addressing modes that require multiple memory accesses substantially complicate pipeline control and make it difficult to keep the pipeline flowing smoothly.

Perhaps the best example is the DEC Alpha and the DEC NVAX. In comparable technology, the newer instruction set architecture of the Alpha allowed an implementation whose performance is more than twice as fast as NVAX. In another example, Bhandarkar and Clark [1991] compared the MIPS M/2000 and the DEC VAX 8700 by counting clock cycles of the SPEC benchmarks; they concluded that although the MIPS M/2000 executes more instructions, the VAX on average executes 2.7 times as many clock cycles, so the MIPS is faster.

isdom
vise

4.14 Concluding Remarks

As we have seen in this chapter, both the datapath and control for a processor can be designed starting with the instruction set architecture and an understanding of the basic characteristics of the technology. In Section 4.3, we saw how the datapath for a MIPS processor could be constructed based on the architecture and the decision to build a single-cycle implementation. Of course, the underlying technology also affects many design decisions by dictating what components can be used in the datapath, as well as whether a single-cycle implementation even makes sense.

The ime

Pipelining improves throughput but not the inherent execution time, or **instruction latency**, of instructions; for some instructions, the latency is similar in length to the single-cycle approach. Multiple instruction issue adds additional datapath hardware to allow multiple instructions to begin every clock cycle, but at an increase in effective latency. Pipelining was presented as reducing the clock cycle time of the simple single-cycle datapath. Multiple instruction issue, in comparison, clearly focuses on reducing clock cycles per instruction (CPI).

Pipelining and multiple issue both attempt to exploit instruction-level parallelism. The presence of data and control dependences, which can become hazards, are the primary limitations on how much parallelism can be exploited. Scheduling and speculation, both in hardware and in software, are the primary techniques used to reduce the performance impact of dependences.

The switch to longer pipelines, multiple instruction issue, and dynamic scheduling in the mid-1990s has helped sustain the 60% per year processor performance increase that started in the early 1980s. As mentioned in Chapter 1, these microprocessors preserved the sequential programming model, but they eventually ran into the power wall. Thus, the industry has been forced to try multiprocessors, which exploit parallelism at much coarser levels (the subject of Chapter 7). This trend has also caused designers to reassess the power-performance implications

of some of the inventions since the mid-1990s, resulting in a simplification of pipelines in the more recent versions of microarchitectures.

To sustain the advances in processing performance via parallel processors, Amdahl's law suggests that another part of the system will become the bottleneck. That bottleneck is the topic of the next chapter: the memory system.

4.15 Historical Perspective and Further Reading

This section, which appears on the CD, discusses the history of the first pipelined processors, the earliest superscalars, and the development of out-of-order and speculative techniques, as well as important developments in the accompanying compiler technology.

4.16 Exercises

Contributed by Milos Prvulovic of Georgia Tech

Exercise 4.1

Different instructions utilize different hardware blocks in the basic single-cycle implementation. The next three problems in this exercise refer to the following instruction:

	Instruction	Interpretation
a.	`add Rd,Rs,Rt`	`Reg[Rd]=Reg[Rs]+Reg[Rt]`
b.	`lw Rt,Offs(Rs)`	`Reg[Rt]=Mem[Reg[Rs]+Offs]`

4.1.1 [5] <4.1> What are the values of control signals generated by the control in Figure 4.2 for this instruction?

4.1.2 [5] <4.1> Which resources (blocks) perform a useful function for this instruction?

4.1.3 [10] <4.1> Which resources (blocks) produce outputs, but their outputs are not used for this instruction? Which resources produce no outputs for this instruction?

Different execution units and blocks of digital logic have different latencies (time needed to do their work). In Figure 4.2 there are seven kinds of major blocks. Latencies of blocks along the critical (longest-latency) path for an instruction determine the minimum latency of that instruction. For the remaining three problems in this exercise, assume the following resource latencies:

	I-Mem	Add	Mux	ALU	Regs	D-Mem	Control
a.	400ps	100ps	30ps	120ps	200ps	350ps	100ps
b.	500ps	150ps	100ps	180ps	220ps	1000ps	65ps

4.1.4 [5] <4.1> What is the critical path for a MIPS AND instruction?

4.1.5 [5] <4.1> What is the critical path for a MIPS load (LD) instruction?

4.1.6 [10] <4.1> What is the critical path for a MIPS BEQ instruction?

