第二章参考答案 计算机系统结构基础

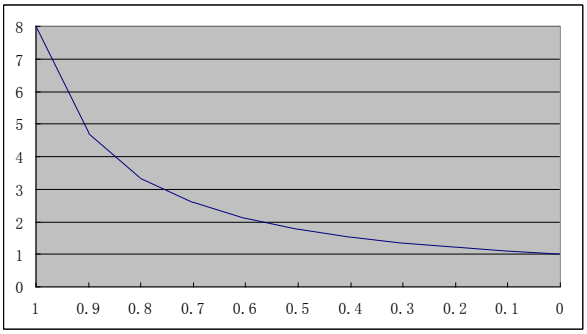
1. 解：A 为 10MIPS，B 为 20MIPS，C 为 40MIPS。

三台机器实际性能相同。

题目问的是运行程序 P 时的性能

2. 解：加速比 y 与向量化比例 x 之间的关系是：y=1/((1-x)+x/8)=1/(1-7x/8)……(A)

(1) **假设原运行时间为 1，新总运行时间为(1-x) + x/8，得加速比 y**



(2) 在式(A)中令 y=2，可解得 x=4/7≈57.14%。

此时向量模式运行时间占总时间比例是((4/7)/8)/(3/7+((4/7)/8))=1/7=14.29%

(3) 硬件方法，整体加速比为 1/((1-0.7)+0.7/16)=2.91

软件方法，设相同加速比下向量化比例为 x，即 1/((1-x)+x/8)=2.91，x=0.75

所以推荐软件方法。

注意不要用错公式。

3. 解：

(1) $MIPS_{EMUL}=(I+F\times Y)/(W\times 10^6)$ ；$MIPS_{FPU}=(I+F)/(B\times 10^6)$ **实际执行总指令除以总时间**

(2) $120=(I+8\times 10^6\times 50)/(4\times 10^6)$ => $I=80\times 10^6$

(3) $80=(80\times 10^6+8\times 10^6)/(B\times 10^6)$ => $B=1.1$

(4) $MFLOPS=F/((B-((W*I)/(I+F*Y)))\times 10^6)\approx 18.46$ **按运行该程序时浮点指令数除以浮点部分所占时间（总时间减去定点部分时间）来计算**

(5) 决策正确，因为执行时间缩短了，这才是关键标准。

注意不要用错公式。

4. 解：当取 c=4000 时：

(1) $y=12.29386-0.18295x+0.0015x^2$

(2) $y=342.47443-6.36386x+0.02727x^2$

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

注意不要用错公式。

6. 解：

a) 先证明 $N=2^k$ 时，正数 $(a_1+a_2+\cdots+a_N)/N \geq \sqrt[N]{a_1a_2\cdots a_N}$。对 k 进行数学归纳法即可。

当 $2^{k-1}<N<2^k$ 时，令 $A=(a_1+a_2+\cdots+a_N)/N$，则 $A=(a_1+a_2+\cdots+a_N+\cdots)/N=(a_1+a_2+\cdots+a_N+A^{k}-N)/2^k$。若 $\sqrt[N]{a_1a_2\cdots a_N}>A$，$A\geq\sqrt[2^k]{a_1a_2\cdots a_N A^{(2^k-N)}}>\sqrt[2^k]{A^N A^{(2^k-N)}}=A$，矛盾。因此当当 $2^{k-1}<N<2^k$ 时，$(a_1+a_2+\cdots+a_N)/N \geq \sqrt[N]{a_1a_2\cdots a_N}$。

b) 证：假设参考机的程序分值为 $Z=\{Z_0, Z_1,\cdots,Z_{n-1}\}$，其中 n 为 SPEC CPU2000 中的程序个数；

而 A 机器的程序分值为 $X=\{x_0,x_1,\cdots,x_{n-1}\}$

B 机器的程序分值为 $Y=\{y_0,y_1,\cdots,y_{n-1}\}$

则有：

A 机器的性能为：$\sqrt[n]{\frac{x_0*x_1*\cdots*x_{n-1}}{Z_0*Z_1*\cdots*Z_n}}$，B 机器的性能为：$\sqrt[n]{\frac{y_0*y_1*\cdots*y_{n-1}}{Z_0*Z_1*\cdots*Z_n}}$

从而，A 与 B 机器的性能比为：

$$\frac{\sqrt[n]{\frac{x_0*x_1*\cdots*x_{n-1}}{Z_0*Z_1*\cdots*Z_n}}}{\sqrt[n]{\frac{y_0*y_1*\cdots*y_{n-1}}{Z_0*Z_1*\cdots*Z_n}}}=\sqrt[n]{\frac{x_0*x_1*\cdots*x_{n-1}}{y_0*y_1*\cdots*y_{n-1}}}$$

