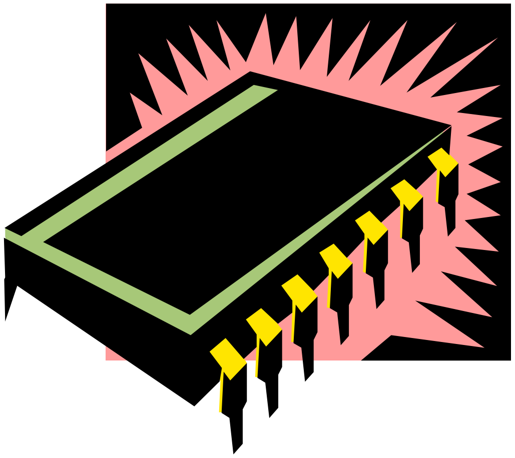
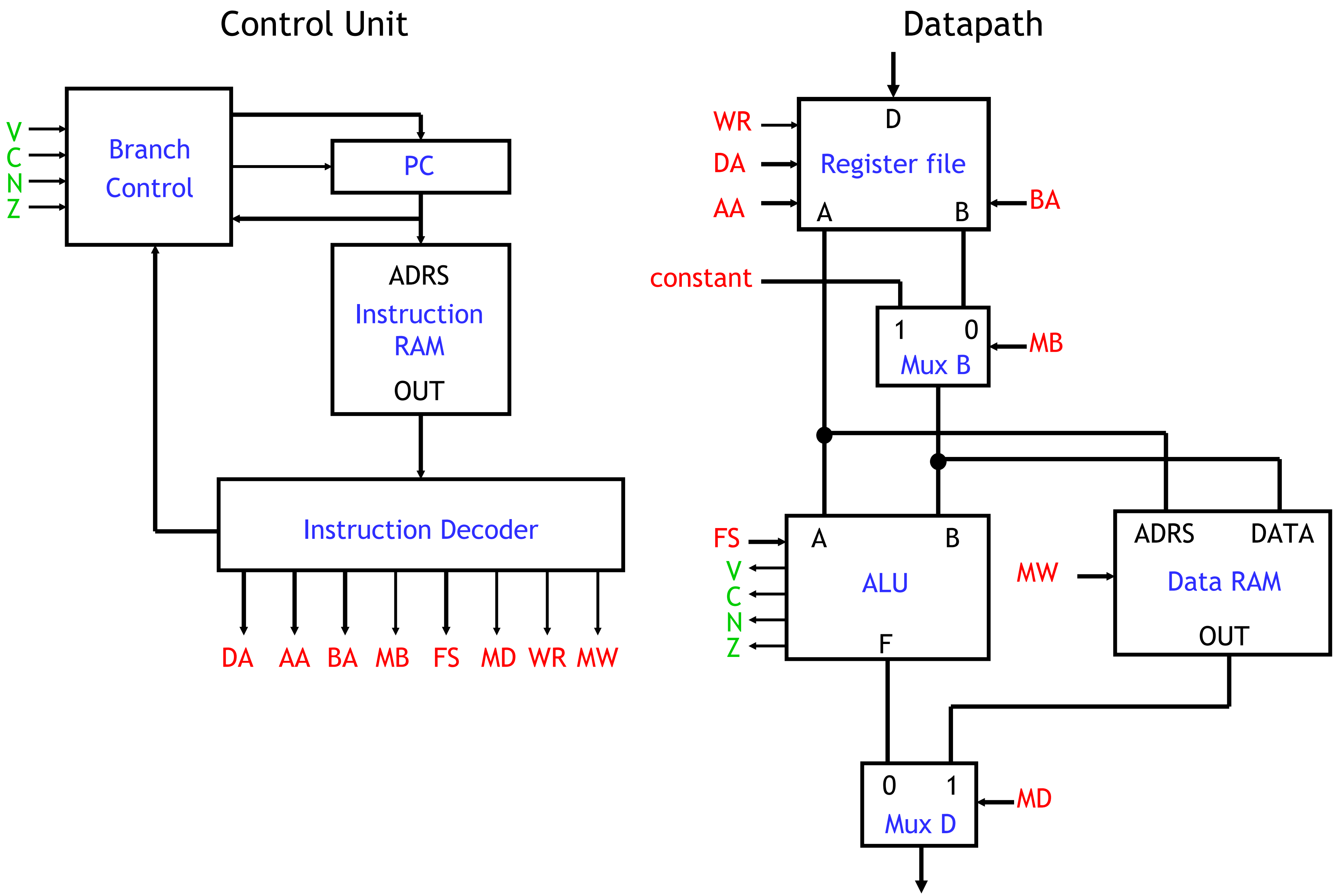
More advanced CPUs



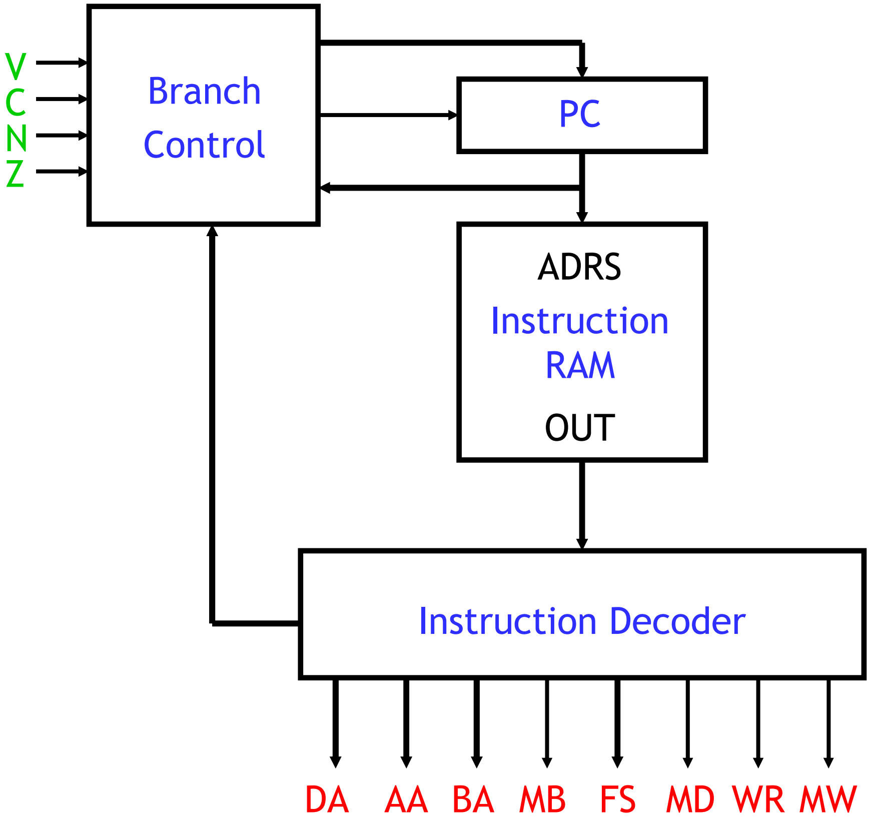
- In the last two weeks we presented the design of a basic processor.
  - The datapath performs operations on register and memory data.
  - A control unit translates program instructions into datapath signals.
  - The instruction set architecture closely reflects the processor design and serves as the interface between hardware and software.
- Today we'll introduce some advanced CPU designs, which are covered in more detail in CS232 and CS333.
  - Multicycle processors support more complex instructions.
  - Pipelined CPUs achieve higher clock rates and performance.

