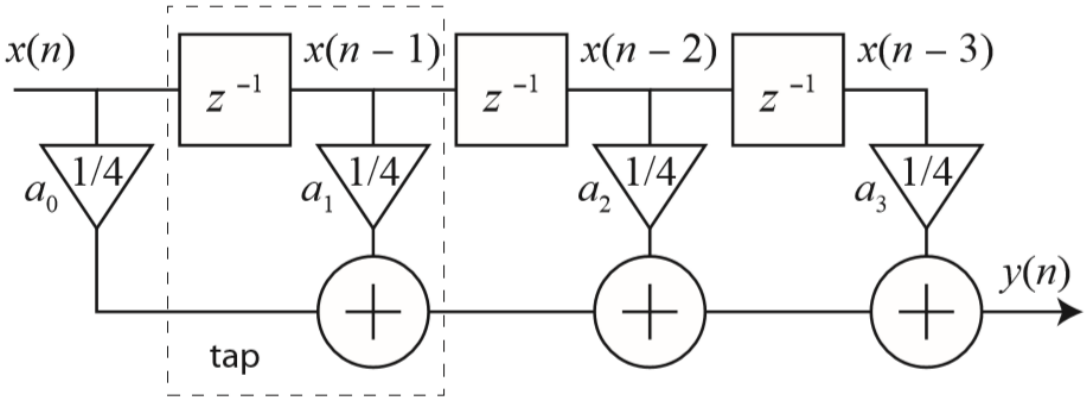
  - In its simplest form, we can consider an input signal x that is a infinite sequence of numerical values (by infinite we mean very long !).

  - For each input value x(n), the FIR filter must compute an output value y(n) according to the formula:

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

where N is the length of the FIR filter and a_i are called the tap values.

Example: Tapped delay line implementation



Structure of tapped delay line implementation of the FIR filter. For each $n \in \mathbb{N}$, each component consumes one input value from each input path and produces one output value on each output path.
The boxes z^{-1} are unit delays and the triangles multiply their inputs by a constant (0.25 in this case).

Suppose N=4 and $a_0=a_1=a_2=a_3=1/4$, then for all n that are natural numbers:

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

Each output is the average of the four most recent inputs. Input values come in from the left and propagate down the **delay line**, which is tapped after each delay element.

The rate at which the input values x(n) are provided and must be processed is the sample rate. You must know the sample rate and N to determine the number of arithmetic operations to be computed per second.

Example

- An FIR filter is provided with samples at a rate of 1 MHz, and that N = 32.
- At what rate must the outputs be computed at?

  (1 MHz) — Each output requires 32 multiplications and N-1 (31 additions)
- How many arithmetic operations are required per second to implement this application?

  63 million arithmetic operations/sec
- To sustain the computation rate:

  arithmetic hardware be fast enough, also data in & out of memory be fast

Example 2

- Consider the two dimensional FIR filter, $y(i,j) = \sum_{n=-N}^{N} \sum_{m=-M}^{M} a_{n,m} x(i-n, j-m)$

- $N = 1$ and $M = 1$ are the minimal values for useful filters.

- Each pixel of the output image will need 9 multiplications and 8 additions.

- Consider a color image with 3 channels (RGB) with resolution 1080x1920. How many multiplications are required per image?

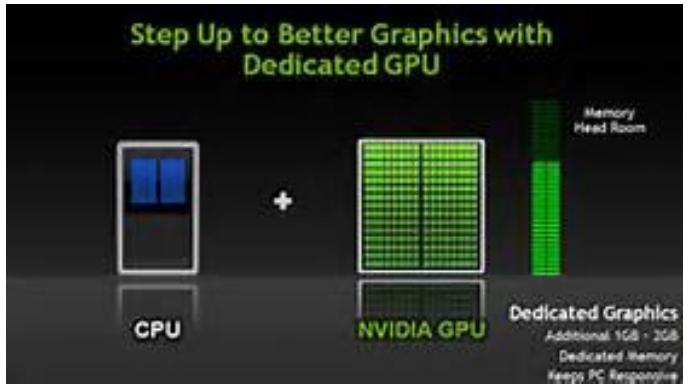
  1080x1920x3x9=55,987,200 multiplications and additions.

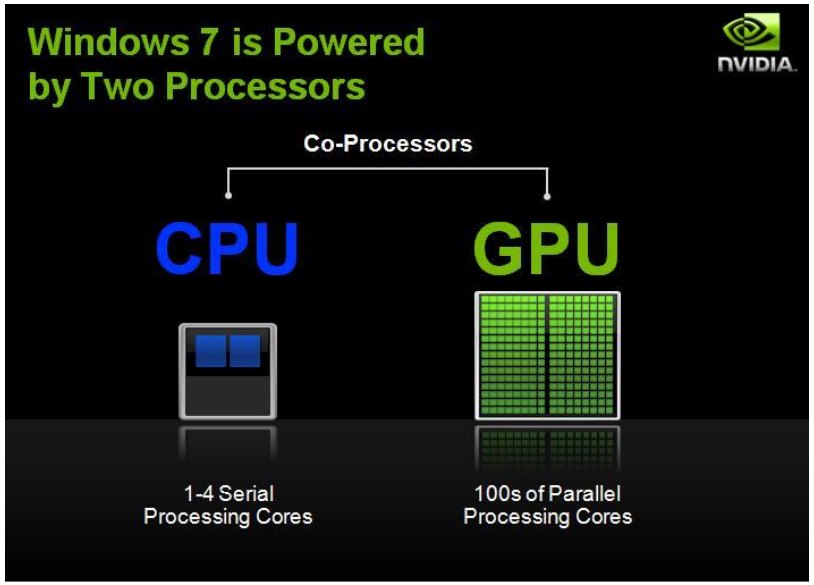
- How many per second if there are 30 frames per second?