可见，其结果与参考样机无关。故得证。

7. 解：

AMD 4 核 Barcelona，2.8G，3 发射每个核 1 个 128 位浮点向量功能部件和 1 个 128 位浮点加法向量部件，峰值性能 4*4*2.8=44.8GFlops。 2 路 L1I 64KB；2 路 L1D 64KB 3 latency；16 路 L2 512KB；32 路 2MB 共享 L3，内存带宽 21.34GB/s

Intel 4 核 Nehalem（i7），2.5G-3G，4 发射每个核 1 个 128 位浮点向量功能部件和 1 个 128 位浮点加法向量部件，峰值性能 4*4*3=48GFlops。4 路 L1I 32KB；4 路 L1D 32KB 4 latency；8 路 256KB L2 12 latency；16 路 8MB L3 30-40 latency；内存带宽 31.92GB/s

8. （1） 言之有理即可。

第三章参考答案 二进制与逻辑电路

1. 解：

(1) $[-(2^{63}-1),\ 2^{63}-1]$和$[-2^{63},\ 2^{63}-1]$

(2) -2^{31} **常识题**

2. 解： **常识题**

(1)

0x7ff0000=0,0000 1111,111 1111 0000 0000 0000 0000=$(1.1111111)_2*2^{(15-127)}$=3.836813561083946426009956057493 4e-34

0xbe400000=1,0111 1100,100 0000 0000 0000 0000=$-(1.1)_2*2^{(124-127)}$=-0.1875

0xff800000=1,1111 1111,000 0000 0000 0000 0000=$-\infty$

(2)

0x4035000000000000=0,10000000011,0101000000000000000000000000000000000000 000000000000=$(1.0101)_2*2^{(1027-1023)}$=21

0x8008000000000000 =

1,00000000000,0000000000000000000000000000000000000000 000000000000=$-(0.1)_2*2^{-1022}=-2^{-1023}$

(3)

-100.0=$-(1.100100)_2*2^6$=0b1 10000101 10010000000000000000000=0xc2c80000

0.25=$(1.0)*2^{-2}$=0b0 01111101 00000000000000000000000=0x3e800000;

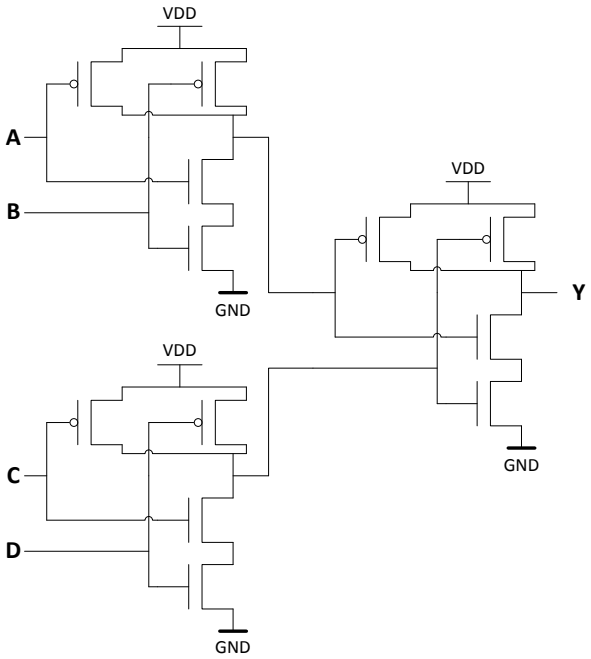
(4)

1024.0=$(1.0)*2^{10}$=0x4090000000000000

0.25=(1.0)*2^{-2}=0x3fd0000000000000

3. 解：

A&B|C&D=～(～(A&B)&～(C&D))，两级与非门的逻辑



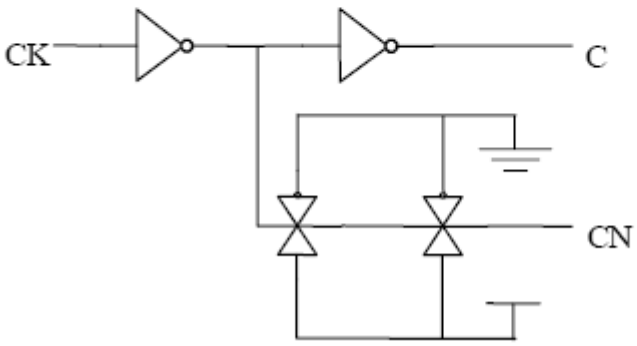
另一种解法，是课本上的~(A&B|C&D)后面再加一个反相器。
常识：N 管要接地，P 管要接电，不能反过来。所以一级逻辑画不出的。

