11 题，100 分，10 分一题

目录

第二章：1 每年送分，5 会算 1

第三章：1、2、3、4 2

第四章：除了 1256，别的都看看 3

第五章：1、2、3、4 会画时空图、算周期、指令调度，7.1 写 verilog 5

verilog 题：ALU 5

第六章：5、6 7

第七章：4、5 10

第八章：1、2、3、6 11

第九章；2、3、5，画简单的加法器 13

Verilog 题：16 位先行进位加法器 14

第十章：1、2、3 15

第十一章：1 必考，但不是原题，3、5 17

Verilog 题：TLB 地址查找 18

第十二章：6、7、8 19

第二章：1每年送分，5会算

1. 在3台不同指令系统的计算机上运行同一程序P时，A机需要执行1.0×10^8条指令，B机需要执行2.0×10^8条指令，C机需要执行4.0×10^8条指令，但实际执行时间都是10s。请分别计算这3台机器在运行程序P时的实际速度，以MIPS为单位。这3台计算机在运行程序P时，哪一台性能最高？为什么？

1. 解：A为10MIPS，B为20MIPS，C为40MIPS。

三台机器实际性能相同。

题目问的是运行程序P时的性能。

5. 对某处理器进行功耗测试，得到如下数据：时钟不翻转，电压1.05V时，电流为500mA；时钟频率为1GHz，电压1.1V时，电流为2500mA。请计算在1.1V下，此处理器的静态功耗以及500MHz下的总功耗。

5. 解：

1.1V下静态功耗1.1*1.1/(1.05/0.5)=0.576W　时钟不翻转的静态功耗按电阻算

1.1V下1GHZ时动态功耗为1.1*2.5-0.576=2.174W　动态功耗+静态功耗=总功耗

1.1V下0.5GHZ动态功耗为2.174*0.5/1=1.087W　动态功耗与翻转率成正比

1.1V下0.5GHZ总功耗为1.087+0.576=1.663W

第三章：1、2、3、4

2. 按照以下要求将浮点数表示出来。

（1）把单精度数转化为十进制数：0x7ff0000，0xbe400000，0xff800000。

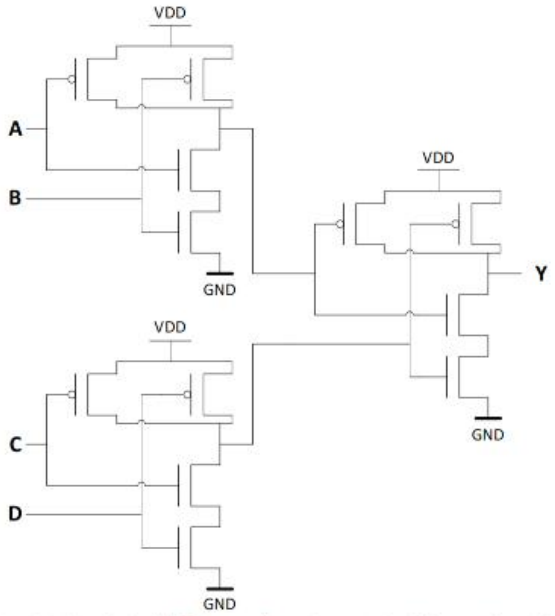
（2）把双精度数转化为十进制数：0x4035000000000000，0x8008000000000000。

（3）把十进制数转化为单精度数：－100.0，0.25。

（4）把十进制数转化为双精度数：1024.0，0.25。

3. 画出 $e=a\&b|c\&d$ 的晶体管级电路图。

4. 计算一个 FO4 的延迟，假设反相器的输入电容为 0.0036pF，平均每个负载连线电容为 0.0044pF，翻转延迟为 0.023ns，每 pF 延迟为 4.5ns。



两种解法，第二种，是课本上的~(A&B|C&D)后面再加一个反相器。
N 管要接地，P 管要接电，不能反过来。所以一级逻辑画不出的。注意！

4. 解：

FO4 延迟=本征延迟+负载延迟=0.023+4.5*((0.0036+0.0044)*4)=0.167ns

本课对 FO4 定义为 1 个反相器驱动 4 个相同的反相器（器件延迟和线延迟是 4 份）。鉴于 FO4 定义存疑（参见 wiki），回答 0.023+4.5*(0.0044+0.0036*4*4)=0.302ns 也算对。

第四章：除了1256，别的都看看

3. 假定在指令系统设计中需要考虑两种条件转移指令的设计方法，这两种方法如下。

(1) CPU A：先通过一条比较指令设置条件码 A，再用一条分支指令检测条件码。

(2) CPU B：比较操作包含在分支指令中。

在两种 CPU 中，条件转移指令都需要两个时钟周期，所有其他指令都需要一个时钟周期。在 CPU A 中，全部指令的 25%是条件转移指令，因为每次条件转移都需要一次比较，所以比较指令约占所有指令的 25%。因为 CPU A 不需要在转移中包含分支，所以它的时钟频率是 CPU B 的 1.2 倍。请问哪一种 CPU 性能更高？如果 CPU A 的时钟频率只是 CPU B 的 1.1 倍，结果又是多少？