  55,987,200x30=1,679,616,000 multiplications/sec and additions.

Graphics Processors

- A graphics processing unit (GPU) is a specialized processor designed to perform the calculations required for graphics rendering.
  - Date back to the 1970's
  - Modern GPUs support 3D graphics, shading and ditital video.
    - Intel, NVIDIA and AMD
    - Important in high end gaming Laptops !
- Some embedded applications (games) are a good match for GPUs but they have evolved towards mode general programming machines. Often used in instrumentation.
  - Very power hungry.





Parallelism

Parallelism

- ✓ Parallelism vs Concurrency
- ✓ Pipelining
- ✓ Instruction-Level Parallelism
- ✓ Multicore Architectures

Parallelism vs Concurrency

- Most processors today provide various forms of parallelism
  - These mechanisms affect the timing of the execution of a program
  - Embedded system designers have to understand them.

- **Concurrent**

  A computer program is concurrent if different parts of the program *conceptually* execute simultaneously

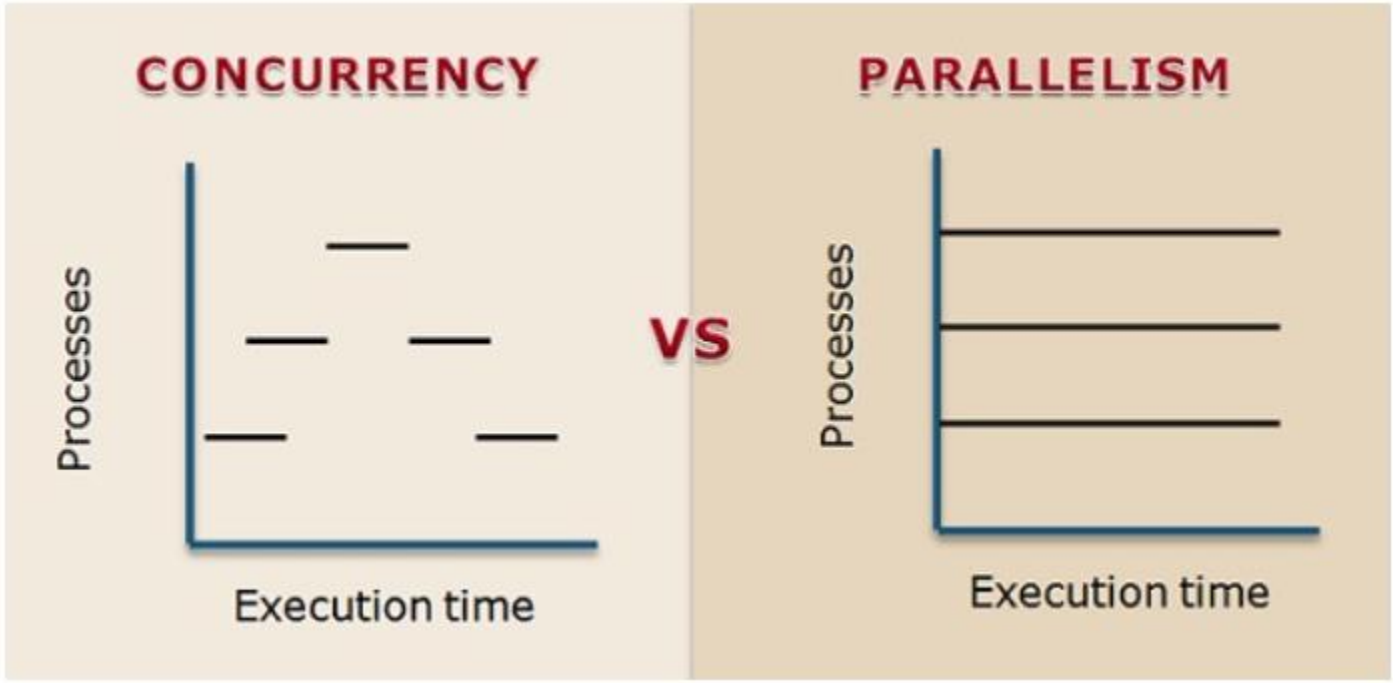
- **Parallel**

  A program is parallel if different parts of the program *physically* execute simultaneously on distinct hardware.

Parallelism vs Concurrency

Concurrency is when certain tasks can start, run, & complete in overlapping time periods

Parallelism is when tasks literally run at the same time



Non-concurrent programs

- Specify a *sequence* of instructions to execute.

- **Imperative language**: if a programming language expresses a computation as a sequence of operations (C is an imperative)
  - to write concurrent programs using C: use **thread library**
  - thread library uses provided by OS and/or the hardware
  - Java is imperative language that directly supports threads

- The correct execution of a program requires that the instructions are executed in the specified sequence.
  - often possible to execute instructions in parallel

Example

```
double x, xSquared, xCubed;
x = 3;
xSquared = x * x;
xCubed = xSquared * x;
```

In this example each statement must be completed in sequence.