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。

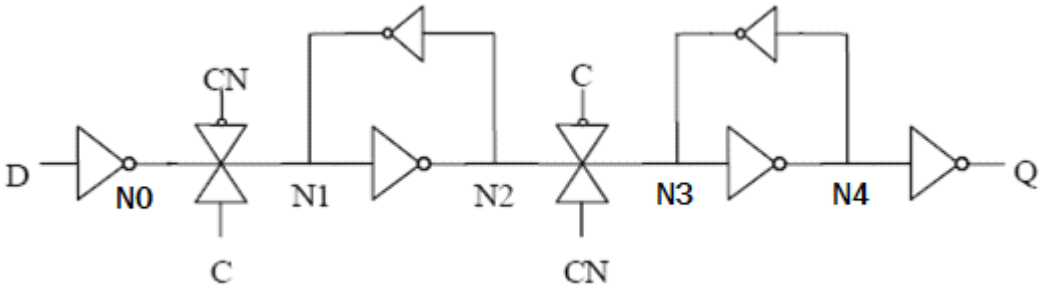
5. 解：



时钟的延迟：

CK→C： 1+1=2ns

CK→CN：1+0.5+0.5=2ns



通过分析电路行为可知这是一个下降沿触发的 D 触发器。

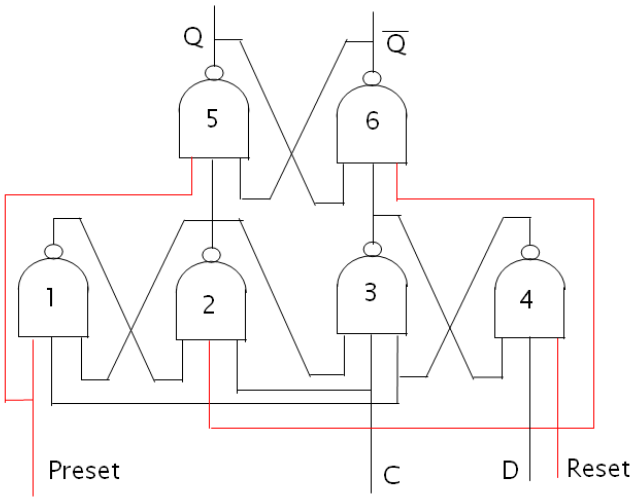
建立时间指的是在时钟信号到达 CK 端之前，将触发器内部 N1 及 N2 状态改变并稳定为与 D 端数据相符所需的时间。这样，D 端数据必须通过 D → N0 → N1 → N2 才能真正改变触发器内部状态，但即使如此，由于 N1 和 N2 间反相器环驱动能力不能确定，为保守起见，还需要加上 N2 → N1 时间。此外考虑到接口处 CK 端时钟信号到 C 和 CN 的传播时延，如果 C 和 CN 的传播时延不一，可能导致传输门输出弱 1 或弱 0 情况，仍从保守情况出发取两者的较小值，另外还要算上传输门控制端栅到漏（源）的延迟。这样，该触发器建立时间 $T_{setup}=T_{D-N0-N1-N2-N1}-(\min(T_{CK-C},T_{CK-CN})+T_{tran})=(1+0.5+1+1)-(\min(2,2)+0.75)=0.75$ ns

保持时间指的是在时钟信号到达 CK 端之后，D 端需要等待多长时间，使得即使其数据变化也不影响触发器内部状态。反过来想，那什么情况下 D 端数据变化可能会影响内部状态呢？只有当前级传输门在完全关断之前，D 端数据已经进入到 N1，进而才有可能对内部状态产生影响。所以只需保证在前级传输门关断时变化的 D 端数据不进入 N1 即可。此外也要考虑到时钟信号的传播延迟，仍从保守情况出发取两者较大值，加上传输门控制端栅到漏（源）的延迟。这样，$T_{hold}=(\max(T_{CK-C},T_{CK-CN})+T_{tran})-T_{D-N0-N1}=(\max(2,2)+0.75)-(1+0.5)=1.25$ns。

CK→Q 时间指的是时钟触发沿到来之后 Q 端输出新的触发器状态所需的时间。只有当后级传输门打开后，Q 端才有可能与触发器内部状态相符，也就是 C=1→0（CN=0→1）时钟下降沿时，这时候 N2 处的状态需要通过 N2→N3→N4→Q，此时由于后级传输门出于打开状态，N3-N4 处的反相器环一般不可能再破坏这个新状态。此外仍出于保守考虑时钟信号的传播延迟取较大值，并加上传输门控制端栅到漏(源)的延迟。这样，该触发器 CK→Q 时间 $T_{CK-Q}=(\max(T_{CK-C},T_{CK-CN})+T_{tran})$

