一、

1、

控制单元产生信号

	RegDst	RegWrite	MemRead	ALUMux	MemWrite	ALUOp	RegMux	Branch
a.	1	1	0	0(Reg)	0	Add	0(ALU)	0
b.	0	1	1	1(Imm)	0	Add	1(Mem)	0

ALUMux 代表连接在 ALU 输入端的多选器的控制信号。其中， 0(Reg)代表多选器选择的是寄存器堆的输出；1(Imm)代表多选器选择的是指令存储器中的立即数。

RegMux 代表连接在寄存器堆的 Data 输入端的多选器的控制信号。其中，0(ALU)代表多选器选择的是 ALU 的输出，1(Mem)代表多选器选择的是数据存储器的输出。

2、

对于指令 a. 除数据存储器和分支加法器(图 1 中靠右侧的 Add)之外的所有单元

对于指令 b. 除分支加法器之外的所有单元

3、

	产生输出但没用到	无输出
a	分支加法器	数据存储器
b	分支加法器，寄存器堆的第二个输出端 (图 1 中寄存器堆两个输出端中下面的一个)	无

4、

关键路径为：I-Mem、Regs(Read)、Mux、ALU、Mux、Regs(Write)

5、

关键路径为：I-Mem、Regs(Read)、Mux、ALU、D-Mem(Read)、Mux、Regs(Write)

6、

由于控制器的速度快于寄存器堆，故关键路径为：I-Mem、Regs(Read)、Mux、ALU、Mux

二、

1、

a.除分支加法器、数据存储器之外的所有单元

b.除分支加法器、数据存储器之外的所有单元，此外，ALU 可以选择继续使用也可以选择不

继续使用，具体要视第②问的回答，如果增加一个专门的移位器，则不需要使用 ALU

2、
a.寄存器堆增加一个输出端 和一个相应的读地址输入端，以读出 Rx；增加一个加法器(或为现有的 ALU 增加一个输入端)，加法器的一个输入端连接至寄存器堆新增的输出端，另一个输入端连接至 ALU 的输出端。如果采用增加一个加法器的方案，则还需增加寄存器堆数据输入选通器的一个输入端，并连接至加法器的输出端
b.增加一个移位器(或为现有的 ALU 增加移位运算功能)，移位器的输入端连接至寄存器堆的一个输出端。如果采用增加一个移位器的方案，则还需增加寄存器堆数据输入选通器的一个输入端，并连接至移位器的输出端

3、
a.如果增加一个加法器，则需增加寄存器堆数据输入选通器的控制信号，以实现数据通道 3 选 1；如果为现有的 ALU 增加一个输入端，则需增加针对三输入端的 ALU 的功能控制信号定义，使其可控制新增的 ADD3 操作
b.如果增加一个移位器，则需增加寄存器堆数据输入选通器的控制信号，以实现数据通道 3 选 1(如同时考虑 a 和 b 中的两条指令，则为 4 选 1)；如为现有的 ALU 增加移位运算功能，则需增加 ALU 的功能控制信号定义，使其可控制新增的移位操作

4、
a. 由于加法单元不在关键路径上，因此对加法器的改进不影响时钟周期。
b. 寄存器堆位于关键路径上，因而，使用更大的寄存器堆后，时钟周期变为 1330ps + 2 × 100ps = 1530ps。

5、
a.加速比由时钟周期本身的变化以及需要执行的时钟周期数目共同决定，对加法器的改进不影响时钟周期，并且，需要执行的时钟周期数目也不变，因此加速比为 1.000
b.需要的指令数减少 5%，需要的时钟周期数目也相应减少 5%，同时，时钟周期由 1330ps 增加为 1530ps，因而加速比为 (1/0.95)×(1330/1530) = 0.915

6、
a. 原来的处理器的总成本为 1000(I-Mem) + 200(Regs) + 500(Control) + 100(ALU) + 2000(D-Mem) + 2×30(2 个加法单元) + 3×10(3 个多选器) = 3890，更换加法器之后的总成本为 3890 + 2 × 20 = 3930，相对成本为 3930 / 3890 = 1.010，性能/价格比为 1.000 / 1.010 = 0.990，成本增加但性能没有提升。
b. 使用更大的寄存器堆的成本为 3890 + 200 = 4090，相对成本为 4090 / 3890 = 1.051，性能/价格比为 0.915 / 1.051 = 0.871，说明用更大的投入反而换来了性能的下降。

三、
1、
a. 由于指令存储器慢于加法器，因此，时钟周期决定于指令存储器的延迟，时钟周期为 400ps。
b. 时钟周期为 500ps。

2、
a.关键路径为 I-Mem、Sign-extend、Shift-left-2、Add(ALU)、Mux，因此，时钟周期为 400ps + 20ps + 2ps + 120ps + 30ps = 572ps。
b.时钟周期为 500ps + 90ps + 20ps + 180ps + 100ps = 890ps。

