

4.1 Consider the following instruction:

Instruction: AND Rd, Rs, Rt

Interpretation: $\text{Reg}[\text{Rd}] = \text{Reg}[\text{Rs}] \text{ AND } \text{Reg}[\text{Rt}]$

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

4.1.1 The values of the signals are as follows:

RegWrite	MemRead	ALUMux	MemWrite	ALUop	RegMux	Branch
0	0	1 (Imm)	1	AND	X	0

ALUMux is the control signal that controls the Mux at the ALU input, 0 (Reg) selects the output of the register file, and 1 (Imm) selects the immediate from the instruction word as the second input to the ALU.

RegMux is the control signal that controls the Mux at the Data input to the register file, 0 (ALU) selects the output of the ALU, and 1 (Mem) selects the output of memory.

A value of X is a “don’t care” (does not matter if signal is 0 or 1)

4.1.2 All except branch Add unit and write port of the Registers

4.1.3 Outputs that are not used: Branch Add, write port of Registers

No outputs: None (all units produce outputs)

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 Instruction Set Architecture (ISA), but the decision whether or not to do that depends, among other things, on the cost and complexity the proposed addition introduces into the processor datapath and control. The first three problems in this exercise refer to the new instruction:

Instruction: LWI Rt, Rd(Rs)

Interpretation: $\text{Reg}[\text{Rt}] = \text{Mem}[\text{Reg}[\text{Rd}] + \text{Reg}[\text{Rs}]]$

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?

4.2.1 This instruction uses instruction memory, both register read ports, the ALU to add Rd and Rs together, data memory, and write port in Registers.

4.2.2 None. This instruction can be implemented using existing blocks.

4.2.3 None. This instruction can be implemented without adding new control signals. It only requires changes in the Control logic.

4.4 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
200ps	70ps	20ps	90ps	90ps	250ps	15ps	10ps

4.4.1 [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?

4.4.2 [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?

4.4.3 [10] <§4.3> Repeat 4.4.2, but this time we need to support only conditional PC-relative branches.

The remaining three problems in this exercise refer to the datapath element Shift-left-2:

4.4.4 [10] <§4.3> Which kinds of instructions require this resource?

4.4.5 [20] <§4.3> For which kinds of instructions (if any) is this resource on the critical path?

4.4.6 [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.

4.4.1 I-Mem takes longer than the Add unit, so the clock cycle time is equal to the latency of the I-Mem:

200 ps

4.4.2 The critical path for this instruction is through the instruction memory, Sign-extend and Shift-left-2 to get the offset, Add unit to compute the new PC, and Mux to select that value instead of PC+4. Note that the path through the other Add unit is shorter, because the latency of I-Mem is longer than the latency of the Add unit. We have:

$200 \text{ ps} + 15 \text{ ps} + 10 \text{ ps} + 70 \text{ ps} + 20 \text{ ps} = 315 \text{ ps}$

4.4.3 Conditional branches have the same long-latency path that computes the branch address as unconditional branches do. Additionally, they have a long-latency path that goes through Registers, Mux, and ALU to compute the PCSrc condition. The critical path is the longer of the two, and the path through PCSrc is longer for these latencies:

$200 \text{ ps} + 90 \text{ ps} + 20 \text{ ps} + 90 \text{ ps} + 20 \text{ ps} = 420 \text{ ps}$

4.4.4 PC-relative branches.

4.4.5 PC-relative unconditional branch instructions. We saw in part c that this is not on the critical path of conditional branches, and it is only needed for PC-relative branches. Note that MIPS does not have actual unconditional branches (bne zero,zero,Label plays that role so there is no need for unconditional branch opcodes) so for MIPS the answer to this question is actually “None”.

4.4.6 Of the two instructions (BNE and ADD), BNE has a longer critical path so it determines the clock cycle time. Note that every path for ADD is shorter than or equal to the corresponding path for BNE, so changes in unit latency

will not affect this. As a result, we focus on how the unit’s latency affects the critical path of BNE.

This unit is not on the critical path, so the only way for this unit to become critical is to increase its latency until the path for address computation through sign extend, shift left, and branch add becomes longer than the path for PCSrc through registers, Mux, and ALU. The latency of Regs, Mux, and ALU is 200 ps and the latency of Sign-extend, Shift-left-2, and Add is 95 ps, so the latency of Shift-left-2 must be increased by 105 ps or more for it to affect clock cycle time.

4.5 For the 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
20%	20%	0%	25%	25%	10%

4.5.1 [10] <§4.3> In what fraction of all cycles is the data memory used?

4.5.2 [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?