We must know “x’ to find xSquared and we cannot find xCubed until we know xSquared.

```
double x, xSquared, xCubed;
x = 3;
xSquared = x * x;
xCubed = x * x * x;
```

Do you need to know xSquared to find xCubed?

What does this mean?

Statements are independent and executed in parallel

Concurrency in Embedded Systems

- Embedded programs interact with physical processes and many activities progress at the same time.
- An embedded program often needs to monitor and react to multiple concurrent sources of stimulus and simultaneously control multiple output devices.
- Embedded programs are almost always concurrent programs.
  - Timeliness matters: actions in the physical world should be performed on right time
- Imperative and concurrent programs can be executed both sequentially and in parallel

Parallelism in Hardware

- The application does not (necessarily) demand multiple activities execute simultaneously
  - it demands that things be done very quickly

- Of course, many other applications will combine both forms of concurrency, arising from parallelism and from application requirements.

- Here we will focus on hardware approaches to deliver parallelism
  - ✓Pipelining
  - ✓Instruction-level parallelism
  - ✓Multicore architectures

- Later we will look at memory systems. These strongly influence how parallelism is handled.

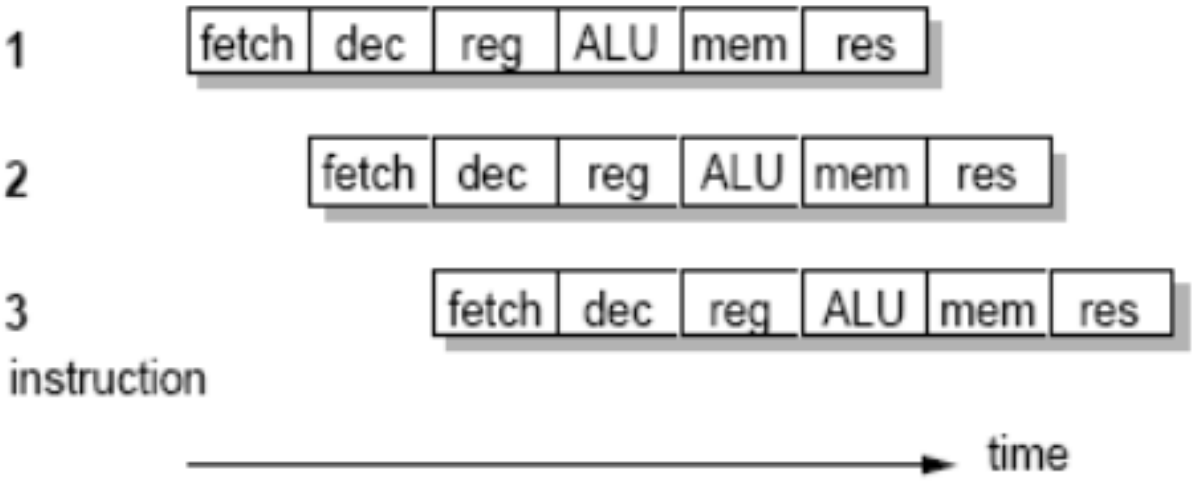
Pipelining

- ✓ The process of fetching next instruction while the current instruction is being executed – Each instruction can be broken down into steps

- ❖ Fetch instruction from memory (fetch)
- ❖ Decode instruction (dec)
- ❖ Access operands from register bank (reg)
- ❖ Combine operands to form either a result or memory address (ALU)
- ❖ Access memory to read or write data (mem)
- ❖ Write result into the register bank (res)

- ✓ Not all of these steps will be needed for some instructions.
- ✓ These 6 steps can occur concurrently in a 6 stage pipeline.

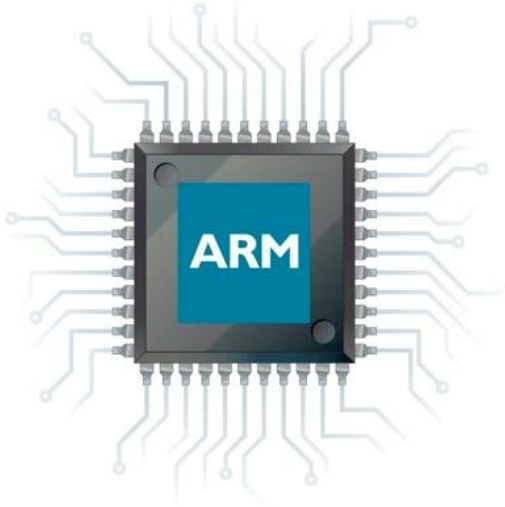
Pipelined instruction execution



Pipelining allows more than one instruction to be executed at a time; but in different phases. In the diagram above

1. instruction 3 is being fetched
2. whilst inst. 2 is being decoded
3. & operands for inst. 1 are being accessed from a register