$+T_{N2-N3-N4-Q}=(\max(2,2)+0.75)+(0.5+1+1)$ =5.25ns

6. 解：



先不考虑 preset 和 reset 信号的影响，即 preset=1 且 reset=1，分析如下：
1. 当 C 信号发生 1→0 的变化时，2 单元和 3 单元强制输出{1,1},5 单元和 6 单元的状态继续保持。
2. 当 C 信号发生 0→1 的变化时，
若 D 输入为 0，4 单元输出为 1,1 单元输出为 0，使得 2 单元和 3 单元输出分别为 1 和 0，进而 5 单元和 6 单元的 Q 和 QN 输出分别为 0 和 1；
若 D 输入为 1，4 单元输出 0，使得 1 单元输出为 1，2 单元和 3 单元输出分别为 0 和 1，进而 5 单元和 6 单元的 Q 和 QN 输出分别为 1 和 0。
当 C 信号继续维持在 1 时，由于 2 单元和 3 单元的状态组合只可能是{0,1}和{1,0}中的一种，若 2 单元输出为 0，无论 D 输入如何影响 4 单元输出，1 单元和 3 单元始终为 1,D 输入信号无法穿透进入下一级；若 3 单元输出为 0，4 单元输出时钟为 1，D 输入信号无法穿透 4 单元，因此数据不再变化。
得到第一步结论：当复位无效时（即 preset 与 reset 均为 1 时），该电路只在 C 信号发生 0→1 变化时接受 D 输入信号，因此是一个 D 触发器。

再考虑 preset 和 reset 信号：
当 preset=0 且 reset=1 时，5 单元的 Q 输出为 1，1 单元输出 1，使得 2 单元输出 0，控制 3 单元输出 1，进而 6 单元受到 5 单元输出 Q 信号影响，输出 QN 为 0；
当 preset=1 且 reset=0 时，6 单元的 QN 输出为 1，2 单元输出 1，4 单元输出 1，使得 3 单元输出 0，进而 5 单元受到 6 单元输出 QN 信号影响，输出 Q 为 0；
当 preset=0 且 reset=0 时，5 单元和 6 单元的 Q 和 QN 输出都是 1，不符合单元逻辑要求，因此应当避免。

第四章参考答案 指令系统结构

1. **从内存取指令，也算内存交换；所有指令、数据均以字节为单位，不存在半字节等。**

解：(1)

	堆栈型	累加器型	寄存器-存储器型	寄存器-寄存器型

汇编代码	Push B
	Push C
	Sub
	Pop A
	Push A
	Push C
	Sub
	Pop D
	Push D
	Push A
	Add
	Pop B
		Load C
		Neg
		Add B
		Store A
		Load C
		Neg
		Add A
		Store D
		Add A
		Store B
			Load R1,B
			Sub R1,C
			Store R1,A
			Sub R1,C
			Store R1,D
			Add R1,A
			Store R1,B
				Load R1,B
				Load R2,C
				Sub R3,R1,R2
				Store R3,A
				Sub R4,R3,R2
				Store R4,D
				Add R5,R4,R3
				Store R5,B

指令字节	30	26	28	29
内存交换字节	48	42	42	39
代码量衡量		√		
交换数据量衡量				√

(有的同学认为累加器型有 sub 指令，目前也算对。这种情况下，两个都是累加器型最优)

2. **常识题**

解：小尾端：

Address	0xxx000	0xxx001	0xxx010	0xxx011	0xxx100	0xxx101	0xxx110	0xxx111
0x	4E	4F	53	47	4E	4F	4F	4C
ASCII	N	O	S	G	N	O	O	L

大尾端：

Address	0xxx000	0xxx001	0xxx010	0xxx011	0xxx100	0xxx101	0xxx110	0xxx111
0x	4C	4F	4F	4E	47	53	4F	4E
ASCII	L	O	O	N	G	S	O	N

3. **注意理解题意，分别计算总指令数量和所需时间**

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

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 lw r1,0(r3) add r3, r2, r1 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。

5.

a） 条件转移指令的跳转范围。

16+2 位 256KB（+-128KB）

b） 直接跳转指令的跳转范围。

26+2 位 256MB （并非+-128MB）

6.

解：

小尾端的解答：

```
dli r2, 1005
lwr r1, 0x0(r2)
lwl r1, 0x3(r2)
dli r2, 2005
swr r1, 0x0(r2)
swl r1, 0x3(r2)
```

部分同学按大尾端做的，也算正确。

7.