Our basic processor



A single-cycle processor

- This is a single-cycle processor, since every instruction executes in one clock cycle.
  1. An instruction is read from the instruction memory.
  2. The instruction decoder generates the correct datapath control signals.
  3. Source registers are read.
  4. An ALU or data memory operation is performed in the datapath.
  5. The PC is incremented, or reloaded for branches and jumps.



Limitations of single-cycle CPUs

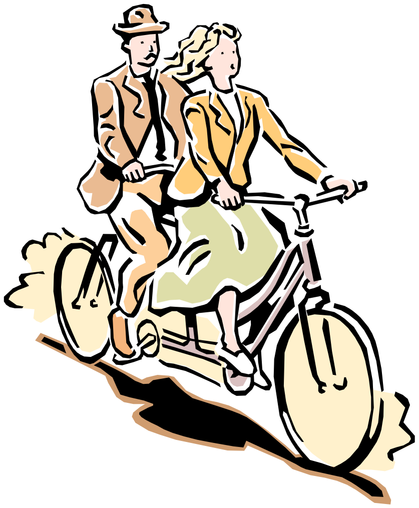
- That’s a lot of work to squeeze into one clock cycle!
- The clock cycle time, or the length of each cycle, has to be long enough to allow any single instruction to complete.
  - But the longer the cycle time, the lower the clock rate can be.
  - For example, a 10ns clock cycle time corresponds to a 100MHz CPU, while a 1GHz processor has a cycle time of just 1ns!
- Our basic CPU expects each instruction to execute in just one cycle, so we are limited to a relatively simple assembly language.
  - To support complex instructions, we would have to lengthen the cycle time, thus decreasing the clock rate.
  - This also means that any hardware which needs multiple clock cycles, such as serial adders or multipliers, cannot be easily used.

RISC vs. CISC

- Our sample processor uses a Reduced Instruction Set Computer or RISC architecture, which became popular in the mid-80s.
  - We support only relatively simple instructions, so some programs can be long and tedious to write in assembly.
  - On the other hand the hardware is fairly simple too. This makes it not only easier to understand, but also easier to optimize and make fast.
- Older CPUs use Complex Instruction Set Computer or CISC architectures.
  - Programs for CISC machines are usually shorter and easier to write, due to support for more complex instructions and addressing modes.
  - But the processor hardware is also much more complex, and harder to understand and optimize.
- In the 1980s there was a great debate between these two architectures, with much wailing and gnashing of teeth.
  - The ease of assembly programming has become much less relevant now, with modern compiler technology and high-level languages.
  - Today’s processors combine the best of both CISC and RISC models.

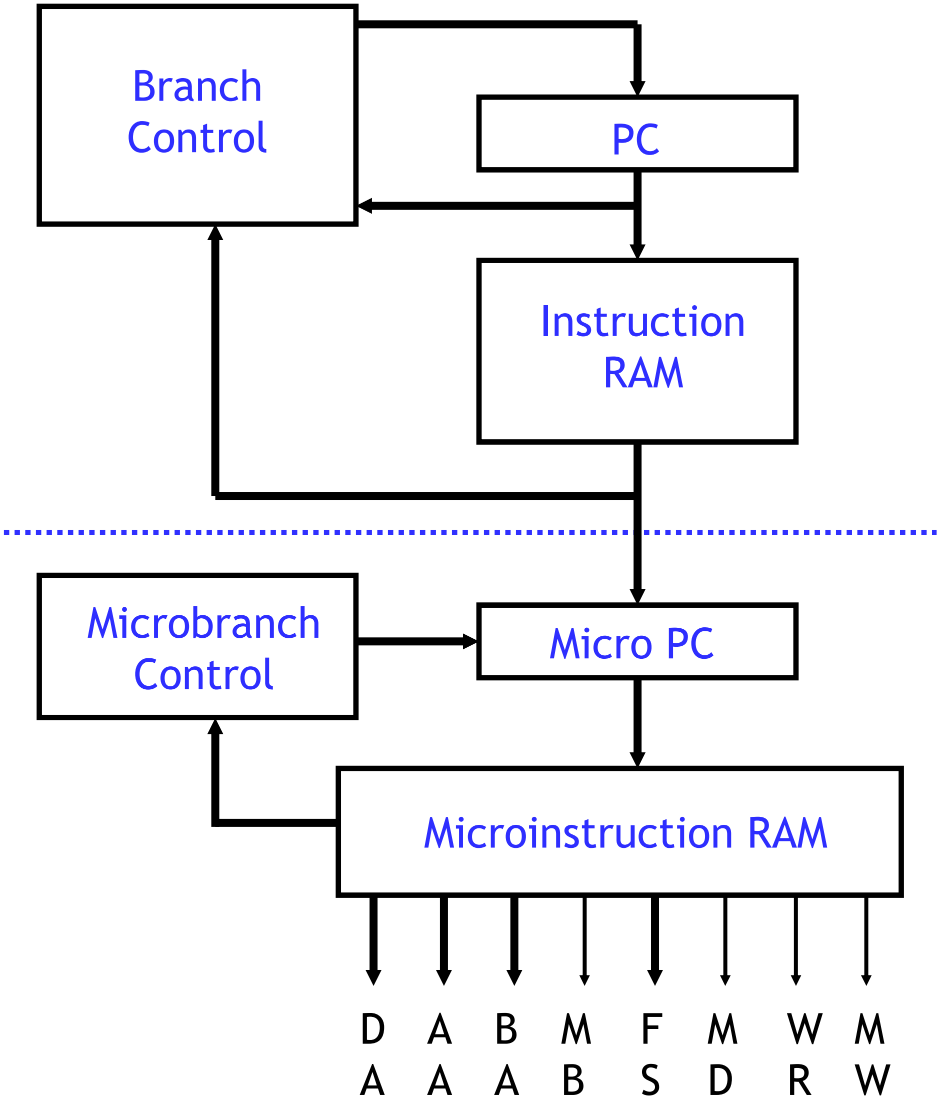
Multicycle processors

- A multicycle processor can support a more complex instruction set.
- Each complex instruction is implemented as a microprogram—a sequence of simpler, single-cycle operations like the ones we've seen already.
  - This is kind of like writing a function to implement the more complex instruction, except the function is stored in hardware.
  - By breaking longer instructions into sequences of shorter ones, we can keep cycle times low and clock rates high.