Exercise 4.2

The basic single-cycle MIPS implementation in Figure 4.2 can only implement some instructions. New instructions can be added to an existing ISA, but the decision whether or not to do that depends, among other things, on the cost and complexity such an addition introduces into the processor datapath and control. The first three problems in this exercise refer to this new instruction:

	Instruction	Interpretation
a.	`add3 Rd,Rs,Rt,Rx`	`Reg[Rd]=Reg[Rs]+Reg[Rt]+Reg[Rx]`
b.	`sll Rt,Rd,Shift`	`Reg[Rd]= Reg[Rt] << Shift` (shift left by Shift bits)

4.2.1 [10] <4.1> Which existing blocks (if any) can be used for this instruction?

4.2.2 [10] <4.1> Which new functional blocks (if any) do we need for this instruction?

4.2.3 [10] <4.1> What new signals do we need (if any) from the control unit to support this instruction?

When processor designers consider a possible improvement to the processor datapath, the decision usually depends on the cost/performance tradeoff. In the following three problems, assume that we are starting with a datapath from Figure 4.2, where I-Mem, Add, Mux, ALU, Regs, D-Mem, and Control blocks have latencies of 400ps, 100ps, 30ps, 120ps, 200ps, 350ps, and 100ps, respectively, and costs of 1000, 30, 10, 100, 200, 2000, and 500, respectively. The remaining three problems in this exercise refer to the following processor improvement:

	Improvement	Latency	Cost	Benefit
a.	Faster Add	–20ps for Add units	+20 per Add unit	Replaces existing Add units with faster ones.
b.	Larger Registers	+100ps for Regs	+200 for Regs	Fewer loads and stores needed to save and restore register values. This results in 5% fewer instructions.

4.2.4 [10] <4.1> What is the clock cycle time with and without this improvement?

4.2.5 [10] <4.1> What is the speed-up achieved by adding this improvement?

4.2.6 [10] <4.1> Compare the cost/performance ratio with and without this improvement.

Exercise 4.3

Problems in this exercise refer to the following logic block:

	Logic Block
a.	Small I-Memory with four 8-bit words
b.	Small Registers unit with two 8-bit registers

4.3.1 [5] <4.1, 4.2> Does this block contain logic only, flip-flops only, or both?

4.3.2 [20] <4.1, 4.2> Show how this block can be implemented. Use only AND, OR, NOT, and D-elements.

4.3.3 [10] <4.1, 4.2> Repeat Exercise 4.3.2, but the AND and OR gates you use must all be 2-input gates.

Cost and latency of digital logic depends on the kinds of basic logic elements (gates) that are available and on the properties of these gates. The remaining three problems in this exercise refer to these gates, latencies, and costs:

	NOT		2-input AND or OR		Each additional input for AND/OR		D-element	
	Latency	Cost	Latency	Cost	Latency	Cost	Latency	Cost
a.	20ps	1	30ps	2	+0ps	+1	40ps	6
b.	50ps	1	100ps	2	+40ps	+1	160ps	2

4.3.4 [5] <4.1, 4.2> What is the latency of your implementation from Exercise 4.3.2?

4.3.5 [5] <4.1, 4.2> What is the cost of your implementation from Exercise 4.3.2?

4.3.6 [20] <4.1, 4.2> Change your design to minimize the latency, then to minimize the cost. Compare the cost and latency of these two optimized designs.

Exercise 4.4

When implementing a logic expression in digital logic, one must use the available logic gates to implement an operator for which a gate is not available. Problems in this exercise refer to the following logic expressions:

	Control signal 1	Control signal 2
a.	(((A OR B) OR C) OR (A AND C)) OR (A AND B)	(A OR B) OR C
b.	(((A OR B) XOR B) OR (A OR C)) OR (A AND B)	A AND B

4.4.1 [5] <4.2> Implement the logic for the Control signal 1. Your circuit should directly implement the given expression (do not reorganize the expression to "optimize" it), using NOT gates and 2-input AND, OR, and XOR gates.

4.4.2 [10] Assuming that all gates have equal latencies, what is the length (in gates) of the critical path in your circuit from Exercise 4.4.1?

4.4.3 [10] <4.2> When multiple logic expressions are implemented, it is possible to reduce implementation cost by using the some signals in more than one expression. Repeat Exercise 4.4.1, but implement both Control signal 1 and Control signal 2, and try to "share" circuitry between expressions whenever possible.

