1.

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<br>Push C<br>Sub<br>Pop A<br>Push A<br>Push C<br>Sub<br>Pop D<br>Push D<br>Push A<br>Add<br>Pop B | Load C<br>Neg<br>Add B<br>Store A<br>Load C<br>Neg<br>Add A<br>Store D<br>Add A<br>Store B | Load R1,B<br>Sub R1,C<br>Store R1,A<br>Sub R1,C<br>Store R1,D<br>Add R1,A<br>Store R1,B | Load R1,B<br>Load R2,C<br>Sub R3,R1,R2<br>Store R3,A<br>Sub R4,R3,R2<br>Store R4,D<br>Add R5,R4,R3<br>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 | | | | √ |

(累加器型有的版本有 sub 指令，如果有，两个都是累加器型最优)

2.

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.

解：假设 CPU A 总指令数为 x，转移指令有 0.25x，条件码指令 0.25x，其它指令 0.5x

A 执行周期数 2*0.25x+0.75x=1.25x

则 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

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：

题目要求小尾端，因此有：

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

7.

解：

```
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)