题目问的是性能，可以默认问的是运行该程序（满足 25%是条件转移）的性能。

最终以实际运行的时间为标准；想计算时间，需要先计算总指令数量，才能算出所需时间。

解：假设 CPU A 总指令数为 x，根据题意，转移指令有 0.25x，条件码指令

0.25x，其它指令 0.5x，因此 A 执行周期数 2*0.25x+0.75x=1.25x

可得 CPU B 总指令数为 0.75x，其中转移指令 0.25x，其它指令 0.5x。

B 执行周期数 2*0.25x+0.5x=1x

当 CPU A 频率为 1.2 倍时，性能是 CPU B 的 1.2/1.25=0.96 倍

当 CPU A 频率为 1.1 倍时，性能是 CPU B 的 1.1/1.25=0.88 倍

因此 CPU A 两种情况下都差

些 load 数量。

```
LW R1,0(R3)
ADD R2, R2, R1
```

可以合并成一条指令：

```
ADD R2, 0(R3)
```

(1) 给出一段符合上述例子的代码(load 得到的值立即作为运算指令的源操作数)，但是编译器依然无法用寄存器—内存形式的指令消除这条 load 指令。

(2) 假设这样的修改带来了 5%的主频下降，同时没有 CPI 影响。如果 load 占所有指令的 26%，最少要消灭 load 指令的百分之多少，才能使得新指令集不导致性能下降？

(3) 在传统的静态 5 级流水线(IF ID EXE MEM WB)上，寄存器—内存形式的指令有何实现困难？

4.

a） lw $1, 0($n)

add $2, $2, $1

bnez $1, 1f　　//任何将$1 作为 src 的指令都可以

b） 假设需要减少 x 的 load 指令。减少后，指令数为 1-0.26x。则(1-0.26x)/0.95=1

x=19%

c） 困难在于访存 MEM 在 EXE 之前就要进行，而 add $2, 0($n)需要先访存后 EXE

题意是增加寄存器-内存形式指令，因此有的同学曾经回答“lw　r1,0(r3)　　add r3, r2, r1”，但可以通过增加 add r3, r2, 0(r3)这样的寄存器-内存形式指令来实现消除。虽然与题干不完全一致，但是是可以通过简单思考，就知道这个也属于“寄存器-内存”形式指令。

7. 用 MIPS 的 LL/SC 指令编写一段从内存单元 100(R2)取数、把取出来的数加 100 并存回到 100(R2)的原子操作代码，并说明如果在此过程中处理器发生中断或该单元被其他处理器修改时处理器如何保证上述操作的原子性。

```
1:  ll r1, 100(r2)
    add r1, r1, 100
    sc r1, 100(r2)
    beqz r1, 1b
    nop
```

如果在 ll 和 sc 之间（含 ll 和 sc），发生了中断或者其他处理器修改 100(r2)，则 sc 之后 r1 会变 0，回到标号 1 重新执行。

第五章：1、2、3、4 会画时空图、算周期、指令调度，7.1 写 verilog

```
LW    R1, 0(R0)
LW    R2, 4(R0)
ADD   R3, R1, R2      ;a=b+e
SW    R3, 12(R0)
LW    R4, 8(R0)
ADD   R5, R1, R4      ;c=b+f
SW    R5, 16(R0)
```

分析上述指令序列之间的相关，并重排序执行序列避免相关。重排序指令序列可以比原先指令序列的执行减少多少拍？

2. 对于下面的计算：

```
A=A+B
C=A-B
```

写出 MIPS 程序代码，并且画出 5 级静态流水线（无旁路）的流水线时空图。

3. 对于浮点向量运算 X(i)＝a＊X(i)＋Y(i)，假设 X 和 Y 的首地址分别存在定点寄存器 R1 和 R2 中，a 的值存在浮点寄存器 F0 中。

（1）试写出对应的 MIPS 汇编代码。

（2）假设处理器为标准的单发射 5 级流水线（IF/ID/EX/MEM/WB）结构，功能部件足够，Load、Store 操作和整数操作都花费 1 个时钟周期，浮点加法操作为 3 个周期，浮点乘法操作为 4 个周期。给出第一个循环所有指令的流水线时空图。

4. 假定某 RISC 处理器为标准的单发射 5 级流水线（IF/ID/EX/MEM/WB）结构。下面的代码在该处理器中执行。

```
Loop:  LD      R1, 0(R2)      ;从地址 0+R2 处读入 R1
       DADDI   R1, R1, #4     ;R1=R1+4
       SD      0(R2), R1      ;将 R1 存入地址 0+R2 处
       DADDI   R2, R2, #4     ;R2=R2+4
       DSUB    R4, R3, R2     ;R4=R3-R2
       BNEZ    R4, Loop       ; R4 不等于 0 时跳转到 Loop
       NOP
```

已知 R3 的初值为 R2＋400。

（1）不使用旁路硬件，但在同一个周期内寄存器的读和写能进行旁路，分支预测采用 not taken 策略，如果猜测错误则在转移指令执行（EX）后刷新（flushing）流水线上的错误指令，计算执行一个循环需要多少个时钟周期。

（2）使用旁路硬件，采用 taken 策略进行分支预测，如果猜测错误则在转移指令执行（EX）后刷新（flushing）流水线上的错误指令，计算执行一个循环需要多少个时钟周期。