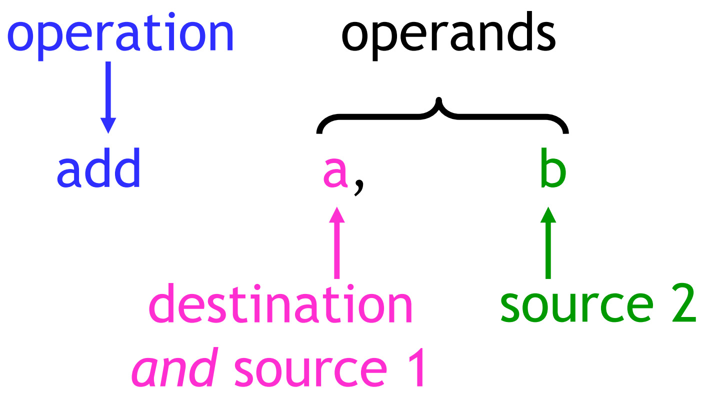
Multicycle control

- The top half of the diagram is our original control unit.
- Each complex instruction is sent to the microcontrol unit in the lower half of the picture.
  - The microinstruction RAM holds microprograms for each of the complex instructions.
  - Each microinstruction is then decoded to generate the right datapath control signals.
- The two parts of this diagram look similar because they serve similar purposes—the top half handles complex instructions, while the bottom handles microinstructions.



The Intel 8086 instruction set

- Intel’s 8086 instruction set, introduced in 1978, is one CISC architecture that is still extremely popular today.
- The 8086 is a two-address, register-to-memory architecture.



operation

operands

add a, b

destination *and* source 1

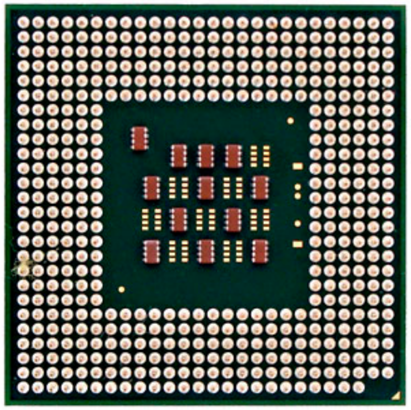
source 2

- This is interpreted as a = a + b.
  - a can be a register or a memory reference.
  - b can be a register, a memory reference, or a constant.
  - a and b cannot *both* be memory addresses.
- This example shows a single instruction that can both access memory *and* perform an ALU operation, unlike instructions for our sample processor.

Modern 8086 processors

- Modern processors from Intel such as the Pentium 4 are still based on the 8086 instruction set.
- They use multicycle implementations similar to what we just showed.
  - The simpler 8086 instructions can be executed very quickly.
  - More complex instructions are translated into sequences of simpler microinstructions for the processor's "RISC core."
  - Modern compilers try to avoid the slower, inefficient instructions in the ISA, which are provided only for compatibility.

3.2GHz Pentium 4 from
Digit-Life.com



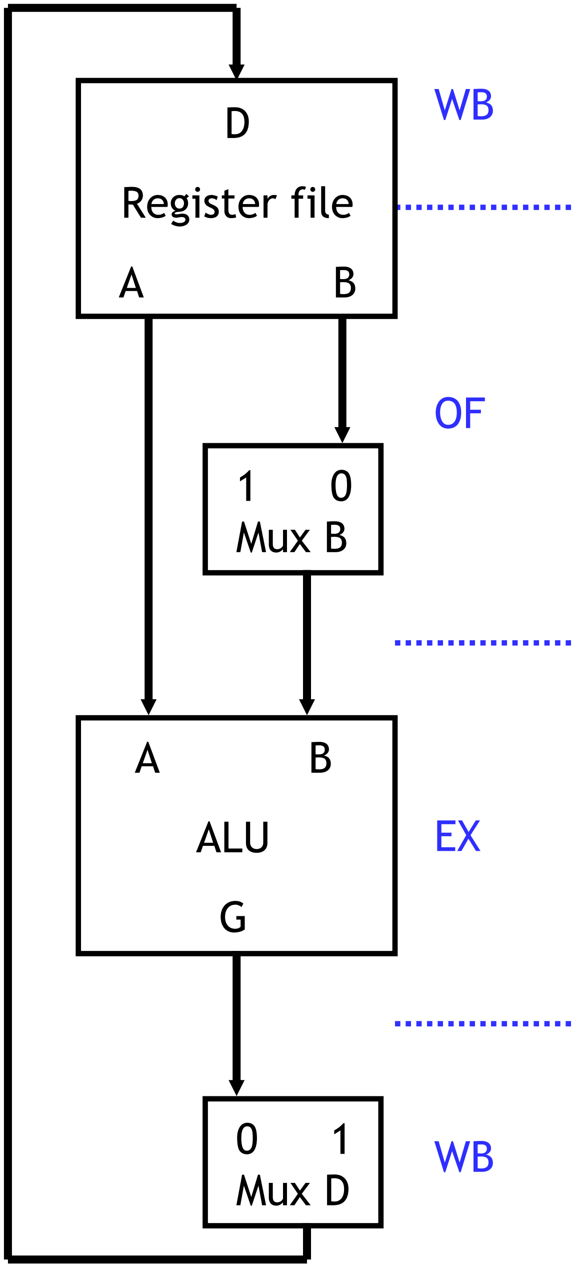
DEC VAX 11/780

- Another famous CISC architecture is the VAX, designed in 1978 by Digital Equipment Corporation.
- It boasts one of the most complex instruction sets ever.
- VMS, the VAX multiuser, cluster-based operating system, was designed by Dave Cutler, who was also in charge of Windows NT.
- The VAX had a 32-bit processor, seven years before Intel’s 80386.
- The cycle time was 200ns. 5MHz!
- All of this cost $200,000.