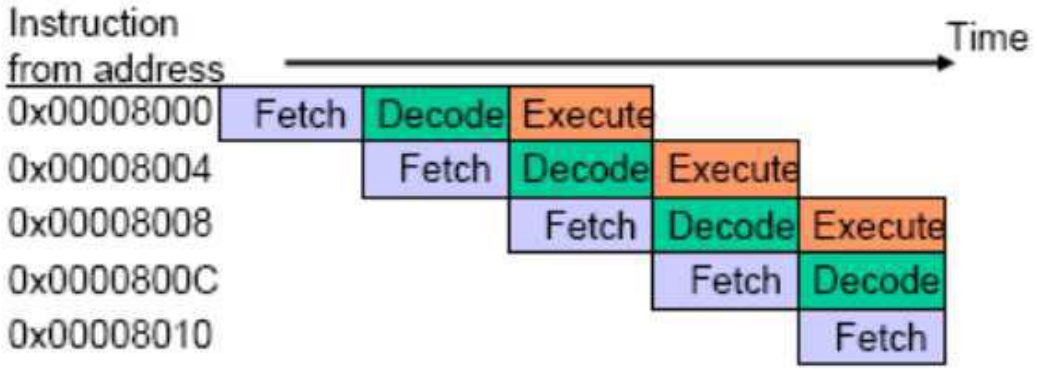
ARM Instruction Pipelines



- ✓ ARM processor developed by **A**dvanced **R**ISC (reduced instruction set computer) **M**achines.
- ✓ Extensively used in consumer electronic devices (smartphones, tablets etc.).
  - – Require fewer transistors, enables smaller size
- ✓ For the same basic speed of transistor operation, an *n* stage instruction pipeline allows the microprocessor to execute up to *n* times as many instructions in a given time.
- ✓ The ARM7 core has a three stage pipeline whereas the ARM9 core has a five stage pipeline.

3 Stage Pipeline – ARM7

- In a three stage pipeline (e.g. ARM7) the CPU can simultaneously execute an instruction, decode the next instruction and fetch the next but one instruction.



Fetch: In this stage the ARM processor fetches instruction from the memory

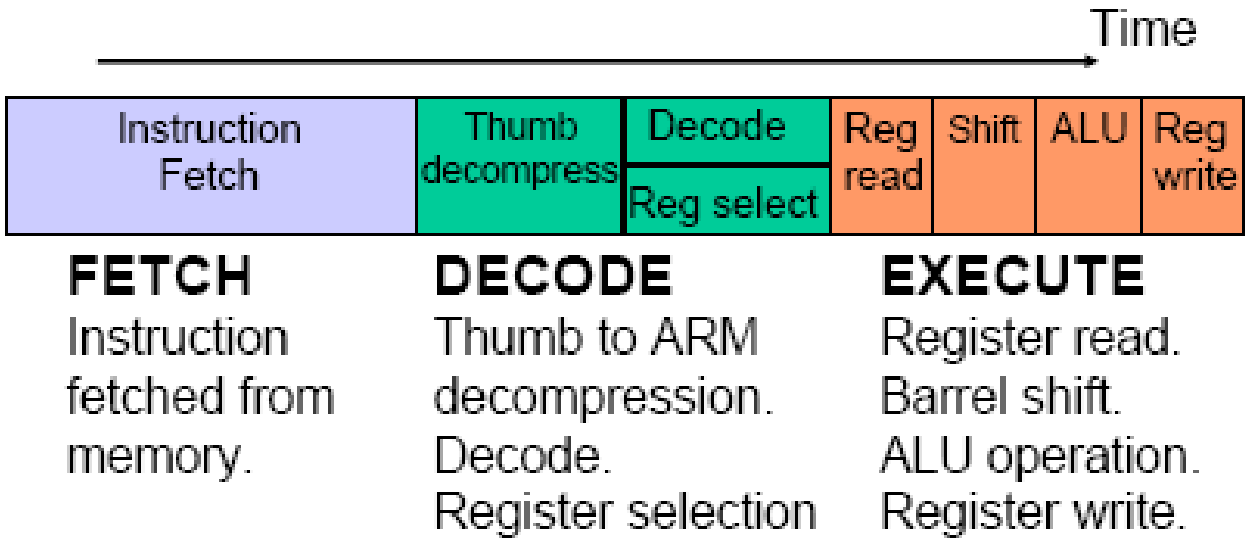
Decode: In this stage recognizes the instruction that is to be executed

Execute: Processor processes the instruction and writes the result back to desired register