解： **常识题**

```
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。

8. 解：

X86 的减法指令如下：

定点减法：

SUB AL,imm8 Subtract imm8 from AL

SUB AX,imm16 Subtract imm16 from AX

SUB EAX,imm32 Subtract imm32 from EAX

SUB RAX,imm32 Subtract sign-extend imm32 from RAX

SUB r/m8,imm8 Subtract imm8 from 8bit register or 8bit memory location

SUB r/m16,imm16 Subtract imm16 from 16bit register or 16bit memory location

SUB r/m32,imm32 Subtract imm32 from 32bit register or 32bit memory location

SUB r/m64,imm32 Subtract sign-extend imm32 from 64bit register or 64bit memory location

SUB r/m16,imm8 Subtract sign-extend imm8 from 16bit register or 16bit memory location

SUB r/m32,imm8 Subtract sign-extend imm8 from 32bit register or 32bit memory location

SUB r/m64,imm8 Subtract sign-extend imm8 from 64bit register or 64bit memory location

SUB r/m8,r8 Subtract 8bit register from 8bit register or 8bit memory location

SUB r/m16,r16 Subtract 16bit register from 16bit register or 16bit memory location

SUB r/m32,r32 Subtract 32bit register from 32bit register or 32bit memory location

SUB r/m64,r32 Subtract sign-extend 32bit register from 64bit register or 64bit memory location

SUB r8,r/m8 Subtract 8bit register or 8bit memory location from 8bit register

SUB r16,r/m16 Subtract 16bit register or 16bit memory location from 16bit register

SUB r32,r/m32 Subtract 32bit register or 32bit memory location from 32bit register

SUB r64,r/m64 Subtract 64bit register or 64bit memory location from 64bit register

Flag 影响：
OF, SF, ZF, AF, PF, CF 被影响

Protected 模式下例外：
#GP(0) If the destination is located in a non-writable segment
If a memory operand effective address is outside the CS DS ES FS or GS segment limit
If the DS ES FS or GS register contains a NULL segment selector
#SS(0) If a memory operand effective address is outside the SS segment limit
#PF(fault-code) If a page fault occurs
#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3
Real-address 模式下例外
#GP If a memory operand effective address is outside the CS FS ES FS or GS segment limit
#SS If a memory operand effective address is outside the SS segment limit
Virtual-8086 模式下例外
#GP(0) If a memory operand effective address is outside the CS FS ES FS or GS segment limit
#SS(0) If a memory operand effective address is outside the SS segment limit
#PF(fault-code) If a page fault occurs
#AC(0) If alignment checking is enabled and an unaligned memory reference is made
Compatibility 模式下例外
同 Protected 模式
64bit 模式例外
#SS(0) If a memory address referencing the SS segment is in a non-canonical form
#GP(0) if the memory address is in a non-canonical form
#PF(fault-code) If a page fault occurs
#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3

X86 浮点减法指令如下：
FSUB m32fp Subtract m32fp from ST(0) and store result in ST(0)
FSUB m64fp Subtract m64fp from ST(0) and store result in ST(0)
FSUB ST(0),ST(i) Subtract ST(i) from ST(0) and store result in ST(0)
FSUB ST(i),ST(0) Subtract ST(0) from ST(i) and store result in ST(i)
FSUBP ST(i),ST(0) Subtract ST(0) from ST(i) and store result in ST(i),and pop register stack
FSUBP Subtract ST(0) from ST(1) and store result in ST(1),and pop register stack
FISUB m32int Subtract m32int from ST(0) and store result in ST(0)
FISUB m16int Subtract m16int from ST(0) and store result in ST(0)

FSUBR m32fp Subtract ST(0) from m32fp and store result in ST(0)
FSUBR m64fp Subtract ST(0) from m64fp and store result in ST(0)
FSUBR ST(0),ST(i) Subtract ST(0) from ST(i) and store result in ST(0)
FSUBR ST(i),ST(0) Subtract ST(i) from ST(0) and store result in ST(i)
FSUBRP ST(i),ST(0) Subtract ST(i) from ST(0) and store result in ST(i),and pop register stack
FSUBRP Subtract ST(1) from ST(0) and store result in ST(1),and pop register stack
FISUBR m32int Subtract ST(0) from m32int and store result in ST(0)
FISUBR m16int Subtract ST(0) from m16int and store result in ST(0)
//FISUB 和 FISUBR 将定点转化为 X86 的扩展双精度浮点格式(80bit)

FPU Flags 影响：
C1 set to 0 if stack underflow occurred
