

Answers to HW1

1.51(10)

Cost per die = Cost per wafer/(Dies per wafer X Yield) = 6000/(1500 X 50%) = 8

Cost per chip = (Cost per die + Cost\_packaging + Cost\_testing)/Test yield = (8 + 10)/90% = 20

Price = Cost per chip X (1 + 40%) = 28

If we need to sell  $n$  chips, then  $500,000 + 20n = 28n$

$$n = 62,500$$

1.59(10)

We can write Dies per wafer =  $f(\text{Die area}^{-1})$  and Yield =  $f(\text{Die area}^{-2})$

and thus Cost per die =  $f(\text{Die area}^3)$ .

Cost increases much quickly as die area increases, so designers shall draw their layout compactly to save area.

1.61(10)

From the caption in Figure 1.15, we have 165 dies at 100% yield. If the defect density is 1 per square centimeter, then the yield is approximated by

$$\frac{1}{\left[1 + \left(\frac{\frac{1}{100\text{mm}^2} \times 250\text{mm}^2}{2}\right)\right]^2} = .198.$$

Thus,  $165 \cdot .198 = 32$  dies with a cost of  $\$1000/32 = \$31.25$  per die.

1.62(10)

Defects per area.

1.63(10)

$$\text{Yield} = \frac{1}{(1 + \text{Defects per area} \times \text{Die area}/2)^2}$$

$$\text{Defects per area} = \frac{2}{\text{Die area}} \left( \frac{1}{\sqrt{\text{Yield}}} - 1 \right)$$

1.64(10)

1980	Die area	0.16
	Yield	0.48
	Defect density	5.54
1992	Die area	0.97
	Yield	0.48
	Defect density	0.91
1992 + 1980	Improvement	6.09

4.10(10)

Using C1, the average CPI for I1 is  $(.4 * 2 + .4 * 3 + .2 * 5) = 3$ , and the average CPI for I2 is  $(.4 * 1 + .2 * 2 + .4 * 2) = 1.6$ . Thus, with C1, I1 is  $((6 \cdot 10^9 \text{cycles/second}) / (3 \text{ cycles/instruction})) / ((3 \cdot 10^9 \text{cycles/second}) / (1.6 \text{ cycles/instruction})) = 16/15$  times as fast as I2.

Using C2, the average CPI for I2 is  $(.4 * 2 + .2 * 3 + .4 * 5) = 3.4$ , and the averageCPI for I2 is  $(.4 * 1 + .4 * 2 + .2 * 2) = 1.6$ . So with C2, I2 is faster than I1 by factor of  $((3 \cdot 10^9 \text{cycles/second}) / (1.6$

$\text{cycles/instruction}) / ((6 \cdot 10^9 \text{ cycles/second}) / (3.4 \text{ cycles/instruction})) = 17/16$ .

For the rest of the questions, it will be necessary to have the CPIs of I1 and I2 on programs compiled by C3. For I1, C3 produces programs with CPI  $(.6 * 2 + .15 * 3 + .25 * 5) = 2.9$ . I2 has CPI  $(.6 * 1 + .15 * 2 + .25 * 2) = 1.4$ .

The best compiler for each machine is the one which produces programs with the lowest average CPI. Thus, if you purchased either I1 or I2, you would use C3.

Then performance of I1 in comparison to I2 using their optimal compiler (C3) is  $((6 \cdot 10^9 \text{ cycles/second}) / (2.9 \text{ cycles/instruction})) / ((3 \cdot 10^9 \text{ cycles/second}) / (1.4 \text{ cycles/instruction})) = 28/29$ . Thus, I2 has better performance and is the one you should purchase.

#### 4.14(10)

The total execution time for the machines are as follows:

Computer A =  $2 + 20 + 200$  seconds = 222 seconds

Computer B =  $5 + 20 + 50$  seconds = 75 seconds

Computer C =  $10 + 20 + 15$  seconds = 45 seconds

Thus computer C is faster. It is  $75/45 = 5/3$  times faster than computer B and  $222/45 = 74/15$  times faster than computer A.

#### 4.15(10)

With the new weighting the total execution time for the group of programs is:

Computer A =  $8 * 2 + 2 * 20 + 1 * 200$  seconds = 256 seconds

Computer B =  $8 * 5 + 2 * 20 + 1 * 50$  seconds = 130 seconds

Computer C =  $8 * 10 + 2 * 20 + 1 * 15$  seconds = 135 seconds

So with this workload, computer B is faster by a factor of  $135/130 = 1.04$  with respect to computer C and a factor of  $256/130 = 1.97$  with respect to computer A. This new weighting reflects a bias from the previous results by a bias toward program 1 and program 2, which resulted in computer A and computer B looking comparatively better than before.

#### 4.22(10)

Using Amdahl's law (or just common sense), we can determine the following:

Speedup if we improve only multiplication =  $100 / (30 + 50 + 20/4) = 100/85 = 1.18$ .

Speedup if we only improve memory access =  $100 / (30 + 50/2 + 20) = 100/75 = 1.33$ .

Speedup if both improvements are made =  $100 / (30 + 50/2 + 20/4) = 100/60 = 1.67$ .

#### 2.15(30)

Hence, the results from using *if-else* statements are better:

```
set_array:    addi    $sp, $sp, -52 # move stack pointer
              sw    $fp, 48($sp) # save frame pointer
              sw    $ra, 44($sp) # save return address
              sw    $a0, 40($sp) # save parameter (num)
              addi  $fp, $sp, 48 # establish frame pointer
              add  $s0, $zero, $zero # i = 0
              addi $t0, $zero, 10 # max iterations is 10
loop:        sll  $t1, $s0, 2 # $t1 = i * 4
              add $t2, $sp, $t1 # $t2 = address of array[i]
              add $a0, $a0, $zero # pass num as parameter
              add $a1, $s0, $zero # pass i as parameter
              jal  compare # call compare(num, i)
```

```

sw $v0, 0($t2) # array[i] = compare(num, i);
addi $s0, $s0, 1
bne $s0, $t0, loop # loop if i<10
lw $a0, 40($sp) # restore parameter (num)
lw $ra, 44($sp) # restore return address
lw $fp, 48($sp) # restore frame pointer
addi $sp, $sp, 52 # restore stack pointer
jr $ra # return
compare:
addi $sp, $sp, -8 # move stack pointer
sw $fp, 4($sp) # save frame pointer
sw $ra, 0($sp) # save return address
addi $fp, $sp, 4 # establish frame pointer
jal sub # can jump directly to sub
slt $v0, $v0, $zero # if sub(a,b) >= 0, return 1
sli $v0, $v0, 1
lw $ra, 0($sp) # restore return address
lw $fp, 4($sp) # restore frame pointer
addi $sp, $sp, 8 # restore stack pointer
jr $ra # return
sub:
sub $v0, $a0, $a1 # return a-b
jr $ra # return

```

The following is a diagram of the status of the stack:

