EMBEDDED PROCESSORS

- In the context of embedded systems, **concurrency plays a part that is much more central than merely improving performance**
- **Embedded programs interact with physical processes**, and in the physical world, **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** that affect the physical world
- **Embedded programs are almost always concurrent programs**, and concurrency is an intrinsic part of the logic of the programs. It is not just a way to get improved performance
- Indeed, **finishing a task earlier is not necessarily better than finishing it later**
  - **actions** performed in the physical world **often need to be done at the right time** (neither early nor late)
  - Picture for example an engine controller for **a gasoline engine**. Firing the **spark plugs** earlier is most certainly not better than firing them later. They **must be fired at the right time**

EMBEDDED PROCESSORS

- Just as imperative programs can be executed sequentially or in parallel, **concurrent programs can be executed sequentially or in parallel**
- **Sequential execution** of a concurrent program is done typically today by a **multitasking operating system**, which **interleaves** the execution of **multiple tasks** in a **single sequential stream of instructions**
- Of course, the **hardware** may **parallelize** that execution if the **processor** has a **multi-issue** or **VLIW architecture**
    - Hence, a **concurrent program** may be **converted** to a **sequential stream** by an **operating system** and back to **concurrent program** by the **hardware**, where the latter translation is done to improve performance
- These **multiple translations** greatly **complicate** the problem of **ensuring** that **things occur** at the **right time**

EMBEDDED PROCESSORS

- Parallelism in the hardware exists to improve performance for computation-intensive applications
- From the programmer’s perspective, concurrency arises as a consequence of the hardware designed to improve performance, not as a consequence of the application problem being solved
  - The application does not (necessarily) demand that multiple activities proceed simultaneously, it just demands that things be done very quickly.
  - Of course, many interesting applications will combine both forms of concurrency, arising from parallelism and from application requirements.

EMBEDDED PROCESSORS

Application software execution requires several stages of processing at development and execution time

Level	Input	Processing	Output	Time
High-level Language	High-level programming language sources	Translation to assembly by high-level language compilers	Target Assembly (ARM, RISC-V, PowerPC, MIPS, 68k, x86-32, x86-64, …) sources	
Assembly Language	Target Assembly (ARM, RISC-V, PowerPC, MIPS, 68k, x86-32, x86-64, …) sources	Translation to Machine Language by assemblers	Machine Language encoded in object files	Development
Target System (specific hardware)	Machine Language encoded in object files	Configuration for target, binding	Executables and libraries	
Instruction Set Architecture (ISA)	Executables and libraries	Direct execution, translation to microinstructions or both by CPUs	Direct execution: **Data (operands, addresses) + digital logic control signals** Translation to microinstructions:	
Microarchitecture	Microinstruction streams	Execution of microinstruction streams	**Data (operands, addresses) + digital logic control signals**	Execution
Digital logic	**Data (operands, addresses) + digital logic control signals**	Hardware (datapath) configured by control signals processes data (operands, addresses)	Application results	

Digital logic often **include multiple** similar **functional units** (e.g. ALUs, MACs, memory controllers) for **parallel execution**. Besides hardware parallelism, **several techniques** are used to **improve performance in general-purpose CPUs**: long pipelines, out-of-order and speculative executions, branch prediction. These techniques **are avoided in embedded processors** (Why?).