Vance Haemmerle’s VAX

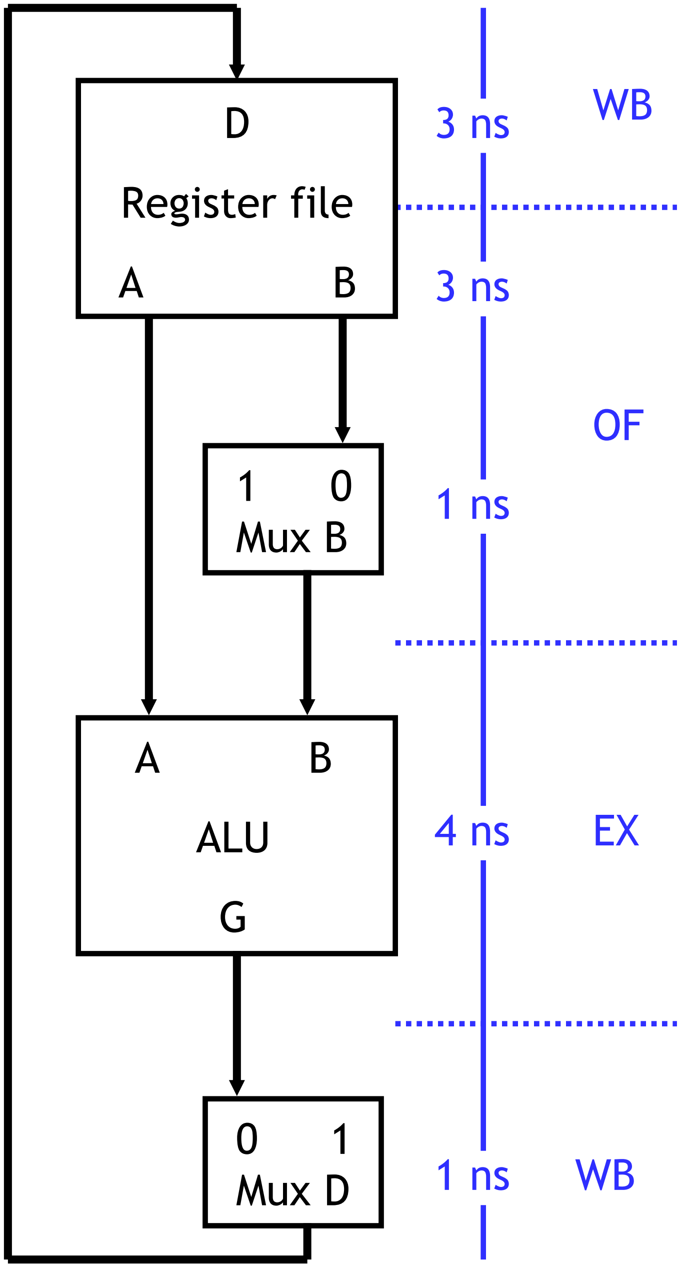
A simplified review of the datapath

- Let's go back to our original processor design.
- The datapath is one big register transfer system that goes through several "stages."
  - First, data is read from the register file. This is sometimes called the operand fetch stage or OF for short.
  - Next, the ALU performs an operation in the execute stage (EX).
  - Finally, in the writeback stage (WB), a result is saved back to the register file.
- Today we'll focus on ALU operations and ignore memory reads and writes.



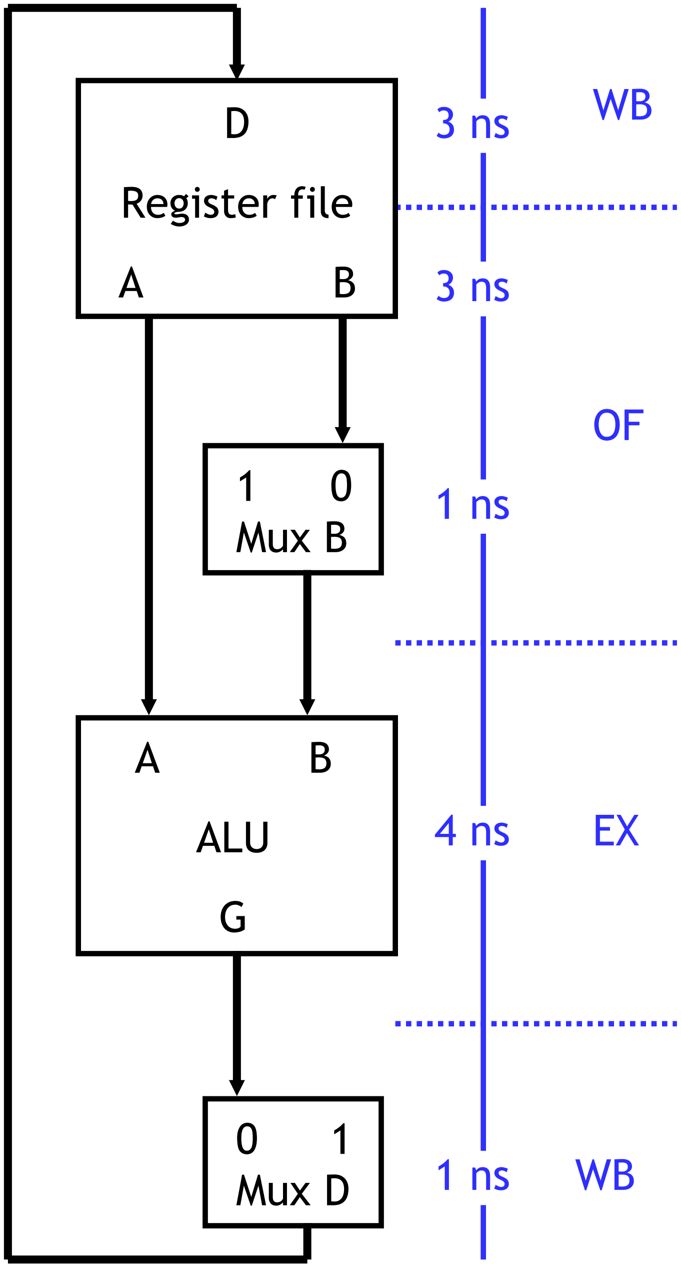
Ticking away the moments that make up a dog day

- How much time is required to execute one instruction, assuming the following delays for the datapath hardware elements?
  - Muxes have a 1ns delay.
  - The ALU needs 4ns to produce a result.
  - It takes 3ns to read from or write to the register file.
- Adding these delays up, it takes 12ns to run a single instruction.
  - This means the system's clock cycle time must be at least 12ns.
  - The clock rate would then be at most 1s/12ns = 83.3 MHz; this is 83.3 million instructions per second!