（3）使用旁路硬件，分支指令具有单拍的分支延迟，可以重排指令序列并对包括延迟槽在内的指令序列进行调度，计算执行一个循环需要的时钟周期数。

verilog 题：ALU

```
module alu_module
(
input [ 3:0] aluop,
input [31:0] in1,
input [31:0] in2,
output [31:0] out
);
wire slt_res;
assign slt_res = (in1[31] && !in2[31]) & 1'b1 |
(in1[31] && in2[31] ) & (in1<in2) |
(!in1[31] &&!in2[31]) & (in1<in2) |
(!in1[31] && in2[31]) & 1'b0;
assign out =
{32{aluop==4'b0000}} & (in1+in2) |
{32{aluop==4'b0001}} & (in1-in2) | {32{aluop==4'b0010}} & {31'b0,
slt_res} |
{32{aluop==4'b0011}} & {31'b0, in1<in2} |
{32{aluop==4'b0100}} & (in1&in2) |
{32{aluop==4'b0101}} & (in1|in2) |
{32{aluop==4'b0110}} & (in1^in2) |
{32{aluop==4'b0111}} & ~(in1|in2) |
{32{aluop==4'b1000}} & (in1<<in2) |
{32{aluop==4'b1010}} & (in1>>in2) |
{32{aluop==4'b1011}} & ($signed(in1)>>>in2); //verilog2001 中算术
右移的新表达方式
endmodule
```

第六章：5、6

5. 以下这个循环是高斯消元法中的核心操作，称为 DAXPY 循环（双精度的 a 乘以 X 再加上 Y），以下代码实现了对长度为 100 的向量进行 DAXPY 操作：Y＝a＊X＋Y。

```
bar:
      LDC1      F2, 0(R1)        ;取数 X(i)
      MUL.D     F4, F2, F0       ;乘法操作 a*X(i)
      LDC1      F6, 0(R2)        ;取数 Y(i)
      ADD.D     F6, F4, F6       ;加法操作 a*X(i)+Y(i)
      SDC1      F6, 0(R2)        ;存数 Y(i)
      DADDIU    R1, R1, #8       ;X 的下标加 1
      DADDIU    R2, R2, #8       ;Y 的下标加 1
      DSGTUI    R3, R1, #800     ;测试循环是否结束
      BEQZ      R3, bar          ;如果循环没有结束，转移到 bar
      NOP
```

在单发射静态流水线上，假定浮点流水线的延迟如附表 6.1 所示（延迟为 N 表示第 T 拍操作数准备好开始运算，第 $T+N-1$ 拍可以写回结果），分支指令在译码阶段（ID）计算结果，采用了分支延迟槽技术，整数操作在一拍之内发射和完成，并且结果是完全旁路的（fully bypassed）。

附表 6.1　浮点流水线的延迟

产生结果的指令	使用结果的指令	延迟（时钟周期）
FP ALU op	Another FP ALU op	4
FP ALU op	Store double	3
Load double	FP ALU op	2
Load double	Store double	1

（1）把这个循环展开足够的次数，要求消除所有停顿周期和循环开销指令。循环将会被展开多少次？写出调度后的代码，每产生一个结果需要多少执行时间？

（2）写出 DAXPY 循环在软件流水后的代码，可以省略软件流水的装入代码和排空代码，每产生一个结果需要多少执行时间？

5. 解：但尽量按最少展开次数来做题。

a)　展开两次循环即可消除相关带来的阻塞影响。

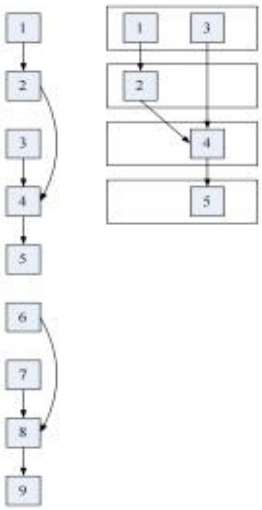
```
bar:
L.D F2, 0(R1) ;
L.D F3, 8(R1) ;
L.D F6, 0(R2) ;
MUL.D F4, F2, F0 ;
MUL.D F5, F3, F0 ;
L.D F7, 8(R2) ;
DADDUI R1, R1, #16 ;
ADD.D F6, F4, F6 ;
ADD.D F7, F4, F7 ;
DADDUI R2, R2, #16 ;
S.D -16(R2), F6 ;
```

DSGTUI R3, R1, #800 ;
BEQZ R3, bar ;
S.D -8(R2), F7 ;

14 拍两个，也就是 7 拍 1 个。软流水的全代码参见下面表格。

b） 首先分析原有循环中数据流图，将运算部分的流图进行流水切分。

```
1    L.D       F2, 0(R1)
2    MUL.D     F4, F2, F0
3    L.D       F6, 0(R2)
4    ADD.D     F6, F4, F6
5    S.D       0(R2), F6
6    DADDUI    R1, R1, #8
7    DADDUI    R2, R2, #8
8    DSGTUI    R3, R1, #800
9    BEQZ      R3, bar
```