Set if result was rounded up; cleared otherwise
C0, C2, C3 undefined

浮点例外：
#IS stack underflow occurred
#IA Operand is an SNaN value or unsupported format
Operands are infinities of like sign
#D Source operand is a denormal value
#U Result is too small for destination format
#O Result is too large for destination format
#P Value cannot be represented exactly in destination format

Protected 模式下例外：
#GP(0) If a memory operand effective address is outside the CS DS ES FS or GS segment limit
If the DS ES FS or GS register contains a NULL segment selector
#SS(0) If a memory operand effective address is outside the SS segment limit
#NM CR0.EM[bit2] or CR0.TS[bit3] = 1
#PF(fault-code) If a page fault occurs
#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3
Real-address 模式下例外
#GP If a memory operand effective address is outside the CS FS ES FS or GS segment limit
#SS If a memory operand effective address is outside the SS segment limit
#NM CR0.EM[bit2] or CR0.TS[bit3] = 1
Virtual-8086 模式下例外
#GP(0) If a memory operand effective address is outside the CS FS ES FS or GS segment limit

#SS(0)　If a memory operand effective address is outside the SS segment limit
#PF(fault-code)　If a page fault occurs
#AC(0)　If alignment checking is enabled and an unaligned memory reference is made
#NM　　CR0.EM[bit2] or CR0.TS[bit3] = 1
Compatibility 模式下例外
同 Protected 模式
64bit 模式例外
#SS(0)　If a memory address referencing the SS segment is in a non-canonical form
#GP(0)　if the memory address is in a non-canonical form
#NM　　CR0.EM[bit2] or CR0.TS[bit3] = 1
#MF　　If there is a pending x87 FPU exception
#PF(fault-code)　If a page fault occurs
#AC(0)　If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3

MIPS 的减法如下：
定点减法：
DSUB rd, rs, rt　64bit 减法
　例外：
　Integer Overflow, Reserved Instruction
DSUBU rd, rs, rt 64bit 无符号减法
　例外：
　Reserved Instruction
SUB rd, rs, rt　　32bit 减法
　限制：
　On 64-bit processors, if either GPR rt or GPR rs does not contain sign-extended 32-bit values (bits 63..31 equal), then the result of the operation is UNPREDICTABLE
　例外：
　Integer Overflow
SUBU rd, rs, rt　32bit 无符号减法
　限制：
　On 64-bit processors, if either GPR rt or GPR rs does not contain sign-extended 32-bit values (bits 63..31 equal), then the result of the operation is UNPREDICTABLE
　例外：
　无

MIPS 浮点减法(SUB.fmt)如下：
SUB.S fd, fs, ft　　单精度
SUB.D fd, fs, ft　　双精度

SUB.PS fd, fs, ft　　并行单精度(将 fs 和 ft 的上下两部分分别相减)

限制：

The field fs, ft, fd must specify FPRs valid for operands of type fmt. If they are not valid, the result is UNPREDICTABLE

The operands must be values in format fmt, if they are not, the result is UNPREDICTABLE and the value of the operand FPRs becomes UNPREDICTABLE

The result of SUB.PS is UNPREDICTABLE if the processor is executing in 16 FP registers mode.

例外：

Coprocessor Unusable, Reserved Instruction

浮点例外：

Inexact, Overflow, Underflow, Invalid Op, Unimplemented Op

X86 和 MIPS 减法指令比较：

字长：

X86 的定点减法指令支持 8 位、16 位、32 位、64 位字长；而 MIPS 定点减法支持 32 位和 64 位字长。

浮点：

X86 的浮点减法指令均为 80 位扩展双精度格式；而 MIPS 浮点减法支持单精度、双精度和并行单精度格式。

寻址方式：

MIPS 的减法只支持寄存器寻址。

X86 的定点减法支持寄存器寻址、立即数和内存寻址方式(直接寻址、变址寻址、间接寻址、基址寻址、基址加变址寻址)；

X86 的浮点减法支持寄存器寻址(浮点寄存器栈)和内存寻址方式(直接寻址、变址寻址、间接寻址、基址寻址、基址加变址寻址)。

其他区别：

X86 的定点减法会修改 Flags，浮点减法会修改 FPU flags，而 MIPS 的减法没有 Flags。

X86 的减法和 MIPS 减法产生的例外由于体系结构的不同而有很大不同：

X86 的减法会产生 General-Protection 例外、Stack-Segment Fault 例外；除了在 real-address 模式下之外，还会产生 Page Fault 例外、Alignment Check 例外；所有的浮点减法还会产生 Device Not Available（No Math Coprocessor）例外；在 64bit 模式下进行浮点减法还会产生 x87 FPU Floating Point Error（Math Fault）例外。

MIPS 的 DSUB、DSUBU 和所有的浮点减法会触发保留指令例外，DSUB 和 SUB 会触发溢出例外，浮点减法会触发协处理器不可用例外和一些浮点例外(Inexact, Overflow, Underflow, Invalid Op, Unimplemented Op)

第五章参考答案 静态流水线

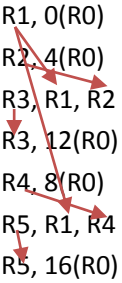
1. 解：

答案按照课程 PPT 及教科书中所述流水线结构予以求解。该流水线有两个特点：（1）数据相关的判断及阻塞是在 ID 级进行的；（2）图中没有画出的前递路径，则认为不存在该类型

前递。寄存器堆的示意图在图 5.2 中，并不存在寄存器堆的读写旁路，因此 RAW 相关在 WB 和 ID 两级流水上产生的冲突不会自动消除。存在的是运算指令 EX 到 EX 的前递，以及 MEM 到 EX 的前递。

小常识：前递不能在时间线上跨拍：第 4 拍的 MEM，不能给隔一拍第 6 拍的 EX 前递。

