Ex4: 4.4, 4.7, 4.9.1~4.9.4, 4.11, 4.14.1~4.14.3, 4.18.1~4.18.4
due April 18th (mid-term exam).

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.

4.7 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?

4.9 In this exercise, we examine how data dependences affect execution in the basic 5-stage pipeline described in Section 4.5. Problems in this exercise refer to the following sequence of instructions:

```
or r1,r2,r3
or r2,r1,r4
or r1,r1,r2
```

Also, assume the following cycle times for each of the options related to forwarding:

Without Forwarding	With Full Forwarding	With ALU-ALU Forwarding Only
250ps	300ps	290ps

4.9.1 [10] <§4.5> Indicate dependences and their type.

4.9.2 [10] <§4.5> Assume there is no forwarding in this pipelined processor. Indicate hazards and add `nop` instructions to eliminate them.

4.9.3 [10] <§4.5> Assume there is full forwarding. Indicate hazards and add `NOP` instructions to eliminate them.

4.9.4 [10] <§4.5> What is the total execution time of this instruction sequence without forwarding and with full forwarding? What is the speedup achieved by adding full forwarding to a pipeline that had no forwarding?

4.11 Consider the following loop.

```
loop:lw  r1,0(r1)
     and r1,r1,r2
     lw  r1,0(r1)
     lw  r1,0(r1)
     beq r1,r0,loop
```

Assume that perfect branch prediction is used (no stalls due to control hazards), that there are no delay slots, and that the pipeline has full forwarding support. Also assume that many iterations of this loop are executed before the loop exits.

4.11.1 [10] <§4.6> Show a pipeline execution diagram for the third iteration of this loop, from the cycle in which we fetch the first instruction of that iteration up to (but not including) the cycle in which we can fetch the first instruction of the next iteration. Show all instructions that are in the pipeline during these cycles (not just those from the third iteration).

4.11.2 [10] <§4.6> How often (as a percentage of all cycles) do we have a cycle in which all five pipeline stages are doing useful work?

4.14 This exercise is intended to help you understand the relationship between delay slots, control hazards, and branch execution in a pipelined processor. In this exercise, we assume that the following MIPS code is executed on a pipelined processor with a 5-stage pipeline, full forwarding, and a predict-taken branch predictor:

```
        lw r2,0(r1)
label1: beq r2,r0,label2 # not taken once, then taken
        lw r3,0(r2)
        beq r3,r0,label1 # taken
        add r1,r3,r1
label2: sw r1,0(r2)
```

4.14.1 [10] <§4.8> Draw the pipeline execution diagram for this code, assuming there are no delay slots and that branches execute in the EX stage.

4.14.2 [10] <§4.8> Repeat 4.14.1, but assume that delay slots are used. In the given code, the instruction that follows the branch is now the delay slot instruction for that branch.

4.14.3 [20] <§4.8> One way to move the branch resolution one stage earlier is to not need an ALU operation in conditional branches. The branch instructions would be "`bez rd,label`" and "`bnez rd,label`", and it would branch if the register has and does not have a zero value, respectively. Change this code to use these branch instructions instead of `beq`. You can assume that register R8 is available for you to use as a temporary register, and that an `seq` (set if equal) R-type instruction can be used.

4.18 In this exercise we compare the performance of 1-issue and 2-issue processors, taking into account program transformations that can be made to optimize for 2-issue execution. Problems in this exercise refer to the following loop (written in C):

```
for(i=0;i!=j;i+=2)
  b[i]=a[i]-a[i+1];
```

When writing MIPS code, assume that variables are kept in registers as follows, and that all registers except those indicated as Free are used to keep various variables, so they cannot be used for anything else.

i	j	a	b	c	Free
R5	R6	R1	R2	R3	R10, R11, R12

4.18.1 [10] <§4.10> Translate this C code into MIPS instructions. Your translation should be direct, without rearranging instructions to achieve better performance.

4.18.2 [10] <§4.10> If the loop exits after executing only two iterations, draw a pipeline diagram for your MIPS code from 4.18.1 executed on a 2-issue processor shown in Figure 4.69. Assume the processor has perfect branch prediction and can fetch any two instructions (not just consecutive instructions) in the same cycle.

4.18.3 [10] <§4.10> Rearrange your code from 4.18.1 to achieve better performance on a 2-issue statically scheduled processor from Figure 4.69.

4.18.4 [10] <§4.10> Repeat 4.18.2, but this time use your MIPS code from 4.18.3.