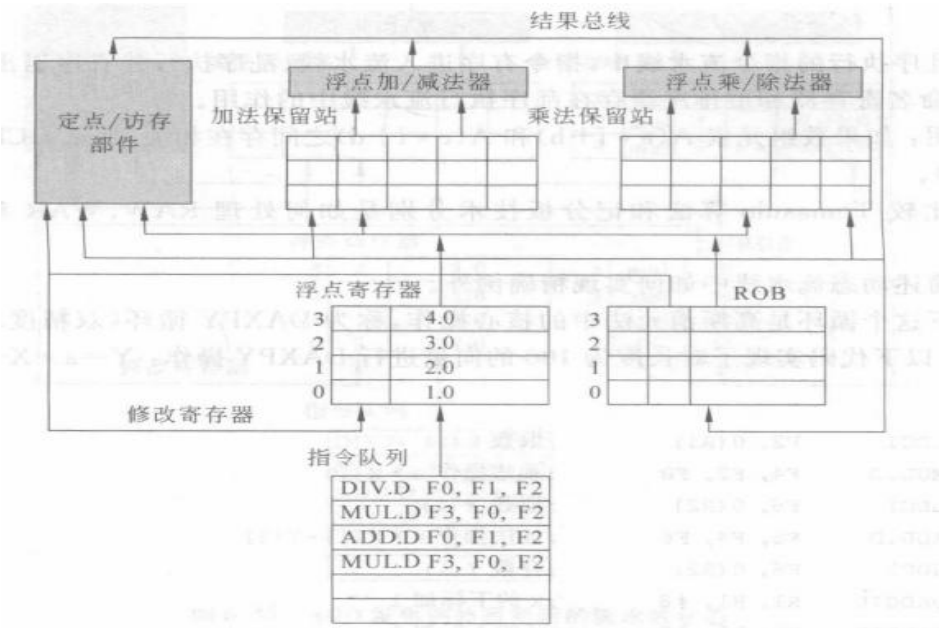
软件流水后主题循环代码如下：

```
S.D       -24(R2), F6       ;第 i-3 次循环的 5
ADD.D     F6, F4, F8        ;第 i-2 次循环的 4
MUL.D     F4, F2, F0        ;第 i-1 次循环的 2
L.D       F2, 0(R1)         ;
L.D       F8, -8(R2)        ;
DADDUI    R1, R1, #8        ;
DSGTUI    R3, R1, #800      ;
BEQZ      R3, bar           ;
DADDUI    R2, R2, #8        ;
```

9 拍处理 1 个元素

注： 假如 SD 的偏移为 A，LD(R1)的偏移为 B，LD(R2)的偏移为 C，循环退出条件为 D，有 A+16 = C ， D+B=800，一个例子是（-24，0，-8， 800）或者（-16，0， 0， 800）

全部代码：



附图 6.1　支持精确例外处理的浮点流水线

指令执行的状态表：(数字表示该操作在第几个时钟周期发生)

指令	发射	执行	写回	提交
DIV F0, F1, F2	1	2	4	5
MUL F3, F0, F2	2	4	6	7
ADD F0, F1, F2	3	4	5	8
MUL F3, F0, F2	4	5	7	9

Cycle 1

3		4.0		3			
2		3.0		2			
1		2.0		1			
0	0	1.0		0	div	F0	

resbus

Cycle 2

3	1	4.0		3			
2		3.0		2			
1		2.0		1	mul	F3	
0	0	1.0		0	div	F0	

resbus

Cycle 3

3	1	4.0		3			
2		3.0		2	add	F0	
1		2.0		1	mul	F3	
0	2	1.0		0	div	F0	

resbus

Cycle 4

3	3	4.0		3	mul	F3	
2		3.0		2	add	F0	
1		2.0		1	mul	F3	

resbus
(0, 0.67)

0	2	1.0		0	div	F0	0.67

Cycle 5

3	3	4.0		3	mul	F3		resbus
2		3.0		2	add	F0	5.0	(2, 5.0)
1		2.0		1	mul	F3		
0	2	0.67		0				

Cycle 6

3	3	4.0		3	mul	F3		resbus
2		3.0		2	add	F0	5.0	(1, 2.0)
1		2.0		1	mul	F3	2.0	
0	2	0.67		0				

第七章：4、5

4. 假设流水线延迟如附表 7.1 所示，分支延迟为 1 个周期，没有延迟槽。

附表 7.1 流水线延迟

产生结果的指令	使用结果的指令	延迟(时钟周期)
FP ALU op	Another FP ALU op	4
FP ALU op	Store double	3
Load double	FP ALU op	2
Load double	Store double	1

下面循环计算 Y[i]＝a＊X[i]＋Y[i]高斯消元法中的关键一步。

```
L:  LDC1    F4,0(R2)     ;读 Y[i]
    LDC1    F0, 0(R1)    ;读 X[i]
    MUL.D   F0,F0,F2     ;求 a＊X[i]
    ADD.D   F0,F0,F4     ;求 a＊X[i]+Y[i]
    SDC1    F0,0(R2)     ;保存 Y[i]
    DADDIU  R2,R2,-8
    DADDIU  R1,R1,-8
    BNEZ    R1,L
    NOP
```

(1) 假设目标机器的流水线是单发射的，将次循环展开足够的次数，使得代码执行没有不必要的延迟，写出调度后的代码并计算一个元素的执行时间。

(2) 假设目标机器的流水线是双发射的，将次循环展开足够的次数，使得代码执行没有不必要的延迟，写出调度后的代码并计算一个元素的执行时间。

(3) 自己写一段与题中类似的 C 代码，用 gcc 的不同优化编译选项编译后，查看汇编代码，对不同优化选项进行比较，描述 gcc 做的优化。

L:	L.D	F0, 0(R1)	; load X[i]
	L.D	F6, -8(R1)	; load X[i-1]
	MUL.D	F0, F0, F2	; a*X[i]
	MUL.D	F6, F6, F2	; a*X[i-1]
	L.D	F4, 0(R2)	; load Y[i]
	L.D	F8, -8(R2)	; load Y[i-1]
	ADD.D	F0, F0, F4	; a*X[i] + Y[i]
	ADD.D	F6, F6, F8	; a*X[i-1]+ Y[i-1]
	DSUBUI	R2, R2, 16	
	DSUBUI	R1, R1, 16	
	S.D	F0, 16(R2)	; store Y[i]
	S.D	F6, 8(R2)	; store Y[i-1]
	BNEZ	R1, L	