3、
a.根据题目中给出的延迟，后者为关键路径，因此时钟周期为 400ps + 200ps + 30ps + 120ps + 30ps = 780ps。
b.根据题目中给出的延迟，后者为关键路径，因此时钟周期为 500ps + 220ps + 100ps + 180ps + 100ps = 1100ps。

4、
a.所有指令。
b.与存取有关的指令，如：lw, sw 等。

5、
a.所有指令的关键路径都不会包含这个加法器，因为指令存储器的速度慢于加法器，而所有的指令都必须执行读指令这一操作。
b.与存取有关的指令，因为只有与存取有关的指令会用到该单元。

6、
a.beq 指令的关键路径为 I-Mem、Regs(Read)、Mux、ALU、Mux，延迟为 780ps，add 指令的关键路径为 I-Mem、Regs(Read)、Mux、ALU、Mux、Regs(Write)，延迟为 980ps。这里只需讨论该单元延迟变化相比于较长的关键路径的影响，较长的关键路径为 add 指令，其延迟为 980ps，而执行加 4 的加法器所在的路径为 Add、Mux，其延迟为 100ps + 30ps = 150ps，若要该单元延迟变化而导致该路径成为关键路径，从而影响时钟周期，则执行加 4 的加法器延迟要大于 980ps – 150ps = 830ps，才会影响时钟周期。
b.数据存储器未被 beq 或 add 指令使用，因此，它的延迟不影响时钟周期。

四、
1、
a.时钟周期为 400ps + 200ps + 30ps + 120ps + 30ps + 200ps = 980ps
b.时钟周期为 500ps + 220ps + 100ps + 180ps + 100ps + 220ps = 1320ps

2、
a.时钟周期为 400ps + 200ps + 30ps + 120ps + 350ps + 30ps + 200ps = 1330ps
b.时钟周期为 500ps + 220ps + 100ps + 180ps + 1000ps + 100ps + 220ps = 2320ps

3、
a.时钟周期为 1330ps
b.时钟周期为 2320ps

4、
a.平均有 20% + 10% = 30% 的时钟周期里，会用到数据存储器

b.平均有 35% + 15% = 50% 的时钟周期里，会用到数据存储器

5、
符号扩展电路实际上在每个周期都有计算结果，但是它的输出在 add 和 not 指令中被忽略了，符号扩展电路的输入只在 addi 指令(提供 ALU 需要的立即数)、beq 指令(提供计算 PC 需要的偏移量)、lw 指令和 sw 指令(提供寻址过程中需要的偏移量)中是需要的
a. 结果为 15% + 20% + 20% + 10% = 65%
b. 结果为 5% + 15% + 35% + 15% = 70%

6、
lw 指令有最长的关键路径为：I-Mem、Registers(Read)、Mux、DMem(Read)、ALU、Mux、Registers(Write)，决定着时钟周期的长度。

a. 在路径中，由于指令存储器的延迟值最大，因此，如将它的延迟从 400ps 减小到 360ps(即减少 10%)，则时钟周期由 1330ps 减小到 1290ps，加速比为 1330/1290=1.031。

b. 在路径中，由于数据存储器的延迟值最大，因此，如将它的延迟从 1000ps 减小到 900ps(即减少 10%)，则时钟周期由 2320ps 减小到 2220ps，于是加速比为 2320/2220=1.045。

五、
1、
a.为了测试是否有固定为 0 缺陷，我们需要一个指令，该指令可以将待测信号置为 1，并且与 0 值有着不同的输出，指令存储器的第 7 位输出只用于指令的立即数或者偏移量部分，因而可以采用指令 ADDI $1,$0,128，该指令可以将 128 放置到寄存器$1 中，如果指令存储器的第 7 位输出有固定为 0 缺陷，那么寄存器$1 中的值就会为 0 而不是 128。

b.能够将输出信号 MemtoReg 置为 1 的只有 load 类指令，我们可以将数据存储器中的每个字都置为 0，然后执行 LW $1,1024($0)，如果寄存器$1 中的值不是 0 而是 1024，说明输出信号 MemtoReg 有固定为 0 缺陷。