ARM7 3 Stage Pipeline: detail

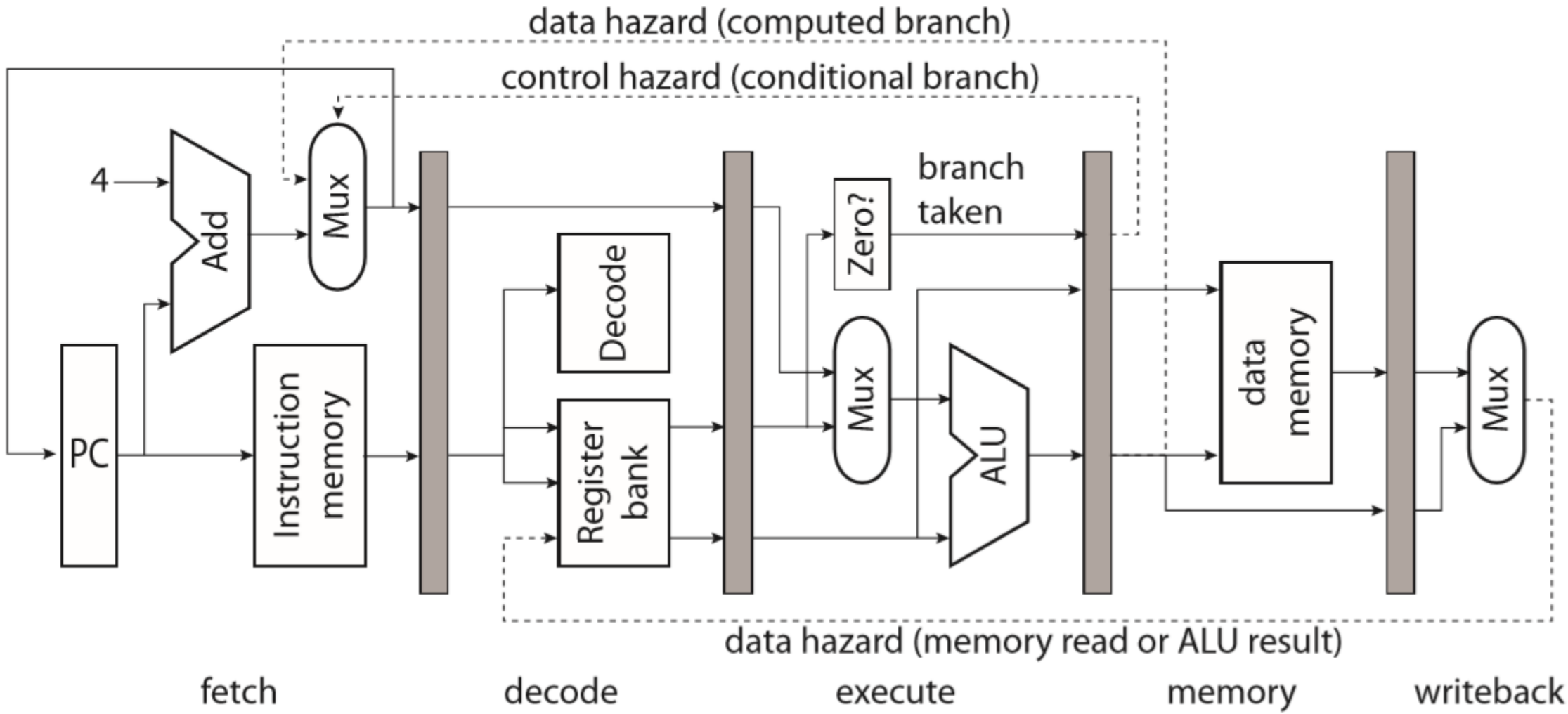
- In each stage of the ARM7 pipeline several things happen; normally consecutively:

- If 3 stages of execution are overlapped, will achieve higher speed of execution (exists in ARM 7 processor)



5 Stage Pipeline

✓Later processors use a 5 stage pipeline (this is more common now!)



Summary: FETCH and DECODE

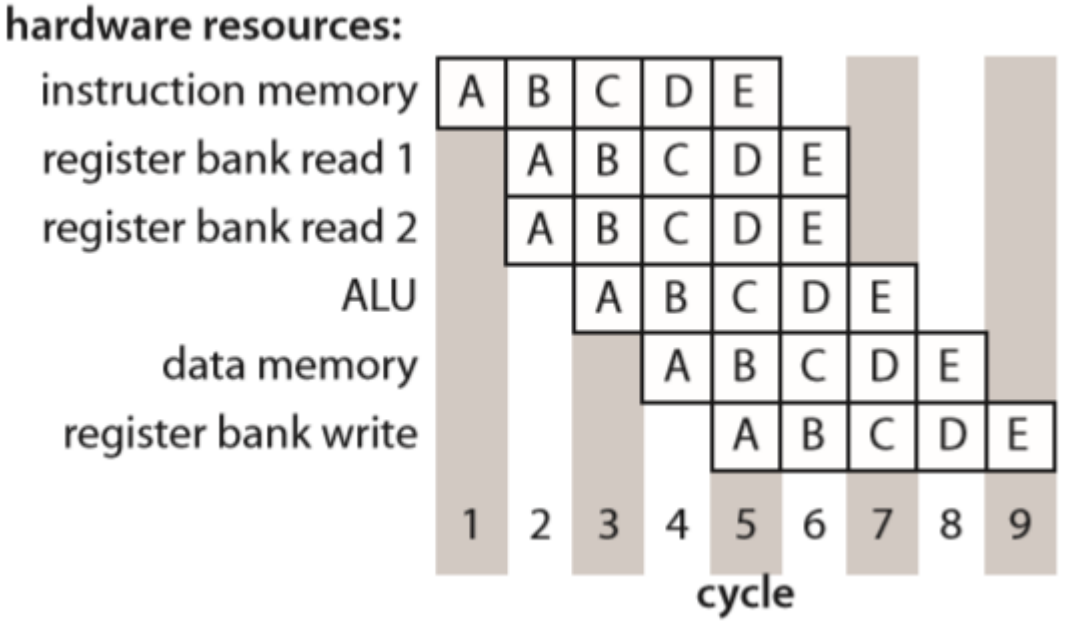
- ✓In the fetch stage the program counter provides an address to the instruction memory.
  - The instruction memory provides encoded instructions (for ARM 7 32 bit)
  - In the fetch stage the program counter is incremented by 4 bytes to become the address of the next instruction (unless a conditional branch instruction is providing an entirely new address for the Program counter)
- ✓The decode pipeline stage extracts register addresses from the 32-bit instruction and fetches the data in the specific registers from the register bank.

Summary: Execute, memory and writeback

- The execute pipeline stage operates on the data fetched from the registers (or on the PC for a computed branch) using the ALU.
- The memory pipeline stage reads or writes to a memory location given by a register.
- The writeback pipeline stage stores results in a register file.