计算一个元素需 14/2=7 拍。

正常展开即可

(2) 假设双发射流水线中有彼此独立的一条定点流水线和一个浮点流水线。

	定点指令线		浮点指令线	
L:	L.D	F0, 0(R1)		
	L.D	F6, -8(R1)		
	L.D	F10, -16(R1)	MUL.D	F0, F0, F2
	L.D	F14, -24(R1)	MUL.D	F6, F6, F2
	L.D	F4, 0(R2)	MUL.D	F10, F10, F2
	L.D	F8, -8(R2)	MUL.D	F14, F14, F2
	L.D	F12, -16(R2)	ADD.D	F0, F0, F4
	L.D	F16, -24(R2)	ADD.D	F6, F6, F8
	DSUBUI	R2, R2, 32	ADD.D	F10, F10, F12
	DSUBUI	R1, R1, 32	ADD.D	F14, F14, F16
	S.D	F0, 32(R2)		
	S.D	F6, 24(R2)		
	S.D	F10, 16(R2)		
	S.D	F14, 8(R2)		

(3) 实验题。

```
int main() {
int i;
for (i=100; i>=0; i--)
Y[i] = a*X[i] + Y[i];return 0;}
```

第八章：1、2、3、6

PHT 表的位数为低 8 位，BHT 表每项 9 位，请画出 SAs 转移猜测的结构图，说明其基本原理，并计算该结构使用的存储单元位数。

附表 8.1　转移猜测的不同组合

	Global PHT	Per address PHT	Per set PHT
Global BHR	GAg	Gap	GAs
Per-address BHR	PAg	PAp	PAs
Per-set BHR	SAg	SAp	SAs

1. 解：BHT $2^6*9=576b$

PHT $2^8*2^9*2=262144b$

共 262720b

基本原理：BHT 根据地址低 6 位选出一个 9 位向量，和地址低 8 位一起到 PHT 中选取 2 位饱和计数。图略

2. 考虑下面一段 MIPS 代码，假设开始时 R1 寄存器存放值 a。

```
        BNEZ    R1,L1       ;分支 b1(a!=0)
        ADDIU   R1,R0,2     ;a==0,a 的值变为 2
L1:     ADDIU   R2,R1,-2
        BNEZ    R2,L2       ;分支 b2(a!=2)
        ⋮
L2:
```

假定采用(1,1)相关分支预测器(根据 1 位全局历史去选择 2 项的局部历史表 PHT，局部历史表为 1 位的计数器)，初始预测位 NT/NT，且 a 值以 0,1,0,1 的规律变化，画出转移预测执行情况表，并统计预测错误的数目。

注：下表中的 NT/NT，表示两张表中属于某分支的项分别为 NT 和 NT。前者为历史为 NT 的表中的项，后者为历史为 T 的表中的项。加粗的是根据当前历史选中的表项。

a	b1 prediction	b1 action	New b1 prediction	b2 prediction	b2 action	New b2 prediction
0	**NT**/NT	NT	**NT**/NT	**NT**/NT	NT	**NT**/NT
1	**NT**/NT	T(miss)	T/NT	NT/**NT**	T(miss)	NT/T
0	T/**NT**	NT	T/NT	**NT**/T	NT	NT/T
1	**T**/NT	T	T/NT	NT/**T**	T	NT/T

因此，总共有 2 次预测错误出现。

3. 一个流水线处理器为条件分支配置了一个分支目标缓冲器 BTB。假设预测错误的损失为 5 个时钟周期，BTB 不命中的开销是 3 个时钟周期，并假设 BTB 命中率为 90%，命中时预测精度为 90%，分支频率为 20%，没有分支的基本 CPI 为 1，请计算：

① 该程序执行的 CPI；

② 另外一个处理器没有转移猜测部件，每个分支有固定 2 个时钟周期的性能损失，与①中的处理器比较执行速度。

3. 解：未来出题时，会明确“额外开销”

a) 1 + 20%*((90%*10%*5)+(10%*3)) = 1.15

b) 1 + 20%*2=1.4

1.4/1.15=1.22，慢 22%。

附表 8.2 流水线延迟

产生结果的指令	使用结果的指令	延迟周期数
FP ALU op	Another FP ALU op	4
FP ALU op	Store double	3
Load double	FP ALU op	2
Load double	Store double	1

(1) 下面程序段实现对数组元素增值，R1 和 R2 的初始值满足 R1＝8n＋R2，n 为循环次数。

```
L1:   LDC1    F0,0(R1)
      ADD.D   F2,F0,F1
      SDC1    F2,0(R1)
      ADDIU   R1,R1,-8
      BNE     R1,R2,L1
      NOP
```

