4.1、4.3、4.5、4.7、4.10、4.15

4.1

(1)

ALUSrc=0

#RegWrite = 1

#ALUOp = 1

MemWrite = 0

MemRead = 0

MemToReg=0

#RegWrite = 1

#RegMux = 1

#ALUMux = 0

#Branch = 0

(2)Registers（寄存器堆）、ALUsrc（ALU 程序寄存器）、ALU、MemToReg mux

(3)所有模块都会产生输出；数据存储单元和立即数生成器的输出没有被使用。

4.3

(1)25%+10%=35%

(2)100%

(3)28%+25%+10%+11%+2%=76%

(4)符号扩展在每个周期都会产生一个输出，如果它的输出不需要，就会被简单地忽略。

4.5

(1)

ALUOp：00

ALU Control Lines：0010

(2)PC_new=PC _initial+4

(3)

ALUsrc: Inputs: Reg[x12] and 0x0000000000000014; Output: 0x0000000000000014

MemToReg: Inputs: Reg[x13] + 0x14 and <undefined>; Output: <undefined>

Branch: Inputs: PC+4 and PC + 0x28

(4)

ALU inputs: Reg[x13] and 0x0000000000000014

PC+4 adder inputs: PC and 4

Branch adder inputs: PC and 0x0000000000000028

(5)

Read register 1 = 0x13 (base address)

Read register 2 = 0x12 (data to be stored)

Write register = 0x0 or don't care (should not write back)

Write data = don't care (should not write back)

RegWrite = false (should not write back)

4.7
(1)30+250+150+25+200+25+20=700ps
(2)30+250+150+25+200+250+25+20=950ps
(3)30+250+150+200+25+250=905ps
(4)30+250+150+25+200+5+25+20=705ps
(5)30+250+150+25+200+25+20=700ps
(6)950ps

4.10
(1)
额外的寄存器将允许我们减少 12%的加载和存储操作，或对所有指令有 0.12*(0.25+0.1)=4.2%。
因此，运行 n 条指令的时间将从 950n 减少到 960*0.958*n=919.68*n。
所以加速比为 950/919.68=1.03
(2)
原始 CPU 的成本是 4496；改进后的 CPU 成本是 4696。
PC: 5
I-Mem: 1000
Register file: 200
ALU: 100
D-Mem: 2000
Sign Extend: 100
Controls: 1000
adders: 30*2
muxes: 3*10
single gates: 1*1

性能增加 3%，CPU 成本增加约 4.4%。
(3)从严格的数学角度来看，增加更多寄存器是没有意义的，因为新 CPU 每单位性能的成本更高。然而，这个简单的计算并没有考虑到性能的实用性。例如，在实时系统中，3%的性能提升可能就是满足或错过截止时间的差别。在这种情况下，为了这 4.4%的额外成本所带来的改进是值得的。

4.15
(1)新的时钟周期时间将是 750。ALU 和数据存储现在将并行运行，因此我们实际上已经将从两个中较快的那个（ALU，时间为 200）从关键路径中移除了。
(2)变慢了。原始 CPU 需要 950n ps 来运行 n 条指令。当为新型机器编译时，相同的程序将有大约 1.35n 条指令。因此，在新机器上的时间将是 750*1.35n=1012.5n。加速比为 0.93，标明执行变慢了。
(3)ls 和 sw 指令的数量是主要因素。ls 和 sw 的使用方式也可能产生影响。例如，如果一个程序的 ls 和 sw 操作倾向于只针对几个不同的地址，那么它也可能在新机器上运行得更快。
(4)答案合理即给分。