Reservation table for the 5 stage pipeline

- ✓ In cycle 5, E is being fetched while D is reading from the register bank, while C is using the ALU, while B is using reading from or writing data memory, while A is writing results to the register bank.



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

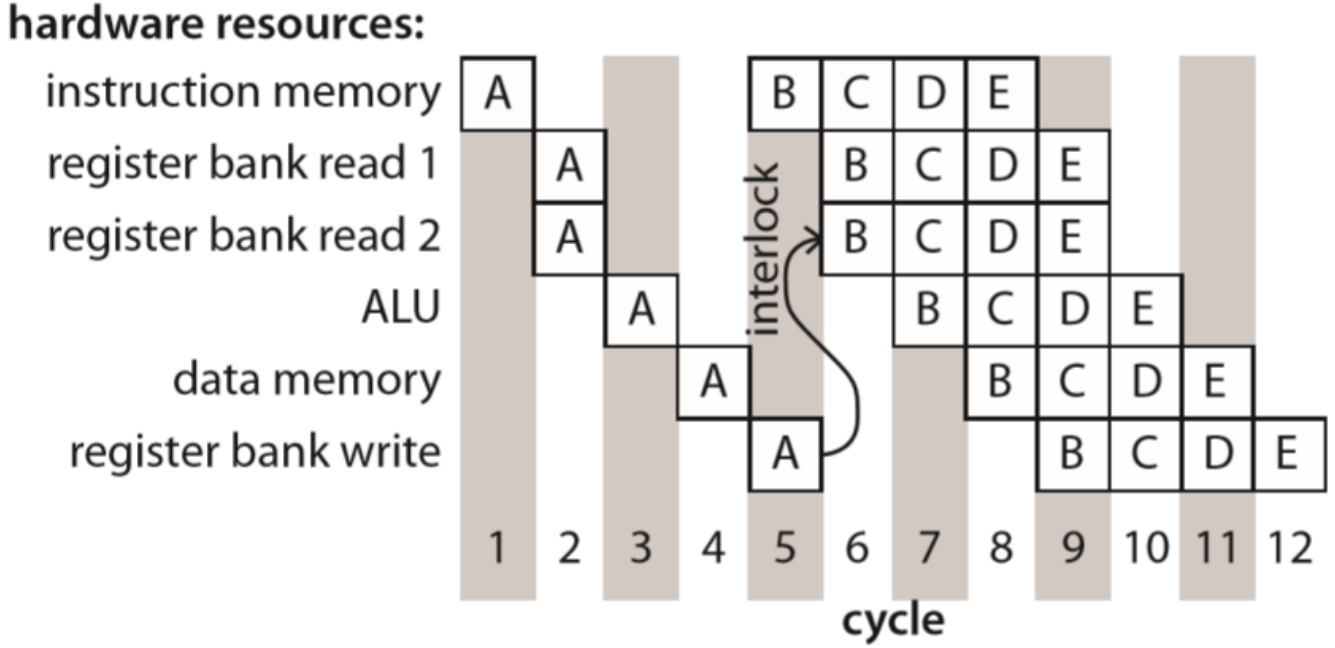
Pipeline Hazards – data hazard

- ✓ Many techniques have been developed by programmers in a variety ways to handle pipeline hazards

- ✓ The simplest technique is known as an **explicit pipeline**.
  - The pipeline hazard is documented and the compiler deals with it.
  - For example where B reads a register written by A, the compiler will insert three **no-op** instructions (do nothing) between A and B.
    - to ensure the write occurs before the read
  - **No-op** instructions form a pipeline bubble

Pipeline Hazards – data hazard

- ✓ Another method is to provide **interlocks**
  - In this technique the instruction decode hardware will detect the hazard and delay the execution of B until A has completed the writeback stage (delayed by 3 cycles).
  - Can be reduced to two cycles – complex forwarding logic



Pipeline Hazards – data hazard

- ✓ **Out-of-order execution**
  - A hardware is provided that detects a hazard but instead of simply delaying the execution of B, proceeds to fetch C, and if C does not read registers written by A or B, and does not write registers read by B, then proceeds to execute C before B.
  - This further reduces the number of pipeline bubbles

