4.2, 4.5, 4.8.1-5, 4.13.1-3

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: Reg[Rt] = Mem[Reg[Rd]+Reg[Rs]]

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

Other than typically required blocks such as the PC and instruction memory, the registers block, ALU, and memory would be needed to add the registers and go to specified location in memory.

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

I don’t believe there needs to be new blocks to perform this function. There might be a problem in that the registers would need to be used as the memory address itself instead of an area of memory being pointed at by the register.

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

There shouldn’t be a necessity for new signals. The required blocks appear to be controlled sufficiently by the control block.

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:

(picture)

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

Data memory is used 35% of the time since store and load word is used 35% of the time.

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?

The sign-extension circuit is being used 60% of the time. This is for the branch, load, and store operations. If the instruction being executed does not need the sign-extension circuit, it is still applied, but the added sign bits are thrown away.

4.8 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:

Also, assume that instructions executed by the processor are broken down as follows:

4.8.1 [5] <§4.5> What is the clock cycle time in a pipelined and non-pipelined processor?

It would be at least twice as fast. A pipelined processor would run four instructions in the same time that a non-pipelined one would run two. Due to pipelining, other instructions will be executed at the same time, so the pipelined processor will actually be around three to four times as fast.

4.8.2 [10] <§4.5> What is the total latency of an LW instruction in a pipelined and non-pipelined processor?

4.8.3 [10] <§4.5> If we can split one stage of the pipelined datapath into two new stages, each with half the latency of the original stage, which stage would you split and what is the new clock cycle time of the processor?

4.8.4 [10] <§4.5> Assuming there are no stalls or hazards, what is the utilization of the data memory?

4.8.5 [10] <§4.5> Assuming there are no stalls or hazards, what is the utilization of the write-register port of the “Registers” unit?

4.13.1 [5] <§4.7> If there is no forwarding or hazard detection, insert nops to ensure correct execution.

4.13.2 [10] <§4.7> Repeat 4.13.1 but now use nops only when a hazard cannot be avoided by changing or rearranging these instructions. You can assume register R7 can be used to hold temporary values in your modified code.

4.13.3 [10] <§4.7> If the processor has forwarding, but we forgot to implement the hazard detection unit, what happens when this code executes?