For the remaining three problems in this exercise, we assume that the following basic digital logic elements are available, and that their latency and cost are as follows:

	NOT		2-input AND		2-input OR		2-input XOR	
	Latency	Cost	Latency	Cost	Latency	Cost	Latency	Cost
a.	20ps	1	30ps	2	34ps	3	40ps	6
b.	50ps	1	100ps	2	120ps	2	150ps	2

4.4.4 [10] <4.2> What is the length of the critical path in your circuit from 4.4.3?

4.4.5 [10] <4.2> What is the cost of your circuit from Exercise 4.4.3?

4.4.6 [10] <4.2> What fraction of the cost was saved in your circuit from Exercise 4.4.3 by implementing these two control signals together instead of separately?

Exercise 4.5

The goal of this exercise is to help you familiarize yourself with the design and operation of sequential logical circuits. Problems in this exercise refer to this ALU operation:

	ALU operation
a.	Add-one (X+1)
b.	Shift left by 2 bits (X<<2)

4.5.1 [20] <4.2> Design a circuit with 1-bit data inputs and a 1-bit data output that accomplishes this operation serially, starting with the least-significant bit. In a serial implementation, the circuit is processing input operands bit by bit, generating output bits one by one. For example, a serial AND circuit is simply an AND gate; in cycle N we give it the Nth bit from each of the operand and we get the Nth bit of the result. In addition to data inputs, the circuit has a Clk (clock) input and a "Start" input that is set to 1 only in the very first cycle of the operation. In your design, you can use D-elements and NOT, AND, OR, and XOR gates.

4.5.2 [20] <4.2> Repeat Exercise 4.5.1, but now design a circuit that accomplishes this operation 2 bits at a time.

In the rest of this exercise, we assume that the following basic digital logic elements are available, and that their latency and cost are as follows:

	NOT		AND		OR		XOR		D-element	
	Latency	Cost	Latency	Cost	Latency	Cost	Latency	Cost	Latency	Cost
a.	20ps	1	30ps	2	20ps	2	30ps	4	40ps	6
b.	40ps	1	50ps	2	60ps	2	80ps	3	80ps	12

The time given for a D-element is its setup time. The data input of a flip-flop must have the correct value one setup-time before the clock edge (end of clock cycle) that stores that value into the flip-flop.

4.5.3 [10] <4.2> What is the cycle time for the circuit you designed in Exercise 4.5.1? How long does it take to perform the 32-bit operation?

4.5.4 [10] <4.2> What is the cycle time for the circuit you designed in Exercise 4.5.2? What is the speed-up achieved by using this circuit instead of the one from Exercise 4.5.1 for a 32-bit operation?

4.5.5 [10] <4.2> Compute the cost for the circuit you designed in Exercise 4.5.1, and then for the circuit you designed in Exercise 4.5.2.

4.5.6 [5] <4.2> Compare cost/performance ratios for the two circuits you designed in Exercises 4.5.1 and 4.5.2. For this problem, performance of a circuit is the inverse of the time needed to perform a 32-bit operation.

Exercise 4.6

Problems in this exercise assume that logic blocks needed to implement a processor's datapath have the following latencies:

	I-Mem	Add	Mux	ALU	Regs	D-Mem	Sign-extend	Shift-left-2
a.	400ps	100ps	30ps	120ps	200ps	350ps	20ps	2ps
b.	500ps	150ps	100ps	180ps	220ps	1000ps	90ps	20ps

4.6.1 [10] <4.3> If the only thing we need to do in a processor is fetch consecutive instructions (Figure 4.6), what would the cycle time be?

4.6.2 [10] <4.3> Consider a datapath similar to the one in Figure 4.11, but for a processor that only has one type of instruction: unconditional PC-relative branch. What would the cycle time be for this datapath?

4.6.3 [10] <4.3> Repeat Exercise 4.6.2, but this time we need to support only *conditional* PC-relative branches.

The remaining three problems in this exercise refer to the following logic block (resource) in the datapath:

	Resource
a.	Add 4 (to the PC)
b.	Data Memory

4.6.4 [10] <4.3> Which kinds of instructions require this resource?

4.6.5 [20] <4.3> For which kinds of instructions (if any) is this resource on the critical path?

4.6.6 [10] <4.3> Assuming that we only support `beq` and `add` instructions, discuss how changes in the given latency of this resource affect the cycle time of the processor. Assume that the latencies of other resources do not change.