Branch

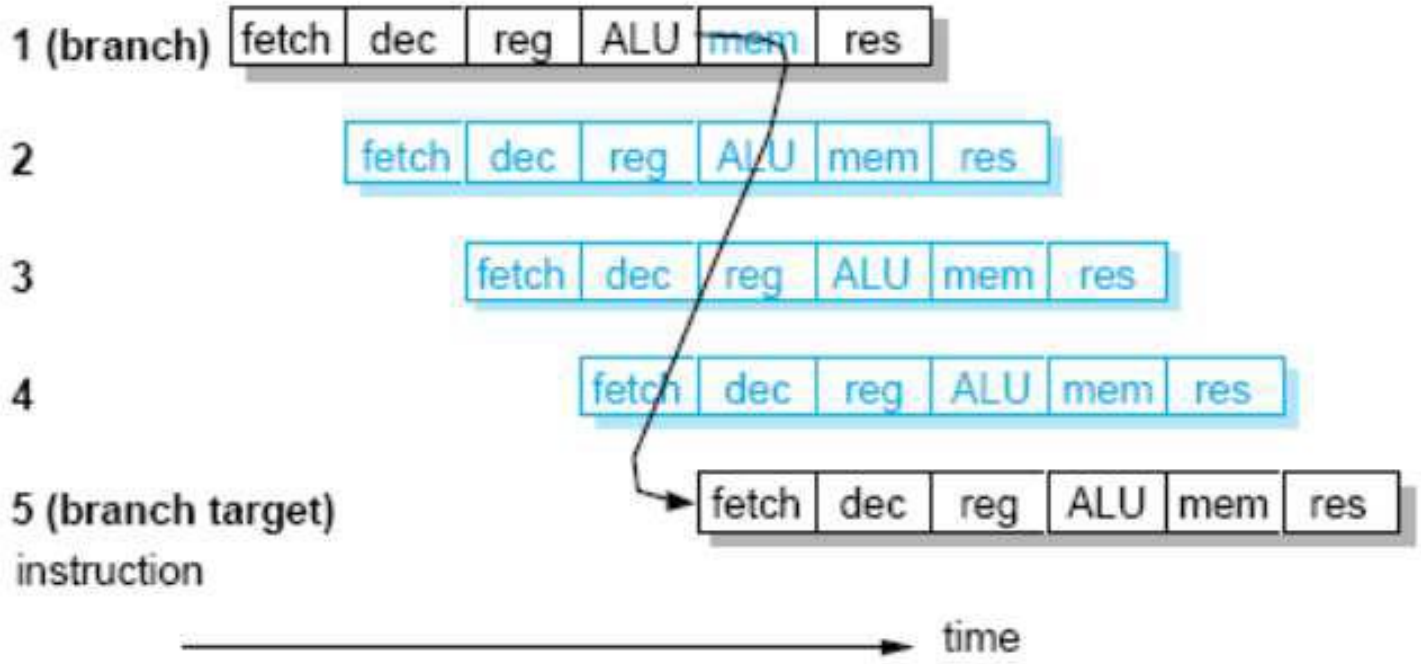
- Pipelines only work properly when instructions are stored in memory at consecutive addresses (0x00008000, 8001, 8002…) or regularly spaced addresses (0x00008000, 8004, 8008..).

- Generally this is the case except when the instruction fetched is a branch (or jump).
    - The purpose of a branch is to take the program to a different part of memory.

- For a pipeline this means losing a number of instructions that have been fetched but do not need to be executed.

- This results in a ‘pipeline flush’.

Pipeline flush on branch



Pipeline Hazards – control hazard

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

Pipeline Hazards – control hazard

- **Delayed Branch**
  - Documents the branch will be taken some time after it is encountered. The compiler will ensure that the instructions that follow are harmless.
- **Interlock**
  - Hardware solution – insert pipeline bubbles as needed
- **Speculative execution**
  - The hardware estimates whether a branch is likely to be taken and begins executing instructions it expects to execute.
  - If its expectation is not met then it undoes any side effect.

Why do pipeline hazards cause problems?

- Timing may be very important and the techniques used can introduce variability in the timing of execution of an instruction sequence.
- Analysis of the timing of a program is difficult when there is a deep pipeline with very elaborate forwarding and speculation.
- Explicit pipelines and delayed branches usage:
  - For example DSP processors are often used in applications where precise timing is essential. They normally use explicit pipelines.
- Out-of-order and speculative execution are common in general-purpose processors, where precise timing is not so critical.

Parallelism in Hardware

- Achieving high performance requires parallelism in the hardware

- Thus takes two broad forms:

  - Instruction Level parallelism (ILP)
  - Multicore architectures

Instruction Level Parallelism (ILP)

- A processor supporting ILP is able to perform multiple independent operations in each instruction cycle.

- There are 4 major forms of ILP:

  - ✓ CISC instructions
  - ✓ Subword parallelism
  - ✓ Superscalar
  - ✓ VLIW

Complex Instruction Set Computer (CISC)

- Processor with complex instructions – CISC
- The philosophy behind such processors is different from that of RISC (reduced instruction set computers)

When do we use CISC machines?

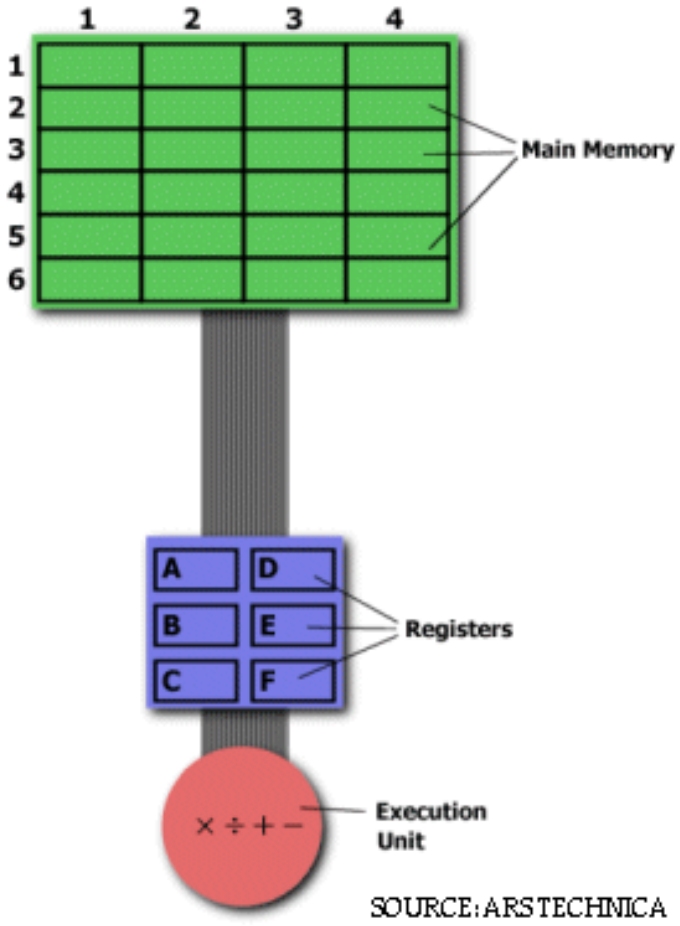
- DSPs are CISC machines, include instructions supporting FIR filtering
- In fact to qualify as a DSP a processor must be able to perform FIR filtering in one instruction cycle per tap.
- Disadvantages: Extremely challenging for a compiler to make optimal use of such instruction set
  - DSPs used with code libraries written and optimized in assembly language