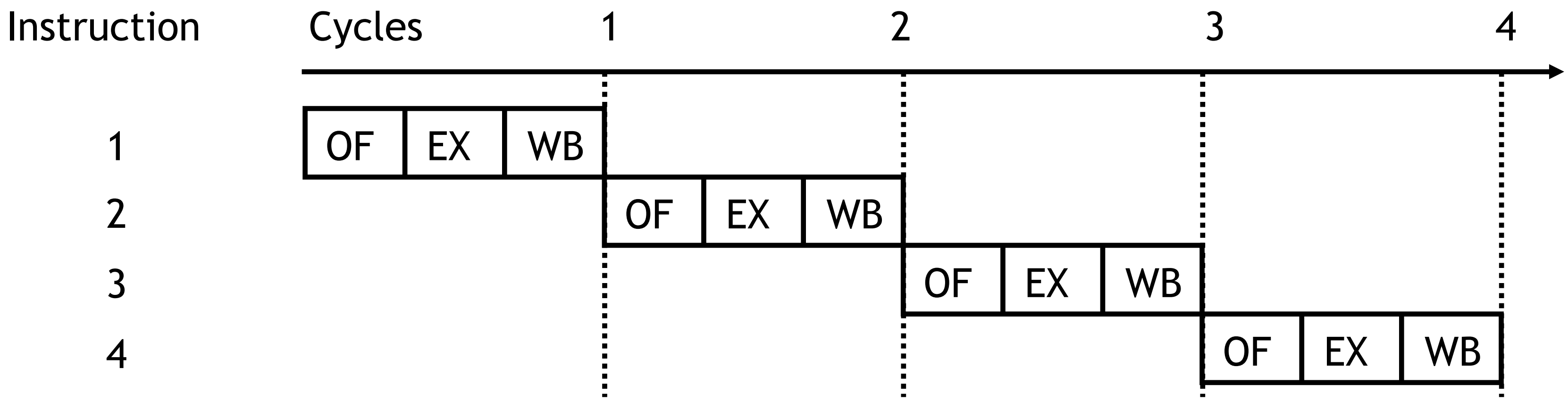
Can we go faster?

- Notice that during the OF and WB stages, the ALU is idle.
- Similarly, when the ALU is executing, the register file is just waiting around.
- If we can put all of these units to work together, then maybe we could execute more instructions per second.
- For example, the ALU could execute one instruction while the register file reads values for the *next* instruction, all during the *same* cycle.



Overlapping instruction executions

- In other words, we normally execute instructions one by one.



Instruction

Cycles 1 2 3 4

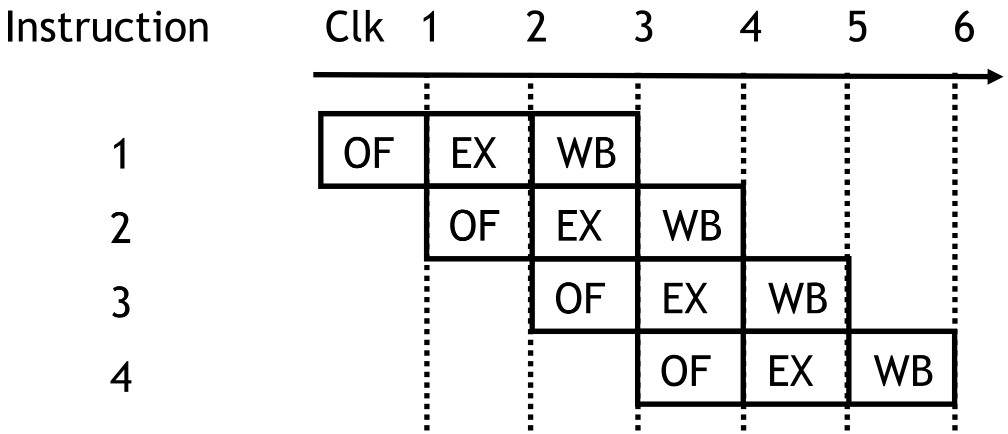
1 OF EX WB

2 OF EX WB

3 OF EX WB

4 OF EX WB

- Now let's overlap the execution of several instructions in the same cycle.



Instruction

Clk 1 2 3 4 5 6

1 OF EX WB

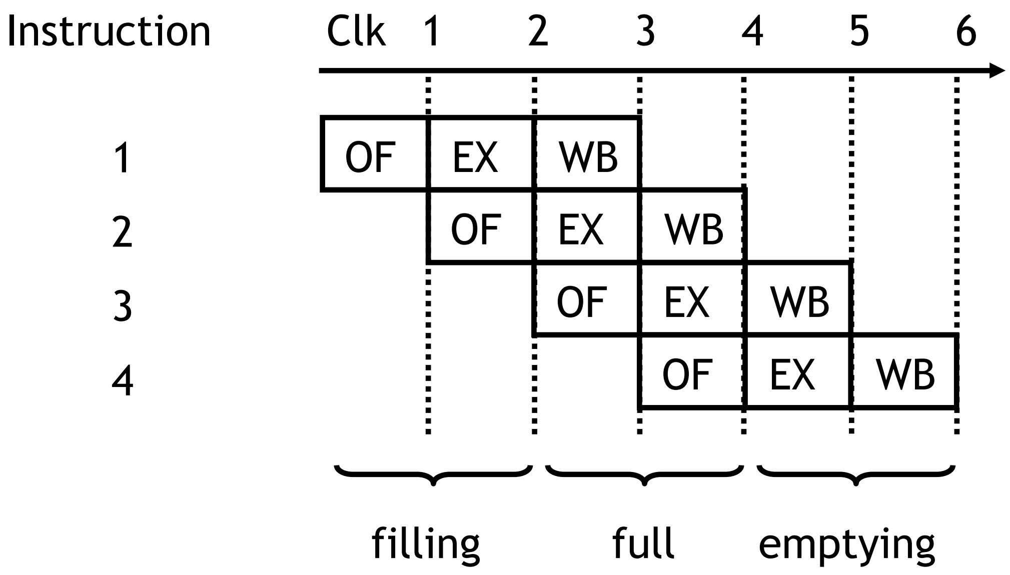
2 OF EX WB

3 OF EX WB

4 OF EX WB

Pipelining

- This technique is known as pipelining.
- There are two more important terms, referring to the diagram below.
  - During the third and fourth cycles, the pipeline is full—the ALU and register file are both busy, and the hardware is fully utilized.
  - In the first two clock cycles, the pipeline is filling, while in the last two cycles it is emptying.