EMBEDDED PROCESSORS

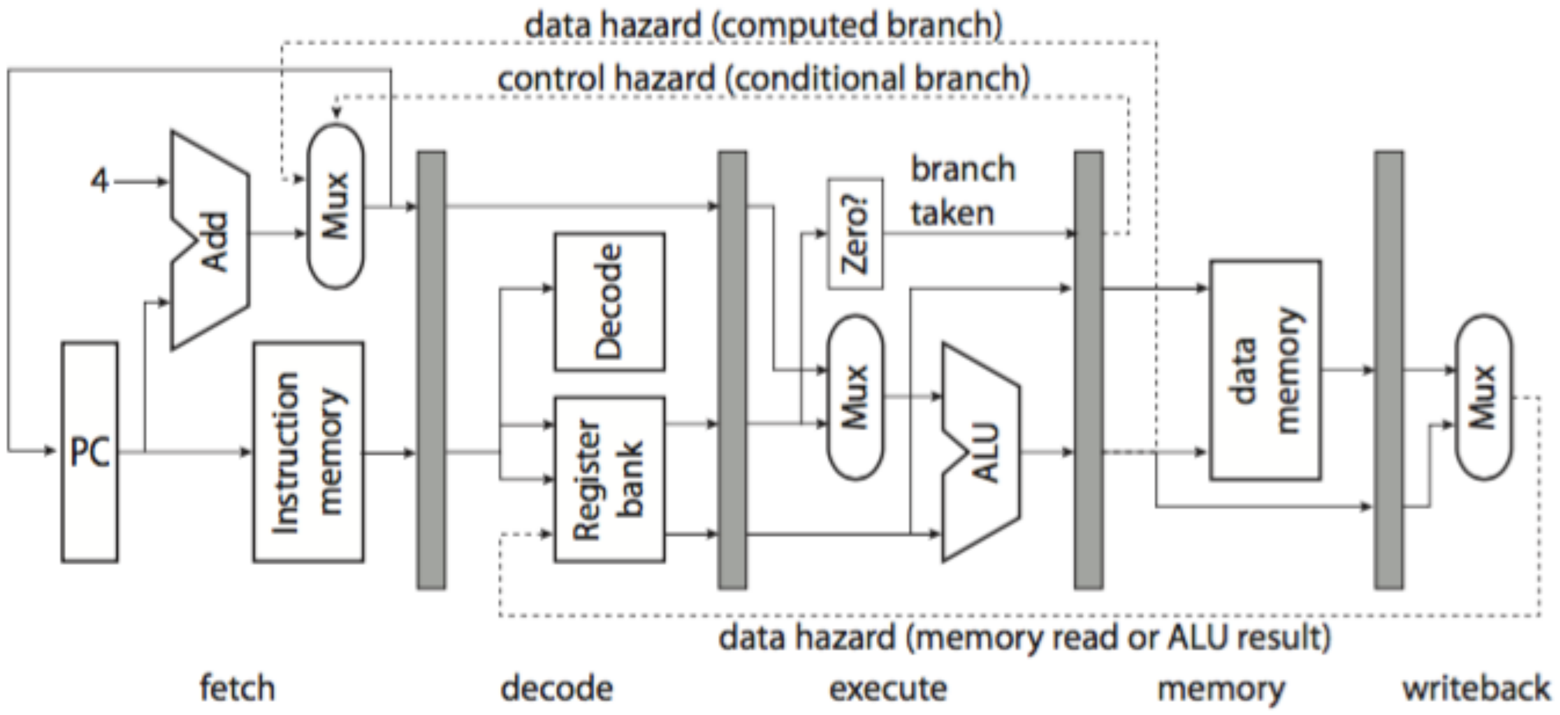
The hardware level can be realized through software specifications too

Level	Input	Processing	Output	Time
High-level Language	Hardware description languages (HDL): Verilog, VHDL, …	Translation to Register Transfer Language (RTL) by HDL compilers	RTL sources	Development (Possibly at execution time too, in the case of FPGAs)
RTL	RTL sources	Synthesis	Raw Binary File (*.rbf), ASICs	
System	Raw Binary File (*.rbf), ASICs	Configuration and integration of modules	Hardware Systems for application specific or general purpose computing (with ISA)	

Hardware configuration is becoming part of common practice in embedded development. Even general-purpose CPUs exist that include configurable hardware (FPGA). **HDLs include** the notions of **time**, **events** and **concurrency** in their formalism.

EMBEDDED PROCESSORS

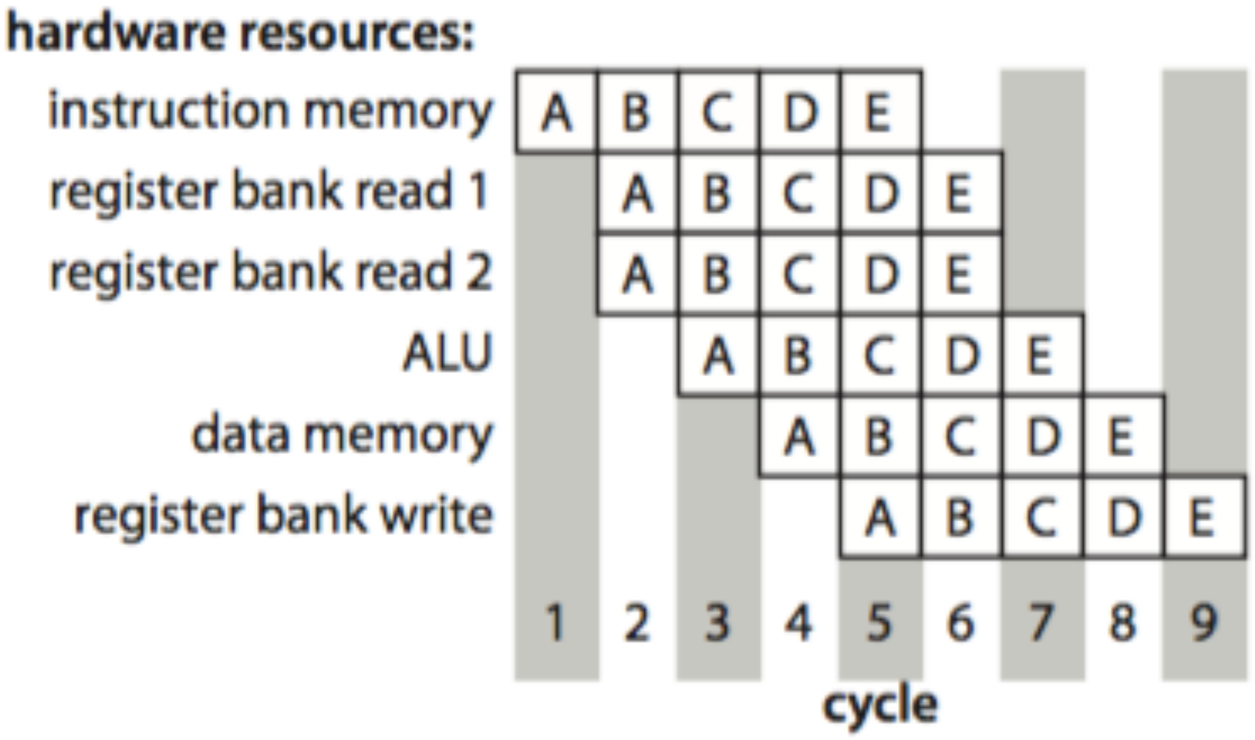
- **Pipelining. A simple five-stage pipeline for a 32-bit machine**