Exercise 4.7

In this exercise we examine how latencies of individual components of the datapath affect the clock cycle time of the entire datapath, and how these components are utilized by instructions. For problems in this exercise, assume the following latencies for logic blocks in the datapath:

	I-Mem	Add	Mux	ALU	Regs	D-Mem	Sign-extend	Shift-left-2
a.	400ps	100ps	30ps	120ps	200ps	350ps	20ps	0ps
b.	500ps	150ps	100ps	180ps	220ps	1000ps	90ps	20ps

4.7.1 [10] <4.3> What is the clock cycle time if the only type of instructions we need to support are ALU instructions (`add`, `and`, etc.)?

4.7.2 [10] <4.3> What is the clock cycle time if we only had to support `lw` instructions?

4.7.3 [20] <4.3> What is the clock cycle time if we must support `add`, `beq`, `lw`, and `sw` instructions?

For the remaining problems in this exercise, assume that there are no pipeline stalls and that the breakdown of executed instructions is as follows:

	add	addi	not	beq	lw	sw
a.	30%	15%	5%	20%	20%	10%
b.	25%	5%	5%	15%	35%	15%

4.7.4 [10] <4.3> In what fraction of all cycles is the data memory used?

4.7.5 [10] <4.3> In what fraction of all cycles is the input of the sign-extend circuit needed? What is this circuit doing in cycles in which its input is not needed?

4.7.6 [10] <4.3> If we can improve the latency of one of the given datapath components by 10%, which component should it be? What is the speed-up from this improvement?

Exercise 4.8

When silicon chips are fabricated, defects in materials (e.g., silicon) and manufacturing errors can result in defective circuits. A very common defect is for one wire to affect the signal in another. This is called a cross-talk fault. A special

class of cross-talk faults is when a signal is connected to a wire that has a constant logical value (e.g., a power supply wire). In this case we have a stuck-at-0 or a stuck-at-1 fault, and the affected signal always has a logical value of 0 or 1, respectively.

The following problems refer to the following signal from Figure 4.24:

	Signal
a.	Instruction Memory, output Instruction, bit 7
b.	Control unit, output MemtoReg

4.8.1 [10] <4.3, 4.4> Let us assume that processor testing is done by filling the PC, registers, and data and instruction memories with some values (you can choose which values), letting a single instruction execute, then reading the PC, memories, and registers. These values are then examined to determine if a particular fault is present. Can you design a test (values for PC, memories, and registers) that would determine if there is a stuck-at-0 fault on this signal?

4.8.2 [10] <4.3, 4.4> Repeat Exercise 4.8.1 for a stuck-at-1 fault. Can you use a single test for both stuck-at-0 and stuck-at-1? If yes, explain how; if no, explain why not.

4.8.3 [60] <4.3, 4.4> If we know that the processor has a stuck-at-1 fault on this signal, is the processor still usable? To be usable, we must be able to convert any program that executes on a normal MIPS processor into a program that works on this processor. You can assume that there is enough free instruction memory and data memory to let you make the program longer and store additional data. Hint: the processor is usable if every instruction "broken" by this fault can be replaced with a sequence of "working" instructions that achieve the same effect.

The following problems refer to the following fault:

	Fault
a.	Stuck-at-1
b.	Becomes 0 if Instruction [31-26] has all bits at 0, no fault otherwise

4.8.4 [10] <4.3, 4.4> Repeat Exercise 4.8.1, but now the fault to test for is whether the "MemRead" control signal has this fault.

4.8.5 [10] <4.3, 4.4> Repeat Exercise 4.8.1, but now the fault to test for is whether the "Jump" control signal has this fault.

4.8.6 [40] <4.3, 4.4> Using a single test described Exercise 4.8.1, we can test for faults in several different signals, but typically not all of them. Describe a series of tests to look for this fault in all Mux outputs (every output bit from each of the five Muxes)? Try to do this with as few single-instruction tests as possible.

Exercise 4.9

In this exercise we examine the operation of the single-cycle datapath for a particular instruction. Problems in this exercise refer to the following MIPS instruction:

	Instruction
a.	`lw $1,40($6)`
b.	`Label: bne $1,$2,Label`

4.9.1 [10] <4.4> What is the value of the instruction word?

4.9.2 [10] <4.4> What is the register number supplied to the register file's "Read register 1" input? Is this register actually read? How about "Read register 2"?

4.9.3 [10] <4.4> What is the register number supplied to the register file's "Write register" input? Is this register actually written?

