1. +3.

(a) 每1.5年，transistor/chip area 增加1倍 ($^{new}/_{old}=2$).

+4.

(b) RISC: reduced instruction set computer. 用基本的 operation 去完成複雜的功能，ex: MIPS, ARM.

CISC: complex instruction set computer. 針對複雜的功能都設計一個 operation 給它，ex: intel 80x86.

+3.

(c) SPEC所給的 benchmark，包含許多測試程式，可藉由跑出來的分數去比較不同電腦的效能.

2. (a). +3

cycle time equal ⇒ 比 instruction ↳ 比 I

$$\frac{12\times 0.25I}{(12\times 0.25+3\times 0.2+4\times 0.3+5\times 0.25)I}=\frac{300}{300+60+120+125}=\frac{300}{605}\doteq 49.6\%$$

(b). +7.

old cycle: $(12\times 0.25+3\times 0.2+4\times 30\%+5\times 0.25)I=6.05I$

new cycle: $(8\times 0.25+3\times 0.2+4\times 0.3+5\times 0.25)I=5.05I$

$$\frac{\text{speed new}}{\text{speed old}}=\frac{\text{time old}}{\text{time new}}=\frac{6.05I\times \text{rate}}{5.05I\times 1.2\times \text{rate}}=\frac{6.05}{6.06}\approx 0.99.$$ ⇒ 改善後較慢.

3.

```
WH: sll  $t0, $s1, 2
    add  $t1, $s0, $t0      # $t1 = A[i]'s address
    lw   $t2, 0($t1)        # $t2 = A[i]

    slt  $t3, $t2, $s2
    beq  $t3, $zero, EXIT

    addi $t4, $t1, -4       # $t4 = A[i-1]'s address
    lw   $ts, 0($t4)        # $ts = A[i-1]

    addi $t6, $ts, 3
    sw   $t6, 0($t1)
    addi $s1, $s1, -1
    j    WH
EXIT:
```

4. +10

1. Simplicity favors regularity, ex: 1. "every" instruction is 32-bits
   2. Formats are regulated

2. Smaller is faster : ex: 1. When operating, we use registers rather than memories.
   2. Each word is "only" 32-bits

3. Make common case faster : ex: 1. addi, subi
   2. $zero

4. Good design makes good compromise: ex: 1. I-format, R-format, J-format.
   明明都是 32-bits 的 instruction, 但還是會因 instruction 的特性給予裡面的 fields 不同的 bits。

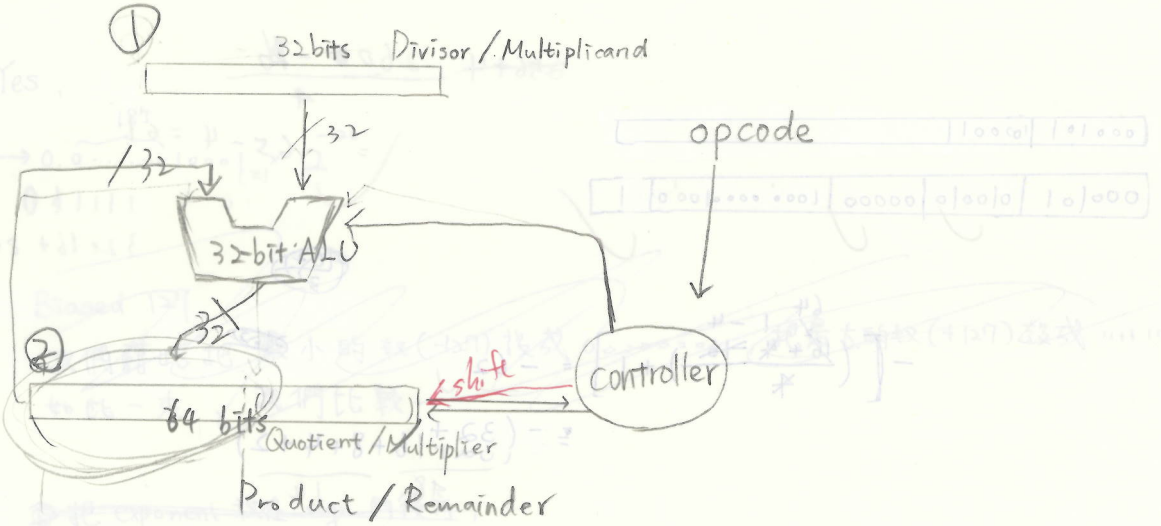
5. +15

```
caller:  addi  $sp , $sp , -8
         sw    $a0 , 4($sp)
         sw    $a1 , 0($sp)
         j     SR

SR:      addi  $sp , $sp , -8
         sw    $s0 , 4($sp)
         sw    $s1 , 0($sp)
         .
         .
         .
         lw    $s0 , 4($sp)
         lw    $s1 , 0($sp)
         addi  $sp , $sp , 8
         jr    ra
```

```
lw   $a0 , 4($sp)
lw   $a1 , 0($sp)
addi $sp , $sp , 8
```

6. +10



How it performs:

1. Multiply:
   (1) We put the multiplicand on the ①, and the Multiplier on the right-half of ②.
   (2) Read least-significant bit of Multiplier, if it's 1, add multiplicand to the left-half of ②. Else, do nothing.
   (3) Shift the total ② to right.
   (4) Check whether we have done 32 times, if we have, go to (5). If not, go to (2) to do again.
   (5) The ② (64-bits) is the result (product) we want.

2. Devide:
   (1) We first put the 被除数 (64-bits) on ②, and the divisor on ①.
   (2) First shift left the ② for 1-bit.
   (3) Use the ②'s left-half to minus divisor, put the result in [②'s left-half].
   (4) Check whether it's negative. If negative, restore it by add divisor to it and place 0 in the least-significant bit of ②. Else, place 1 in the least-significant bit of ②.
   (5) Left-shift the ②.
   (6) Check whether we have done 32 times. If we have, go to (7). Else, go to (3) and do again.
   (7) The Right-half of ② is Quotient, and Left-half is Remainder (final)

7.

(5) rs : 00010

rt : 00000

immediate : 1111111 11100010

PC → $0108_{(16)} = 264_{(10)}$

L1 → $0010_{(16)} = 16_{(10)}$

$\frac{264-16}{4} = 62_{(10)} = 0000000111110$

0000

$-62 =$ 1111111 11100010

8.

(10) (a) Yes. it is denormalized number

for it's exponent is 00000000

but significant ≠ 0

$0.0000000000000000001000 1 \times 2^{-126}$

$= 1.0001 \times 2^{-145}$

(b) comparison 從 msb 開始

所以從 sign bit 開始判斷大小、

sign bit 一樣則判斷 exponent

所以 exponent 擺 30 - 23 bits，再來才是 signi

最重要的是，exponent 採用 bias 127

9.
(1) hand manual: 把小的 exponent 變大 對齊
and shift significant.

採用較大的 exponent : left
so. C1 = 0

較小的F.P.: 的 significant shift right : C2 = 1

Big ALU 加兩個 F.P. 的 significants
既然一邊已經是較小F.P.的significant
另一邊就是較大的 so C3 = 0

ant.