E. A. Lee and S. A. Seshia, Introduction to Embedded Systems

EMBEDDED PROCESSORS

- **Pipelining. A simple five-stage pipeline for a 32-bit machine Reservation table**

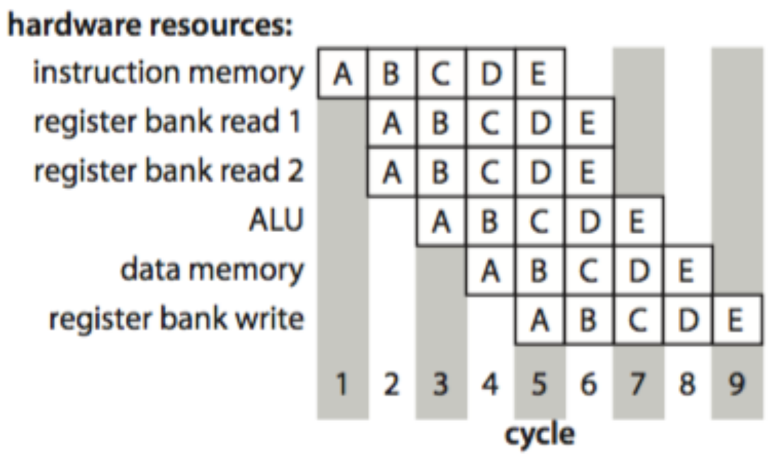


hardware resources:	1	2	3	4	5	6	7	8	9
instruction memory	A	B	C	D	E				
register bank read 1		A	B	C	D	E			
register bank read 2		A	B	C	D	E			
ALU			A	B	C	D	E		
data memory				A	B	C	D	E	
register bank write					A	B	C	D	E

cycle

EMBEDDED PROCESSORS

- **Addressing data hazards: explicit pipeline**
  - The pipeline hazard is simply documented, and the programmer (or compiler) must deal with it
  - For the example where B reads a register written by A, the compiler may insert three no-op instructions (which do nothing) between A and B to ensure that the write occurs before the read
  - These no-op instructions form a pipeline bubble that propagates down the pipeline