```
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)
```



	1	2	3	4	5	6	7	8	9
LW R1, 0(R0)	**IF**	**ID**	**EX**	**MEM**	**WB**				
LW R2, 4(R0)		**IF**	**ID**	**EX**	**MEM**	**WB**			
ADD R3, R1, R2			**IF**	**ID**	**ID**	**ID**	**ID**	**EX**	**MEM**
SW R3, 12(R0)				**IF**	**IF**	**IF**	**IF**	**ID**	**EX**
LW R4, 8(R0)								**IF**	**ID**
ADD R5, R1, R4									**IF**
SW R5, 16(R0)									

	10	11	12	13	14	15
LW R1, 0(R0)						
LW R2, 4(R0)						
ADD R3, R1, R2	**WB**					
SW R3, 12(R0)	**MEM**	**WB**				
LW R4, 8(R0)	**EX**	**MEM**	**WB**			
ADD R5, R1, R4	**ID**	**ID**	**EX**	**MEM**	**WB**	
SW R5, 16(R0)	**IF**	**IF**	**ID**	**EX**	**MEM**	**WB**

排序前共需要 15 拍。

重排后的指令序列为：

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

	1	2	3	4	5	6	7	8	9	10	11	12

LW R1, 0(R0)	IF	ID	EX	MEM	WB							
LW R2, 4(R0)		IF	ID	EX	MEM	WB						
LW R4, 8(R0)			IF	ID	EX	MEM	WB					
ADD R5, R1, R4				IF	ID	ID	EX	MEM	WB			
SW R5, 16(R0)					IF	IF	ID	EX	MEM	WB		
ADD R3, R1, R2							IF	ID	EX	MEM	WB	
SW R3, 12(R0)								IF	ID	EX	MEM	WB

通过指令重排，上述指令序列只需要 12 拍，减少了 3 拍。

利用了存在 MEM 到 EX 的前递，但是没有 WB 到 EX 前递的特点。

2. 解： **题目要求无旁路，注意读题即可**

MIPS 代码如下：