Instruction	Clk 1	2	3	4	5	6
1	OF	EX	WB			
2		OF	EX	WB		
3			OF	EX	WB	
4				OF	EX	WB

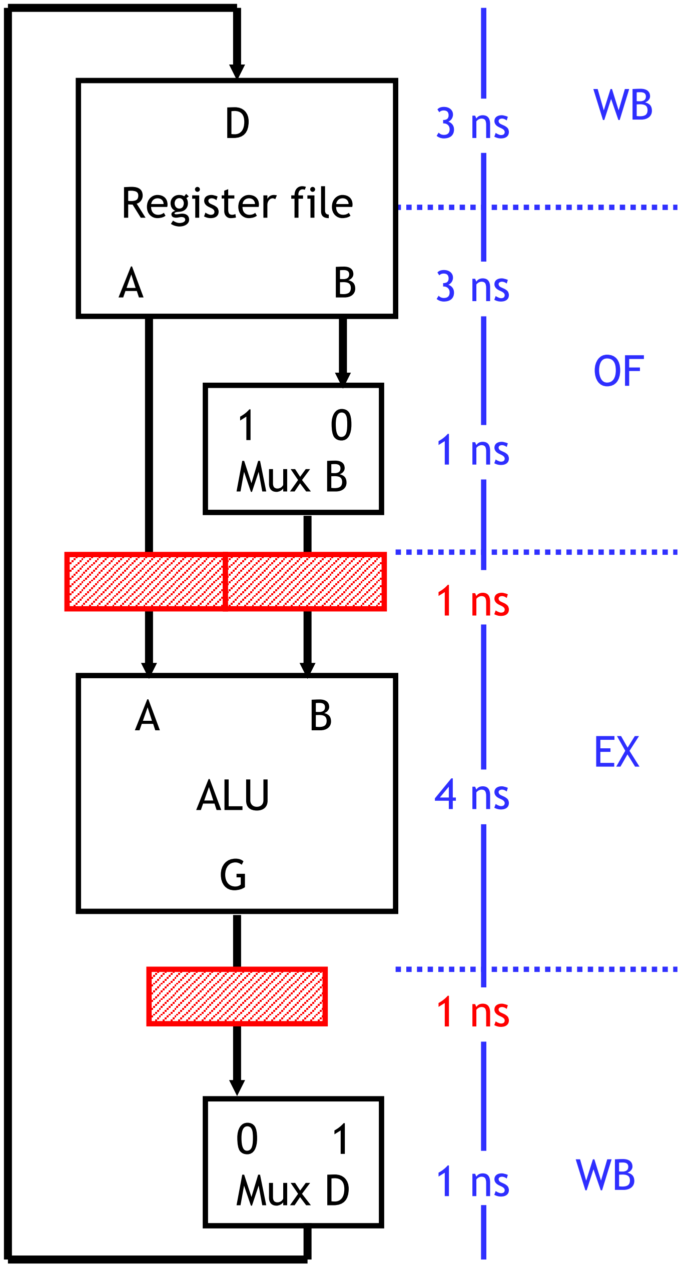
filling full emptying

A pipelined datapath

- For pipelining, we need to add registers between the stages.
- Why? The EX stage uses data that was fetched during the *previous* clock cycle, for the *previous* instruction.

Instruction	Clk	1	2	3	4	5	6
1	OF	EX	WB				
2		OF	EX	WB			
3			OF	EX	WB		
4				OF	EX	WB	

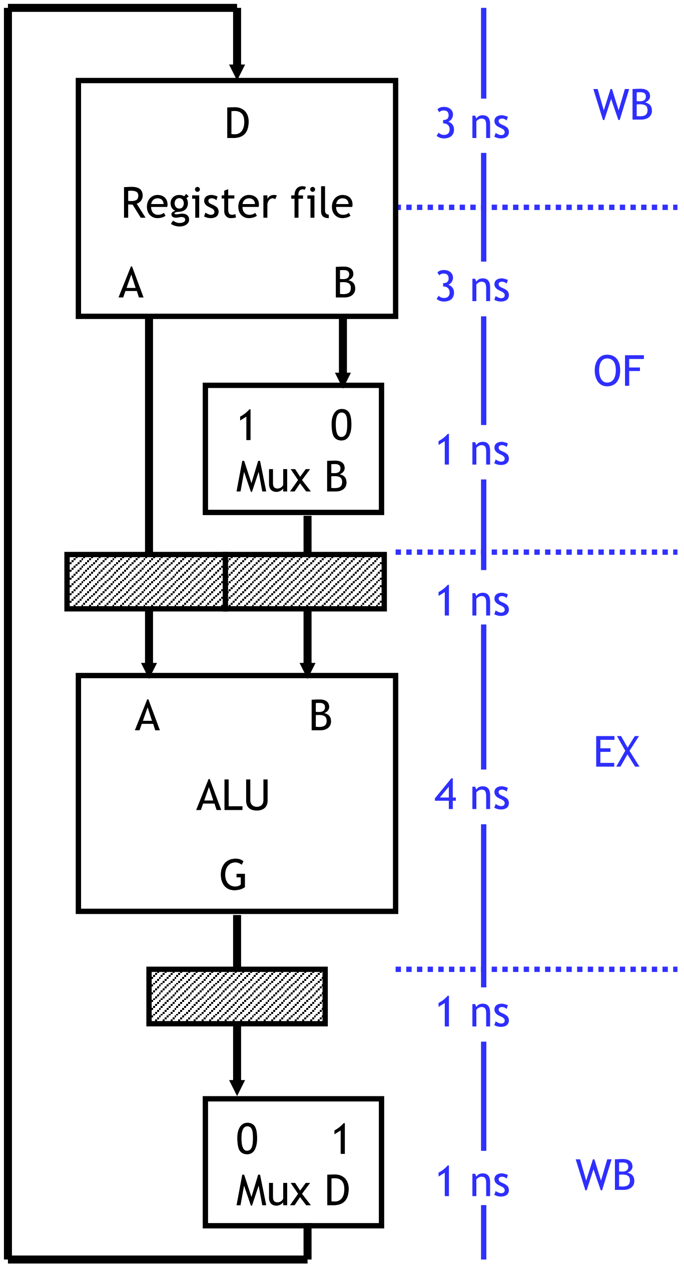
- Similarly, we need to write back the ALU result from the previous cycle.
- Registers will save values from previous cycles. We’ll say they have 1ns delays.