写出完整的软件流水循环代码，包括装入代码和排空代码，并且计算软件流水循环完成所有操作需要的总时钟周期数表达式。

(1)

```
//装入代码
LDC1
F0,0(R1)
ADD.D
F2,F0,F1
LDC1
F0,-8(R1)
ADDIU
R1,R1,-24
```

```
//主体循环
L1:
SDC1
F2,24(R1) //Loop i-2
ADD.D
F2,F0,F1 //Loop i-1
LDC1
F0,8(R1) //Loop iBNE
R1,R2,L1
ADDIU
R1,R1,-8
```

```
//排空代码
SDC1
F2,24(R1)
ADD.D
F2,F0,F1
SDC1
F2,16(R1)
```

总的执行周期数：(4+1) + (5*(n-2)) + (3+2) = 5n

(2) 软流水无法降低分支频率，循环展开可以； 软流水代码量增加少，循环展开会带来代码膨胀

第九章；2、3、5，画简单的加法器

2. 假设每个“非门”“与非门”“或非门”的扇入不超过 4 个且每个门的延迟为 T，请给出使用如下不同算法的 16 位加法器的延迟。

2. 解： 第二问先行进位记得算 pg 和全加器的延迟。

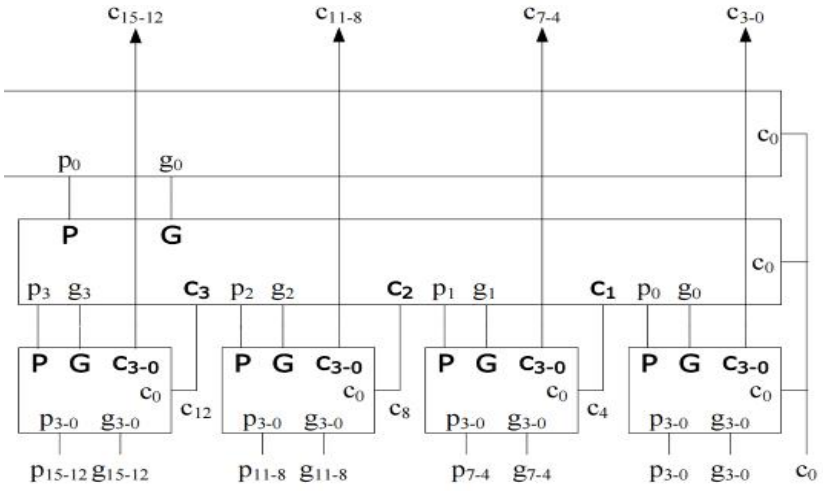
a). 串行进位加法器；
每一级进位传递的延迟为 2T，因此生成 c16 需要 32T；
每一级产生结果的延迟为 3T，因此生成 s15 需要（30T+3T）=33T

b). 先行进位加法器
参照课本图 9.6 的方式搭建：组内并行、组间并行，4 位一组。
延迟共 2T（产生 p、g）+2T（产生每组 P、G）+2T（产生组间进位）+2T（产生组内进位）+ 3T（全加器逻辑）= 11T

c). 先行进位加法器快的原因是它能更快地生成第 i 位的 c 而不需要依赖于第 i-1 位的 c。 **概念题**

3. 假设每个“非门”“与非门”“或非门”的扇入不超过 4 个且每个门的延迟为 T，请给出使用如下不同算法的把 4 个 16 位数相加的延迟。
① 使用多个先行进位加法器；
② 使用加法树及先行进位加法器。

a). 使用多个加法器；采用先行进位加法器两两相加，需要 2*11T=22T 延迟
b). 使用加法树及加法器。
使用加法树把四个数相加变成两个数相加，需要 2 级全加器延迟（6T），然后再
使用先行进位加法器（11T）得到最后结果，因此共 6T+11T=17T 延迟。



Verilog 题：16 位先行进位加法器

```
module add16(a, b, cin, out, cout);
    input    [15:0]   a;
    input    [15:0]   b;
    input             cin;
    output   [15:0]   out;
    output            cout;

    wire [15:0] p = a|b;
    wire [15:0] g = a&b;
    wire [3:0]    P, G;
    wire [15:0]   c;

    assign c[0] = cin;

    C4 C0_3(.p(p[3:0]),.g(g[3:0]),.cin(c[0]),.P(P[0]),.G(G[0]),.cout(c[3:1]));
    C4 C4_7(.p(p[7:4]),.g(g[7:4]),.cin(c[4]),.P(P[1]),.G(G[1]),.cout(c[7:5]));
    C4 C8_11(.p(p[11:8]),.g(g[11:8]),.cin(c[8]),.P(P[2]),.G(G[2]),.cout(c[11:9]));
    C4 C12_15(.p(p[15:12]),.g(g[15:12]),.cin(c[12]),.P(P[3]),.G(G[3]),.cout(c[15:13]));
    C4 C_INTER(.p(P),.G(G),.cin(c[0]),.P(),.G(),.cout({c[12],c[8],c[4]}));

    assign cout = (a[15]&b[15]) | (a[15]&c[15]) | (b[15]&c[15]);
    assign out = (~a&~b&c)|(~a&b&~c)|(a&~b&~c)|(a&b&c);
endmodule
```

```
module C4(p,g,cin,P,G,cout)
    input    [3:0]    p, g;
    input             cin;
    output            P,G;
    output   [2:0]    cout;
```

```
    assign P=&p;
    assign G=g[3]|(p[3]&g[2])|(p[3]&p[2]&g[1])|(p[3]&p[2]&p[1]&g[0]);

    assign cout[0]=g[0]|(p[0]&cin);
    assign cout[1]=g[1]|(p[1]&g[0])|(p[1]&p[0]&cin);
    assign cout[2]=g[2]|(p[2]&g[1])|(p[2]&p[1]&g[0])|(p[2]&p[1]&p[0]&cin);
endmodule
```