2、
检查固定为 0 缺陷需要能够将该信号置为 1 的指令，而检查固定为 1 缺陷需要能够将该信号置为 0 的指令，由于在一个周期中一个信号不可能既为 0 又为 1，因而不能同时检查固定为 0 缺陷和固定为 1 缺陷。若要检查固定为 1 缺陷，与固定为 0 缺陷类似，可采用如下方法：
a.执行指令 ADDI $1,$0,0，如果指令存储器的第 7 位输出有固定为 1 缺陷，那么寄存器$1 中的值就会为 128 而不是 0。

b.这个信号的固定为 1 缺陷不能准确检查出来，因为所有能够将 MemtoReg 信号设置为 0 的指令都会同时将 MemRead 信号设置为 0，这样会导致写入到寄存器$1 中的数据是不确

定的(与我们在检测固定为 0 缺陷时刻意放置的数据没有关系)，有可能最后出现在寄存器$1 中的数据是与原来寄存器中的数据相同的，从而检测不出来固定为 1 缺陷。

3、
a.要避开指令存储器输出信号的固定为 1 缺陷是有可能的，但是比较麻烦，我们必须找到在输出信号第 7 位上为 0 的所有指令(一般是 I 型指令)，然后将这些指令中的立即数进行相应的替代。例如，对于一个第 7 位为 0 的 LW $1,0($0)指令，需要用 LI $1,128、SUB $1,$0,$1 和 LW $1,128($1)代替。

b. MemtoReg 信号的固定为 1 缺陷是不能回避的，MemtoReg 信号的固定为 1 缺陷，会阻止除 load 类指令外的所有指令向寄存器堆中写入数据，而 load 类指令只能将数据从存储器载入到寄存器中，不能模仿 ALU 的运算操作。

4、
a.如果 MemRead 存在固定为 1 缺陷，那么在每一条指令执行的时候都会读取数据存储器，然而对于非 load 类指令，从存储器读出的数据会被多选器抛弃，这导致我们无法设计出检测该缺陷的方法，因为即使存在该缺陷，处理器也是正常工作的。

b.为了测试该缺陷我们需要一个操作码为 0 同时 MemRead 信号为 1 的指令，但是操作码为 0 的指令都是 ALU 的 R 型指令，不是 load 指令，因而它们的 MemRead 信号都为 0，因此，即使存在该缺陷，处理器也是正常工作的，因而我们无法设计出检测该缺陷的方法。

5、
a.如果 Jump 信号存在固定为 1 缺陷，那么每一条指令执行时都会按照执行 J 指令时的方法更新 PC 的值，要检测该缺陷，可以将一条非跳转指令(例如 ADD $1,$0,$0)放在指令存储器中第一条指令的位置，该指令执行之后 PC 的值应当是 0x00000004，若 PC 的值变为 0x00002080 则说明 Jump 信号存在固定为 1 缺陷。

b.为了测试该缺陷我们需要一个操作码为 0 同时 Jump 信号为 1 的指令，然而跳转指令的操作码都不是 0，因此我们无法设计出检测该缺陷的方法，此时处理器是正常工作的。

6、
涉及到 5 个控制信号：RegDst、Jump、Branch、MemtoReg 和 ALUSrc，分别对每一个信号设计测试方案进行测试即可，但是注意有些信号的缺陷 a 或者缺陷 b 是无法检测出来的。

a.RegDst 信号的固定为 1 缺陷可用 lw 指令测试，首先可以将数据存储器全写为 1，执行 lw $1,0x1000($0)，如果载入的字出现在了$2 中而不是$1 中说明 RegDst 存在固定为 1 缺陷。

Jump 的固定为 1 缺陷测试方法同(5)a。

Branch 的固定为 1 缺陷可以用非分支指令测试，也可以用测试 Jump 时同样的方法，ADD $1,$0,$0 指令会使 ALU 的 Zero 输出为 1，导致分支条件“满足”，若执行之后 PC 的值变为 0x00002084 而不是 0x00000004 则说明 Branch 存在固定为 1 缺陷。

MemtoReg 信号的固定为 1 缺陷无法准确检测，原因参见(2)b。

ALUSrc 的固定为 1 缺陷也可以用 ADD $1,$0,$0 指令测试，若执行之后$1 的值为 0x00000820 而不是 0，则说明存在缺陷。

涉及到 5 个控制信号：RegDst、Jump、Branch、MemtoReg 和 ALUSrc，分别对每一个信号设计测试方案进行测试即可，但是注意有些信号的缺陷 a 或者缺陷 b 是无法检测出来的。

b.操作码为 0 的指令都是 ALU 的 R 型指令，对于这些指令 Jump、Branch、MemtoReg 以及 ALUSrc 本就应该为 0，所以这些信号的缺陷无法检测。

RegDst 的缺陷可以用 ADD $1,$2,$3 来测试，若执行完之后$2+$3 的结果出现在了$3 而不是$1 中，说明存在缺陷。

六、

1、
a. 100011 00110 00001 0000000000101000 0X8CC10028
b. 000101 00001 00010 1111111111111111 0X1422FFFF