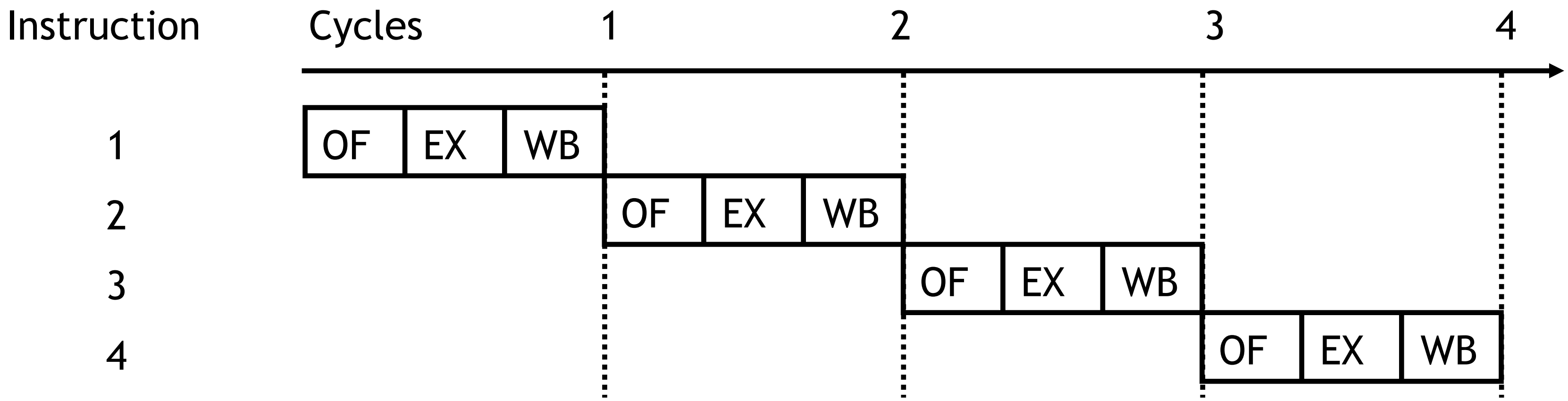
Delays with pipelining

- The total time required to execute one instruction is now 14ns.
  - Operand fetch takes 4ns.
  - Execution requires 5ns.
  - Writeback also needs 5ns.
- This is *slower* than the original 12ns!
- But now the cycle time is just 5ns (the delay of the longest stage), so the clock rate can be 1s/5ns, or 200 MHz.

Instruction	Clk	1	2	3	4	5	6
1	OF	EX	WB				
2		OF	EX	WB			
3			OF	EX	WB		
4				OF	EX	WB	

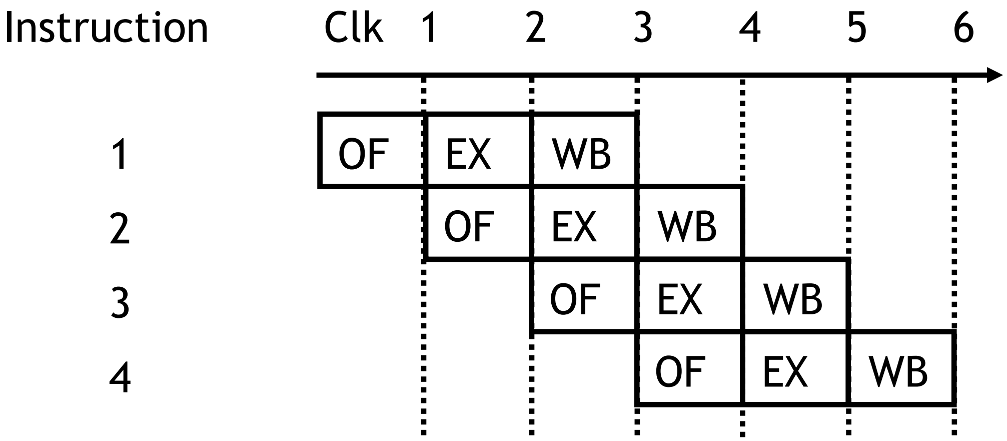


Execution time without pipelining



- How long did it take to execute four instructions originally?
  - Before, each instruction needed one cycle for execution.
  - The cycle time was 12ns, so the total time was 4 × 12 = 48ns.
- Another measure of speed is throughput, or the number of instructions completed per unit of time. Here, one instruction finishes every 12ns.

Execution time with pipelining



Instruction	Clk 1	2	3	4	5	6
1	OF	EX	WB			
2		OF	EX	WB		
3			OF	EX	WB	
4				OF	EX	WB

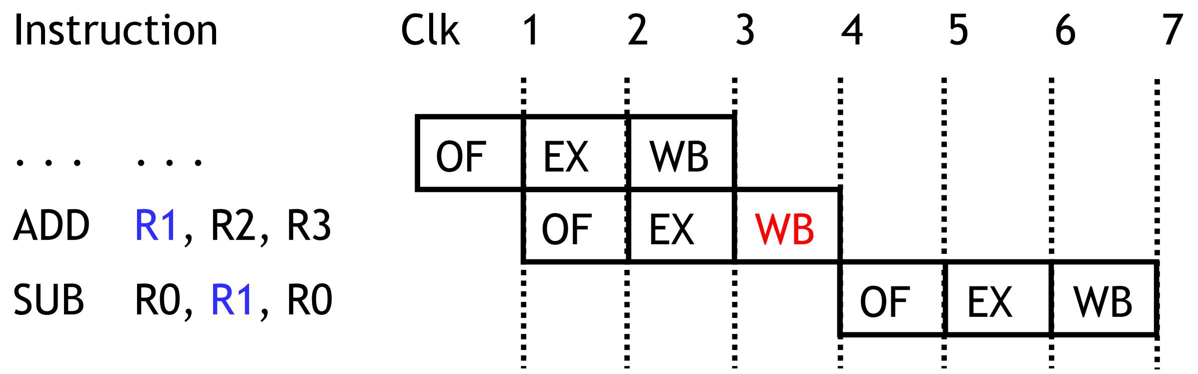
- How about execution time *with* pipelining?
  – It takes six clock cycles to execute four instructions, but with a clock period of just 5ns, the total time is only 6 × 5 = 30ns!
  – The throughput is also higher. From cycles 3-6, you can see that one instruction completes every cycle, or every 5ns.