第十章：1、2、3

1. 一个处理器的页大小为 4KB，一级数据 cache 大小为 64KB，cache 块大小为 32B，指出在直接相联、二路组相联，以及 4 路组相联的情况下需要页着色（page coloring）的地址位数。

1、解： 参考课本内容即可

页大小为 4KB=2^{12}B，页内地址为[11:0]位。

cache 容量 64KB=2^{16}B，地址范围为[15:0]；cache 块大小为 32B=2^{5}B，地址范围为[4:0]。

1）直接相联：

cache 索引位数为地址的[15:5]，需要页着色的是地址[15:12]，共 4 位；

2）二路组相联：

cache 索引位数为地址的[14:5]，需要页着色的是地址[14:12]，共 3 位；

3）四路组相联：

cache 索引位数为地址的[13:5]，需要页着色的是地址[13:12]，共 2 位。

2. 根据以下数据，计算采用不同相联度的二级 cache 时，一级 cache 不命中的损耗。采用直接相联时，L2 命中时间为 4 个时钟周期，采用 2 路组相联和 4 路组相联时，L2 的命中时间将分别增加 1 个时钟周期和 2 个时钟周期，直接相联、2 路组相联、4 路组相联的局部失效率分别为 25%、20%、15%，L2 的不命中损耗（从 L2 失效到数据返回 L2）是 60 个时钟周期。

2、解： 代价、损耗等词语，一看就是描述额外开销

$MissPenalty_{L1}=HitTime_{L2}+MissPenalty_{L2}\times MissRate_{L2}$

1）直接相联：

$MissPenalty_{L1}=4+60\times 25\%=19$ 个时钟周期

2）2 路组相联：

$MissPenalty_{L1}=5+60\times 20\%=17$ 个时钟周期

3）4 路组相联：

$MissPenalty_{L1}=6+60\times 15\%=15$ 个时钟周期

3. 通过分开指令 cache 和数据 cache，能够消除因指令和数据冲突而引起的 cache 缺失。现有一个由大小都为 32KB 的指令 cache 和数据 cache 组成的系统和另一个大小为 64KB 的一体 cache 组成的系统相比较。假设在所有的存储器访问中数据访问占 26%，取指令占 74%。32KB 指令 cache、数据 cache 和 64KB 的一体 cache 每千条指令缺失率分别为 1.5、38 和 40。cache 命中时需要 1 个时钟周期，缺失代价为 100 个时钟周期，在一体 cache 中，Load 和 Store 命中需要一个额外的时钟周期。

（1）计算并比较哪一个组织方式有更低的缺失率。

（2）求出每种组织方式下的平均存储器访问时间。

3、解： 哪个缺失率高，可以直接按 1.5+38 和 40 进行比较；但是如果算各个 cache 的访问缺失率，则需要按下面的方式算：

1）MissRate = CacheMissOps/MemOps

每 1000 条指令中 load/store 指令的条数是 1000×(26%/74%)=351

32KB 指令 cache MissRate 为 0.0015；

32KB 数据 cache MissRate 为 38/351=0.1083

指令 cache 和数据 cache 各 32KB 的 MissRate = (1.5 + 38)/(1000+351) = 0.0292

64KB 一体 cache 的 MissRate = 40/(1000+351) = 0.0296

所以指令 cache 和数据 cache 各 32KB 组织方式缺失率更低

2）AMAT 的定义存在歧义，参考答案提供其中一种算法：实际执行时间 除以 总访问次数

指令 cache 和数据 cache 各 32KB 的 AMAT = (1000+1.5×100+38×100)/(1000+351)=3.66

64KB 一体 cache 的 AMAT = (1000+351+40×100)/(1000+351)=3.96

第十一章：1 必考，但不是原题，3、5

1. 在如下一段 C 语言程序的 for 循环中，

```
void cycle(double* a) {
  int i;
  double b[65536];
  for(i=0; i<3; i++) {
   memcpy(a, b, sizeof(b));
  }
}
```

程序memcpy函数执行过程中可能发生哪些例外，各多少次。

答： 题目中没有给TLB项数和替换算法，参考答案按项数足够计算，但各位同学需要掌握项数不足情况的计算公式。题目中没有考虑modify例外和a数组已在父函数进行过访问的情况，题干不是很严谨。

【说明：此题中试图问的是与页处理相关的例外（忽略实际系统中可能发生的其他内部外部中断），以MIPS体系结构为例，包括三类共四种例外：

(1) TLB refill 例外；

(2) TLB invalid Load例外 / TLB invalid Store例外；

(3) TLB modify 例外。

其中(2)和(3)两类例外仅与程序对于页的访问需求相关，因此回答次数时需要假设操作系统中页的大小；而(1)还与处理器中TLB的容量相关，因此还需要假设处理器中TLB的项数和TLB的替换策略】。