2、

	RD1	是否被读	RD2	是否被读
a.	6	是	1	是
b.	1	是	2	是

3、

	WR	是否被写
a.	1	是
b.	2	否

4、

	控制信号 1	控制信号 2
a.	**RegDst = 0**	**MemRead = 1**
b.	RegWrite = 0	MemRead = 0

5、

$$\text{RegDst} = \overline{\text{OP5}}\ \overline{\text{OP4}}\ \overline{\text{OP3}}\ \overline{\text{OP2}}\ \overline{\text{OP1}}\ \overline{\text{OP0}}$$

$$\text{MemRead} = \text{OP5}\ \overline{\text{OP4}}\ \overline{\text{OP3}}\ \overline{\text{OP2}}\ \text{OP1}\ \text{OP0}$$

$$\text{RegWrite} = \text{OP5}\ \overline{\text{OP4}}\ \overline{\text{OP3}}\ \overline{\text{OP2}}\ \text{OP1}\ \text{OP0} + \overline{\text{OP5}}\ \overline{\text{OP4}}\ \overline{\text{OP3}}\ \overline{\text{OP2}}\ \overline{\text{OP1}}\ \overline{\text{OP0}}$$

$$\text{MemRead} = \text{OP5}\ \overline{\text{OP4}}\ \overline{\text{OP3}}\ \overline{\text{OP2}}\ \text{OP1}\ \text{OP0}$$

七、

1、

a.关键路径延迟为 400ps + 200ps + 30ps + 120ps + 350ps + 30ps + 200ps = 1330ps，留给控制单元产生 MemWrite 信号的最长时间为 1330ps – 400ps –350ps = 580ps

b.关键路径延迟为 500ps + 220ps + 100ps + 180ps + 1000ps + 100ps + 220ps = 2320ps，留给控制单元产生 MemWrite 信号的最长时间为 2320ps – 500ps –1000ps = 820ps

2、

a.Jump 信号具有最长的松弛时间，为 1330ps – 400ps – 30ps = 900ps

b.Jump 信号具有最长的松弛时间，为 2320ps – 500ps – 100ps = 1720ps

3、

a. ALUOp (50ps > 30ps)（最关键信号） 200ps + 30ps – 50ps = 180ps（可用时间）

b. ALUSrc (100ps > 55ps) （最关键信号） 220ps（可用时间）

4、

	RegDst	Jump	Branch	MemRead	MemtoReg	ALUOp	MemWrite	ALUSrc	RegWrite
a.	20ps	-170ps	-270ps	50ps	0ps	20ps	130ps	0ps	70ps
b.	100ps	-120ps	-220ps	0ps	-100ps	135ps	680ps	180ps	100ps

5、

	对时钟周期有影响的信号	代价
a.	RegDst (+20ps) MemRead (+50ps) ALUOp (+20ps) MemWrite (+130ps) RegWrite (+70ps)	290/5=58
b.	RegDst (+100ps) ALUOp (+135ps) MemWrite (+680ps) ALUSrc (+180ps) RegWrite (+100ps)	1195/5=239

6、

	RegDst	Jump	Branch	MemRead	MemtoReg	ALUOp	MemWrite	ALUSrc	RegWrite
a.	110ps	300ps	400ps	80ps	130ps	110ps	0ps	130ps	60ps
b.	580ps	800ps	900ps	680ps	780ps	545ps	0ps	500ps	580ps

八、

1、

a. 00000000 00000000 00000000 00010000（符号扩展）
0001 00001100 00000000 01000000（左移两位）

b. 00000000 00000000 00000000 00001100 （符号扩展）
0000 10001100 00000000 00110000（左移两位）

2、

	ALUOp	Instr
a	00	010000
b	01	001100

3、

	NPC	DATAPATH
a	PC + 4	PC 、 Add(PC + 4) 、 Mux(branch)、Mux(jump)、

		PC
b	如果$1 与$3 不相等，为 PC + 4 否则，为 PC + 4 + 4 × 12	PC 、 Add(PC + 4) 、Mux(branch)、Mux(jump)、PC 或者 PC、Add(PC + 4)、Add(偏移)、Mux(branch)、Mux(jump)、PC

4、

	WR MUX	ALU MUX	Mem	Branch MUX	Jump MUX
a	3	16	0	PC + 4	PC + 4
b	3 或者 0	-3	X	PC + 4	PC + 4

5、

	ALU	Add(PC + 4)	Add(Branch)
a	2 16	PC 4	PC + 4 64
b	-16 -3	PC 4	PC + 4 48

6、

	RD1	RD2	WR	WD	RW
a	2	3	3	0	1
b	1	3	X	X	0