Different instructions require different control signals to be asserted in the datapath. The remaining problems in this exercise refer to the following two control signals from Figure 4.24:

	Control signal 1	Control signal 2
a.	RegDst	MemRead
b.	RegWrite	MemRead

4.9.4 [20] <4.4> What is the value of these two signals for this instruction?

4.9.5 [20] <4.4> For the datapath from Figure 4.24, draw the logic diagram for the part of the control unit that implements just the first signal. Assume that we only need to support `lw`, `sw`, `beq`, `add`, and `j` (jump) instructions.

4.9.6 [20] <4.4> Repeat Exercise 4.9.5, but now implement both of these signals.

Exercise 4.10

In this exercise we examine how the clock cycle time of the processor affects the design of the control unit, and vice versa. Problems in this exercise assume that the logic blocks used to implement the datapath have the following latencies:

	I-Mem	Add	Mux	ALU	Regs	D-Mem	Sign-extend	Shift-left-2	ALU Ctrl
a.	400ps	100ps	30ps	120ps	200ps	350ps	20ps	0ps	50ps
b.	500ps	150ps	100ps	180ps	220ps	1000ps	90ps	20ps	55ps

4.10.1 [10] <4.2, 4.4> To avoid lengthening the critical path of the datapath shown in Figure 4.24, how much time can the control unit take to generate the MemWrite signal.

4.10.2 [20] <4.2, 4.4> Which control signal in Figure 4.24 has the most slack and how much time does the control unit have to generate it if it wants to avoid being on the critical path?

4.10.3 [20] <4.2, 4.4> Which control signal in Figure 4.24 is the most critical to generate quickly and how much time does the control unit have to generate it if it wants to avoid being on the critical path?

The remaining problems in this exercise assume that the time needed by the control unit to generate individual control signals is as follows:

	RegDst	Jump	Branch	MemRead	MemtoReg	ALUOp	MemWrite	ALUSrc	RegWrite
a.	720ps	730ps	600ps	400ps	700ps	200ps	710ps	200ps	800ps
b.	1600ps	1600ps	1400ps	500ps	1400ps	400ps	1500ps	400ps	1700ps

4.10.4 [20] <4.4> What is the clock cycle time of the processor?

4.10.5 [20] <4.4> If you can speed up the generation of control signals, but the cost of the entire processor increases by $1 for each 5ps improvement of a single control signal, which control signals would you speed up and by how much to maximize performance? What is the cost (per processor) of this performance improvement?

4.10.6 [30] <4.4> If the processor is already too expensive, instead of paying to speed it up as we did in 4.10.5, we want to minimize its cost without further slowing it down. If you can use slower logic to implement control signals, saving $1 of the processor cost for each 5ps you add to the latency of a single control signal, which control signals would you slow down and by how much to reduce the processor's cost without slowing it down?

Exercise 4.11

In this exercise we examine in detail how an instruction is executed in a single-cycle datapath. Problems in this exercise refer to a clock cycle in which the processor fetches the following instruction word:

	Instruction word
a.	10001100010000110000000000010000
b.	00010000001000110000000000001100

4.11.1 [5] <4.4> What are the outputs of the sign-extend and the jump "Shift left 2" unit (in the upper left of Figure 4.24) for this instruction word?

4.11.2 [10] <4.4> What are the values of ALU control unit's inputs for this instruction?

4.11.3 [10] <4.4> What is the new PC address after this instruction is executed? Highlight the path through which this value is determined.

The remaining problems in this exercise assume that data memory is all-zeros and that the processor's registers have the following values at the beginning of the cycle in which the above instruction word is fetched:

	$0	$1	$2	$3	$4	$5	$6	$8	$12	$31
a.	0	1	2	3	–4	5	6	8	1	–32
b.	0	–16	–2	–3	4	–10	–6	–1	8	–4

4.11.4 [10] <4.4> For each Mux, show the values of its data output during the execution of this instruction and these register values.

4.11.5 [10] <4.4> For the ALU and the two add units, what are their data input values?

4.11.6 [10] <4.4> What are the values of all inputs for the "Registers" unit?

Exercise 4.12

In this exercise, we examine how pipelining affects the clock cycle time of the processor. Problems in this exercise assume that individual stages of the datapath have the following latencies:

	IF	ID	EX	MEM	WB
a.	300ps	400ps	350ps	500ps	100ps
b.	200ps	150ps	120ps	190ps	140ps