```
LW      R1, 0(R0)      ; load A
LW      R2, 4(R0)      ; load B
ADD     R3, R1, R2     ; A=A+B
SUB     R4, R3, R2     ; C=A-B
SW      R3, 0(R0)      ; store A
SW      R4, 8(R0)      ; store C
```

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
LW R1, 0(R0)	IF	ID	EX	MEM	WB										
LW R2, 4(R0)		IF	ID	EX	MEM	WB									
ADD R3, R1, R2			IF	ID	ID	ID	ID	EX	MEM	WB					
SUB R4, R3, R2				IF	IF	IF	IF	ID	ID	ID	ID	EX	MEM	WB	
SW R3, 0(R0)								IF	IF	IF	IF	ID	EX	MEM	WB
SW R4, 8(R0)												IF	ID	ID	ID

	16	17	18
LW R1, 0(R0)			
LW R2, 4(R0)			
ADD R3, R1, R2			
SUB R4, R3, R2			
SW R3, 0(R0)			
SW R4, 8(R0)	EX	MEM	WB

3. 解： **注意理解什么是“向量运算 X(i)=a*X(i)+Y(i)”**

假设此处的浮点运算均为单精度浮点运算。数组的长度为 N(N<2^{15}-1)。

```
        ADDIU   R3, R0, N       ; N
        SLL     R3, R3, 2       ; N*4
        ADDU    R3, R1, R3      ; R3=R1+N*4
Loop:   LWC1    F1, 0(R1)       ; load X(i)
        LWC1    F2, 0(R2)       ; load Y(i)
```

	MUL.S F3, F0, F1	; a*X(i)
	ADD.S F4, F2, F3	; a*X(i)+Y(i)
	ADDIU R1, R1, 4	; i++ for X
	ADDIU R2, R2, 4	; i++ for Y
	BNE R1,R3, Loop	
	SWC1 F4, -4(R1)	; store to X(i)

本题假设采用图 5.16 所示结构

	1	2	3	4	5	6	7	8	9
LWC1 F1, 0(R1)	IF	ID	EX	MEM	WB				
LWC1 F2, 0(R2)		IF	ID	EX	MEM	WB			
MUL.S F3, F0, F1			IF	ID	EX1	EX2	EX3	EX4	MEM
ADD.S F4, F2, F3				IF	ID	ID	ID	ID	EX1
ADDIU R1, R1, 4					IF	IF	IF	IF	ID
ADDIU R2, R2, 4									IF
BNE R1,R3, Loop									
SWC1 F4, -4(R1)									

	10	11	12	13	14	15	16	17
LWC1 F1, 0(R1)								
LWC1 F2, 0(R2)								
MUL.S F3, F0, F1	WB							
ADD.S F4, F2, F3	EX2	EX3	MEM	WB				
ADDIU R1, R1, 4	ID	ID	EX	MEM	WB			
ADDIU R2, R2, 4	IF	IF	ID	EX	MEM	WB		
BNE R1,R3, Loop			IF	ID	EX	MEM	WB	
SWC1 F4, -4(R1)				IF	ID	EX	MEM	WB

4. 解： **注意读题，本题新增了寄存器的读写旁路**

(1) 无旁路部件，前 99 次循环的流水线时空图如下

	1	2	3	4	5	6	7	8	9	10	11
LD R1, 0(R2)	IF	ID	EX	MEM	WB						
DADDI R1, R1, 4		IF	ID	ID	ID	EX	MEM	WB			
SD R1, 0(R2)			IF	IF	IF	ID	ID	ID	EX	MEM	WB
DADDI R2, R2, 4						IF	IF	IF	ID	EX	MEM
DSUB R4, R3, R2									IF	ID	ID
BNEZ R4, Loop										IF	IF
（延迟槽 NOP）											
LD R1, 0(R2)											

	12	13	14	15	16	17	18	19	20	21
LD R1, 0(R2)										

DADDI R1, R1, 4										
SD R1, 0(R2)										
DADDI R2, R2, 4	WB									
DSUB R4, R3, R2	ID	EX	MEM	WB						
BNEZ R4, Loop	IF	ID	ID	ID	EX	MEM	WB			
（延迟槽 NOP）		IF	IF	IF	ID	EX	MEM	WB		
LD R1, 0(R2)						IF	ID	EX	MEM	WB

前 99 次循环，每次都是预测错，每个循环 16 个时钟周期。最后一个循环，预测对，但整个循环最后一条指令写回需要 19 个时钟周期（**注意延迟槽**）。

共计：16×99+19=1603 时钟周期

(2) 有旁路，预测 taken，前 99 次循环流水线时空图如下：

	1	2	3	4	5	6	7	8	9	10	11	12	13
LD R1, 0(R2)	IF	ID	EX	MEM	WB								
DADDI R1, R1, 4		IF	ID	ID	EX	MEM	WB						
SD R1, 0(R2)			IF	IF	ID	EX	MEM	WB					
DADDI R2, R2, 4					IF	ID	EX	MEM	WB				
DSUB R4, R3, R2						IF	ID	EX	MEM	WB			
BNEZ R4, Loop							IF	ID	EX	MEM	WB		
（延迟槽 NOP）								IF	ID	EX	MEM	WB	
LD R1, 0(R2)									IF	ID	EX	MEM	WB

前 99 次循环，执行一个循环需要 8 拍。最后一次循环完整执行完需要 12 拍。

共计：8×99+12=804 时钟周期

(3) 仍认为分支预测 taken

```
Loop:
    LD      R1, 0(R2)
    DADDI   R2, R2, #4
    DSUB    R4, R3, R2
    DADDI   R1, R1, #4
    BNEZ    R4, Loop
    SD      -4(R2), R1
```

	1	2	3	4	5	6	7	8	9	10	11
LD R1, 0(R2)	IF	ID	EX	MEM	WB						
DADDI R2, R2, 4		IF	ID	EX	MEM	WB					
DSUB R4, R3, R2			IF	ID	EX	MEM	WB				
DADDI R1, R1, 4				IF	ID	EX	MEM	WB			
BNEZ R4, Loop					IF	ID	EX	MEM	WB		
SD -4(R2), R1						IF	ID	EX	MEM	WB	
LD R1, 0(R2)							IF	ID	EX	MEM	WB

前 99 次循环，执行一个循环需要 6 拍。最后一次循环完整执行完需要 10 拍。

整个循环共计：6×99+10=604 时钟周期

5. 解： **常识题**

空操作指令(nop 指令)，其不改变程序可见寄存器、状态寄存器以及内存的状态，以及用于等待需要一定周期执行的操作。nop 指令的作用，常见的有：取指的强制访存对齐（memory alignment），防止相关风险（hazard），以及用于填充延迟槽（branch delay slot）。

7. 解：

(1) ALU 模块

```
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
```

如果信号声明不加 signed 标记，则都为无符号操作（包括小于号“<”、算术右移号“>>>”）。