The miracle of pipelining

- At first it seems like things are slower with pipelining.
  - Each instruction takes a longer time to execute (14ns vs. 12ns).
  - A sequence of instructions also requires more clock cycles (6 vs. 4).
- But programs actually run faster!
  - We can overlap the execution of several instructions in one cycle.
  - The clock period is also much shorter (12ns vs. 5ns).
  - As a result, the total execution time is less overall (30ns vs. 48ns).
- The speedup is even greater for longer sequences of instructions.
  - Our example has just four instructions, and most of the clock cycles are spent filling or emptying the pipeline.
  - The ideal case is when the pipeline is full, and one instruction can be completed on every clock cycle. Then, at 200 MHz, we could execute up to 200 million instructions per second.

The hazards of pipelining

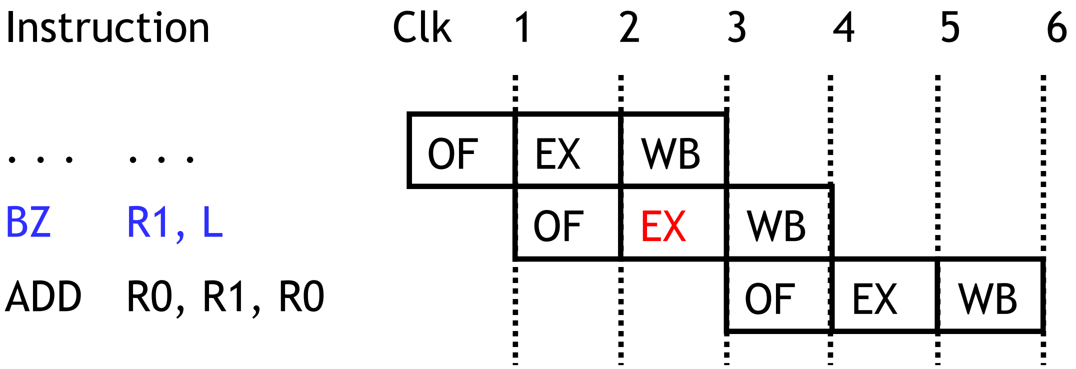
- Unfortunately, things are not so perfect in real life. There are many kinds of hazards which make the pipeline run at less than full efficiency.
- A data hazard arises when one instruction needs to wait for the output of another one, so their executions can't be overlapped.



- Here the SUB instruction uses R1, which is changed by the ADD.
  - SUB can't read R1 until *after* ADD modifies it in cycle 3. Otherwise, the SUB would be using the old, incorrect value of R1.
  - One solution is to stall, or delay, the SUB instruction for two cycles.

Branch hazards

- A branch hazard arises when branch instructions are encountered. We don't know which instruction should be executed next, until the ALU's Z output is generated in the branch's EX stage.
- Again, we could stall the pipeline and wait for the Z output.

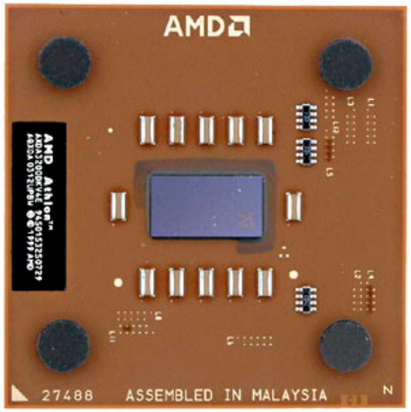


Instruction		Clk	1	2	3	4	5	6
. . .	. . .	OF	EX	WB				
BZ	R1, L		OF	EX	WB			
ADD	R0, R1, R0				OF	EX	WB	

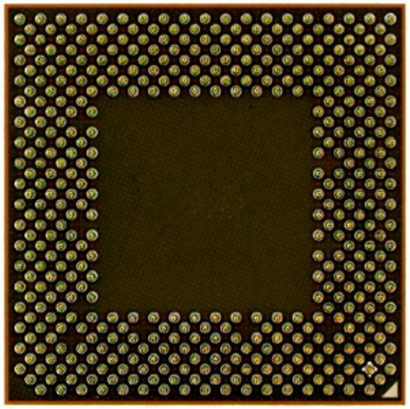
- Another solution is branch prediction. We might guess the branch will not be taken, and just execute the next instruction without stalling.
  - If we guess correctly, then we avoid losing time due to a stall.
  - But this will obviously cause difficulties if we guessed incorrectly and the branch *is* taken.

Pipeline depths and clock rates

- It’s hard to determine the ideal number of stages for a pipelined CPU.
  - Simpler stages can execute faster, resulting in lower cycle times and higher clock rates.
  - But simpler stages also mean more stages and cycles are needed to execute one instruction, and hazards have a much bigger impact on performance.
- Real processors have longer pipelines than our simple 3-stage example.
  - The Intel Pentium III, AMD Athlon XP and Motorola PowerPC G4 have pipelines with 7-12 stages, and clock rates from 1.4-2.2 GHz.
  - Newer CPUs, like the Pentium 4 and IBM PowerPC 970, rely on much longer pipelines with 16-20 stages to reach clock rates up to 3.2 GHz.



Athlon XP 3200+ from Digit-Life.com



Measuring performance

- You can't judge CPU performance by the clock rate alone, since different processors might do different amounts of work in one cycle.
  - The Athlon XP is often considered to be more efficient, because tests show it to be faster than a Pentium 4 at the same clock rate.
  - On the other hand, the Pentium 4 can reach much higher clock rates, which currently makes up for its inefficiencies.
- System performance also depends greatly on the software itself.
  - Programs that use slower instructions or exhibit more hazards will of course be slower than more well-written programs.
  - Modern processors are very complex, so good compilers are critical to achieving maximum performance.

Summary

- Multicycle processors implement complex instructions by translating them into sequences of simpler, one-cycle micro-instructions.
- With pipelining, the execution of several instructions is overlapped.
  - Individual instructions take longer to run, but we can execute several instructions simultaneously.
  - The clock cycle time can also be reduced, which can help to decrease the overall execution time of programs.
  - But there are many kinds of hazards that must be taken care of, since not all instructions can be executed at the same time.
- With so many different instruction sets and implementation techniques, it's difficult to accurately estimate the speeds of different processors.