4.5.1 The data memory is used by LW and SW instructions, so the answer is:

$$25\% + 10\% = 35\%$$

4.5.2 The sign-extend circuit is actually computing a result in every cycle, but its output is ignored for ADD and NOT instructions. The input of the sign-extend circuit is needed for ADDI (to provide the immediate ALU operand), BEQ (to provide the PC-relative offset), and LW and SW (to provide the offset used in addressing memory) so the answer is:

$$20\% + 25\% + 25\% + 10\% = 80\%$$

4.6 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 bit 0 of the Write Register input on the register file in [Figure 4.24](#).

4.6.1 [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?

4.6.2 [10] <§§4.3, 4.4> Repeat 4.6.1 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.

4.6.3 [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.

4.6.4 [10] <§§4.3, 4.4> Repeat 4.6.1, but now the fault to test for is whether the “MemRead” control signal becomes 0 if RegDst control signal is 0, no fault otherwise.

4.6.5 [10] <§§4.3, 4.4> Repeat 4.6.4, but now the fault to test for is whether the “Jump” control signal becomes 0 if RegDst control signal is 0, no fault otherwise.

4.6.1 To test for a stuck-at-0 fault on a wire, we need an instruction that puts that wire to a value of 1 and has a different result if the value on the wire is stuck at zero:

If this signal is stuck at zero, an instruction that writes to an odd-numbered register will end up writing to the even-numbered register. So if we place a value of zero in R30 and a value of 1 in R31, and then execute ADD R31,R30,R30 the value of R31 is supposed to be zero. If bit 0 of the Write Register input to the Registers unit is stuck at zero, the value is written to R30 instead and R31 will be 1.

4.6.2 The test for stuck-at-zero requires an instruction that sets the signal to 1, and the test for stuck-at-1 requires an instruction that sets the signal to 0. Because the signal cannot be both 0 and 1 in the same cycle, we cannot test the same signal simultaneously for stuck-at-0 and stuck-at-1 using only one instruction. The test for stuck-at-1 is analogous to the stuck-at-0 test:

We can place a value of zero in R31 and a value of 1 in R30, then use ADD R30,R31,R31 which is supposed to place 0 in R30. If this signal is stuck-at-1, the write goes to R31 instead, so the value in R30 remains 1.

4.6.3 We need to rewrite the program to use only odd-numbered registers.

4.6.4 To test for this fault, we need an instruction whose MemRead is 1, so it has to be a load. The instruction also needs to have RegDst set to 0, which is the case for loads. Finally, the instruction needs to have a different result if

MemRead is set to 0. For a load, MemRead=0 result in not reading memory, so the value placed in the register is “random” (whatever happened to be at the output of the memory unit). Unfortunately, this “random” value can be the same as the one already in the register, so this test is not conclusive.

4.6.5 To test for this fault, we need an instruction whose Jump is 1, so it has to be the jump instruction. However, for the jump instruction the RegDst signal is “don’t care” because it does not write to any registers, so the implementation may or may not allow us to set RegDst to 0 so we can test for this fault. As a result, we cannot reliably test for this fault.

4.7 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:

10101100011000100000000000010100.

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:

r0	r1	r2	r3	r4	r5	r6	r8	r12	r31
0	-1	2	-3	-4	10	6	8	2	-16

4.7.1 [5] <§4.4> What are the outputs of the sign-extend and the jump “Shift left 2” unit (near the top of Figure 4.24) for this instruction word?

4.7.2 [10] <§4.4> What are the values of the ALU control unit’s inputs for this instruction?

4.7.3 [10] <§4.4> What is the new PC address after this instruction is executed? Highlight the path through which this value is determined.

4.7.4 [10] <§4.4> For each Mux, show the values of its data output during the execution of this instruction and these register values.

4.7.5 [10] <§4.4> For the ALU and the two add units, what are their data input values?

4.7.6 [10] <§4.4> What are the values of all inputs for the “Registers” unit?

4.7.1

Sign-extend	Jump’s shift-left-2
000000000000000000000000010100	0001100010000000000001010000

4.7.2

ALUOp[1-0]	Instruction[5-0]
00	010100

4.7.3

New PC	Path
PC+4	PC to Add (PC+4) to branch Mux to jump Mux to PC

4.7.4

WrReg Mux	ALU Mux	Mem/ALU Mux	Branch Mux	Jump Mux
2 or 0 (RegDst is X)	20	X	PC+4	PC+4

4.7.5

ALU	Add (PC+4)	Add (Branch)
-3 and 20	PC and 4	PC+4 and 20*4

4.7.6

Read Register 1	Read Register 2	Write Register	Write Data	RegWrite
-----------------	-----------------	----------------	------------	----------