RISC V CISC: An Example

Multiplying Two Numbers in Memory

On the right is a diagram representing the storage scheme for a generic computer. The main memory is divided into locations numbered from (row) 1: (column) 1 to (row) 6: (column) 4.

The execution unit is responsible for carrying out all computations. However, the execution unit can only operate on data that has been loaded into one of the six registers (A, B, C, D, E, or F). Let's say we want to find the product of two numbers - one stored in location 2:3 and another stored in location 5:2 - and then store the product back in the location 2:3.



CISC Approach

- The primary goal of CISC architecture is to complete a task in as few lines of assembly as possible.
- This is achieved by building processor hardware that is capable of understanding and executing a series of operations. For this particular task, a CISC processor would come prepared with a specific instruction (we'll call it "MULT").
- When executed, this instruction loads the two values into separate registers, multiplies the operands in the execution unit, and then stores the product in the appropriate register. Thus, the entire task of multiplying two numbers can be completed with one instruction:

  MULT 2:3, 5:2

- MULT is what is known as a "complex instruction." It operates directly on the computer's memory banks and does not require the programmer to explicitly call any loading or storing functions.
- One of the primary advantages of this system is that the compiler has to do very little work to translate a high-level language statement into assembly. Because the length of code is relatively short, very little RAM is required to store instructions.
- The emphasis is put on building complex instructions directly into the hardware.

RISC Approach

- RISC processors only use simple instructions that can be executed within one clock cycle. Thus, the "MULT" command described above could be divided into three separate commands:

- "LOAD," which moves data from the memory bank to a register,

- "PROD," which finds the product of two operands located within the registers,

- and "STORE," which moves data from a register to the memory banks.

- In order to perform the exact series of steps described in the CISC approach, a programmer would need to code four lines of assembly.

RISC V CISC (Advantages)

- At first, the RISC approach may seem like a much less efficient way of completing the operation. Because there are more lines of code, more RAM is needed to store the assembly level instructions.

- The compiler must also perform more work to convert a high-level language statement into code of this form

✓Because each instruction requires only one clock cycle to execute, the entire program will execute in approximately the same amount of time as the multi-cycle "MULT" command.

✓RISC "reduced instructions" require less transistors of hardware space than the complex instructions, leaving more room for general purpose registers. Because all of the instructions execute in a uniform amount of time (i.e. one clock), pipelining is possible.

Subword Parallelism

Example 1:

- If we consider a colour of a pixel it can be represented by 3 bytes in RGB format.
- Each of the RGB bytes has a value ranging from 0 to 255, representing the intensity of the corresponding colour.
- It is not efficient to use a 64-bit ALU to process a single 8-bit number.

- To support such data types some processors support **subword parallelism,** where a wide ALU is divided into narrow slices enabling arithmetic (or logical) operations on smaller words.

Subword Parallelism

Example 2:

- Intel introduced subword parallelism into Pentium processor (MMX)
- MMX instructions divide 64-bit datapath into slices (8 bits)
- Supports identical operations on multiple bytes of image pixel data
- Enhances the performance of image applications
- Many processors including DSPs support subword parallelism

- **Vector processor**: instruction set includes operations on multiple data elements simultaneously

  – Subword parallelism is a form of vector processing

Superscalar

- In Superscalar processors, the hardware can simultaneously dispatch multiple instructions to distinct hardware units when it detects that this will not change the behavior of the program.

- Superscalar processors are rarely (if ever) used for embedded systems.
  - Execution times are very difficult to predict and not be repeatable.

- Instead processors intended for embedded applications will use VLIW architectures.

Very Large Instruction Word (VLIW)

- VLIW processors include multiple function units
- Instead of dynamically determining which instructions can be executed simultaneously, each instruction specifies what each function unit should do in a particular cycle.
- Effectively a VLIW instruction set combines multiple independent operations into a single instruction.
- Multiple operations are executed simultaneously on distinct hardware
- Up to the compiler to ensure simultaneous operations are independent

Multicore Architectures

- **Multicore machine** is combination of several processors on single chip
- For embedded applications, multicore architectures have a significant potential advantage over single-core in that real-time and safety critical tasks can have a dedicated processor.
  - The reason is multicore architectures are now popular in mobile phones is that radio and speech processing are hard real-time functions that have a considerable computational load.
  - In such multicore architectures, user applications cannot interfere with real-time functions.