下面给出一种常用假设条件下的回答：

操作系统页大小为4KB，同时系统的物理内存足够大，即页表分配的物理内存在程序执行过程中不会被交换到swap区上。并且在后续的分析中忽略代码和局部变量i所在的页的影响。

a、b两数组大小均为65536×8=512KB。再假设a、b两数组的起始地址恰为一页的起始地址。

那么a、b两数组各自均需要512KB/4KB=128个页表项。

所以a、b的访问造成的TLB invalid 例外次数为128+128=256次；

假设TLB表项为128项，每项映射连续的偶数奇数两个页，采用LRU算法。那么第一次次循环中，a、b两数组每访问相邻偶数奇数两页的首地址时，均会触发一次TLB refill例外。后续各次循环TLB中均命中。那么共触发的TLB refill例外次数为128/2 + 128/2 = 128次。

3. 假定在某一个 CPU 的 Cache 中需要 64 位虚拟地址，8 位的进程标识，而其支持的物理内存最多有 64GB。请问，使用虚拟地址索引比使用物理地址作为索引的 Tag 大多少？这个值是否随着 Cache 块大小的变化而发生改变？

答：参考课本内容即可得

1、由于虚拟地址有 64 位，进程标识为 8 位；而物理地址是 64GB，即需要 36 位物理地址。这样，使用虚拟地址比使用物理地址总共增加的位数为 8+（64−36）=8+28=36 位。
2、改变块的大小，亦或改变 cache 的其它参数，cache 占据的地址的低位的长度对于虚地址索引和物理地址索引都是一样的，所以对于 cache 的 tag 而言，两中索引方式相差位数保持不变。

Verilog 题：TLB 地址查找

5. 已知一台计算机的虚地址为 48 位，物理地址为 40 位，页大小为 16KB，TLB 为 64 项全相联，TLB 的每项包括一个虚页号 vpn，一个物理页号 pfn，以及一个有效位 valid，请根据如下模块接口写出一个 TLB 的地址查找部分的 Verilog 代码。

```
module tlb_cam(vpn_in,pfn_out,hit,…);
```

```
module tlb_cam(vpn_in, pfn_out, hit, valid_out);

input [33:0] vpn_in;
output [25:0] pfn_out;
output hit;
output valid_out;

reg[60:0] cam_content [63:0];//[60:60] valid [59:34] pfn [33:0] vpn
wire [63:0] entry_hit;

assign entry_hit[0] = (vpn_in==cam_content[0][33:0]);
assign entry_hit[1] = (vpn_in==cam_content[1][33:0]);
……
assign entry_hit[63] = (vpn_in==cam_content[63][33:0]);

assign hit = |entry_hit;

assign pfn_out =  {26{entry_hit[ 0]}} & cam_content[0][59:34] |
                  {26{entry_hit[ 1]}} & cam_content[1][59:34] |
……
                  {26{entry_hit[63]}} & cam_content[63][59:34];

assign valid_out = entry_hit[ 0] && cam_content[ 0][60] ||
                   entry_hit[ 1] && cam_content[ 1][60] ||
                   …
                   entry_hit[63] && cam_content[63][60];
endmodule
```

第十二章：6、7、8

6. 假设在一个双 CPU 多处理器系统中，两个 CPU 用单总线连接，并且采用监听一致性协议（MSI），cache 的初始状态均为无效，然后两个 CPU 对内存中同一数据块进行如下操作：CPU A 读、CPU A 写、CPU B 写、CPU A 读，写出每次访问后两个 CPU 各自的 cache 的状态变化。

6、答： **请掌握 MSI 一致性协议。**

事件	A 状态	B 状态
初始	I	I
CPU A 读	S	I
CPU A 写	M	I
CPU B 写	I	M
CPU A 读	S	S

8. 有以下两个并行执行的进程，在顺序一致性和弱一致性下，它各有几种正确的执行顺序，给出执行次序和最后的正确结果（假设 a 和 b 的初始值为 0）。

```
P1              P2
a=1;            b=1;
print b;        print a;
```

8、答： **概念题**

（1）顺序一致性：

i. a=1 先于 print a 执行，b=1 先于 print b 执行。最终结果：a=1, b=1

ii. a=1 先于 print a 执行，print b 先于 b=1 执行。最终结果：a=1, b=0

iii. print a 先于 a=1 执行，b=1 先于 print b 执行。最终结果：a=0, b=1

（2）弱一致性：

除顺序一致性中的三种执行序外，还可以有第 iv 中执行序：

print a 先于 a=1 执行，print b 先于 b=1 执行。最终结果：a=0, b=0

7. 在下面程序段中，A、B、C、D 为进程 P1、P2、P3 的共享变量，且初始值均为 0。该程序的正确运行解结果是 D=2000。

(1) 在一个用可伸缩网络连接、采用基于目录的 cache 一致性协议的分布式共享存储系统中运行上述程序得到结果是 D=0，请解释产生上述结果的原因。

(2) 在实现顺序一致性的分布式共享存储系统中，对上述程序施加什么限制可以保证执行结果的正确性。

(3) 在实现弱一致性的分布式存储系统中，采用 barrier 进行同步，请问上述程序如何插入 barrier 操作才能保证执行的正确性？

```
P1              P2                  P3
A=2000          while(B!=1) {;}     while(C!=1) {;}
B=1             C=1;                D=A
```

7、答： **概念题**

(1) P1 对 A 的写被网络阻塞，一直没有传播到 P2 和 P3。

(2) 无须施加限制。

(3)

```
P1              P2                  P3
                barrier1            barrier1
                                    barrier2
A=2000          while (B!=1) {;}    while (C!=1) {;}
B=1             C=1;                D=A
```

```
barrier1        barrier2
barrier2
```