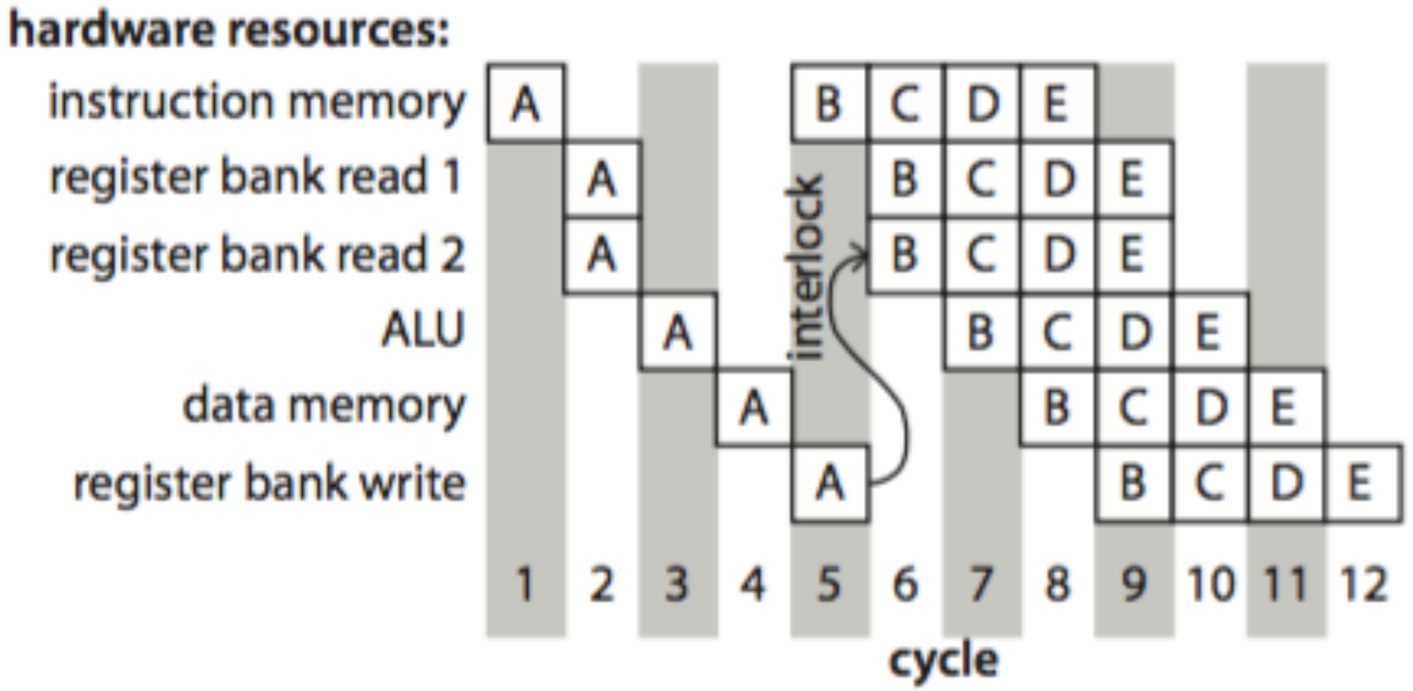


EMBEDDED PROCESSORS

- **Addressing data hazards: Interlocks**
  - The 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
  - For this pipeline, B should be delayed by three clock cycles to permit A to complete
  - This can be reduced to two cycles if slightly more complex forwarding logic is provided, which detects that A is writing the same location that B is reading, and directly provides the data rather than requiring the write to occur before the read

EMBEDDED PROCESSORS

- **Addressing data hazards: Interlocks**
    - Interlocks therefore provide hardware that automatically inserts pipeline bubbles

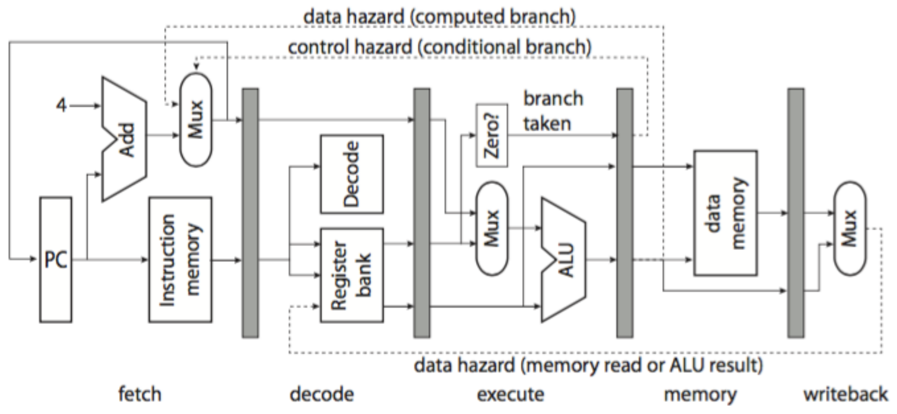


EMBEDDED PROCESSORS

- **Addressing data hazards:**
  - **Out-of-order execution**: hardware is provided that detects a hazard, but instead of simply 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 further reduces the number of pipeline bubbles

EMBEDDED PROCESSORS

- **Control hazards**
  - 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
  - If A is a conditional branch instruction, then A has to have reached the memory stage before the PC can be updated
  - The 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



E. A. Lee and S. A. Seshia, Introduction to Embedded Systems

EMBEDDED PROCESSORS

- **Addressing control hazards:**
  - **Delayed branch**: simply 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
  - **Interlocks**: hardware to insert pipeline bubbles as needed, just as with data hazards
  - **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

EMBEDDED PROCESSORS

Data and Control hazards:

- Except for explicit pipelines and delayed branches, **all of the techniques introduce variability** in the timing of execution of an instruction sequence
- **Analysis of the timing** of a program can **become extremely difficult when there is a deep pipeline** with elaborate forwarding and speculation
  - **Explicit pipelines** are relatively **common in DSP processors**, which are often applied **in contexts where precise timing is essential**
  - **Out-of-order and speculative execution** are common **in general-purpose processors**, where **timing matters only in an aggregate sense**

An embedded system designer needs to understand the requirements of the application and **avoid processors where the requisite level of timing precision is unachievable**