Selected Solutions

to

M. J. Flynn's

Computer Architecture: Pipelined and Parallel Processor Design

July 24, 1996

These solutions are under development. For the most recent edition, check our dated web files: http://umunhum.stanford.edu/.

You must be a course instructor to access this site. Phone 800-832-0034 for authorization. Comments and corrections on the book or solution set are welcome; kindly send email to flynn@ee.stanford.edu. Publisher comments to Jones & Bartlett, One Exeter Plaza, Boston MA 02116.

Contents

Chapter 1. Architecture and Machines

Problem 1.8

A certain computation results in the following (hexadecimal) representation:

Fraction	`+1.FFFF...`
Exponent (unbiased)	`+F` (radix 2)

Show the floating-point representation for the preceding in:

a. IBM long format.

b. IEEE short format.

c. IEEE long format.

We are given a floating point value `+1.FFFF...` with an unbiased exponent of `+F` which corresponds to the value 2^{+15}. For simplicity (and clarity), we will represent this as `+1.FFFF...`$\times 2^{+15}$. the first thing to note is that any rounding (truncation not being considered rounding) will result in rounding the value up. Taking rounding into account would result in the (finite) value `+2.0000...0`$\times 2^{+15}$. We will use this as the basis for our answers in the following solutions.

Note that the form `1234ABCD` will be used to represent hex values and *1010101* will be used to represent binary values. This notation is being used since the numbers that are being used are, by necessity, fractional and traditional representations (for example, the C language form `0x1234abcd`) don't seem as clear when dealing with fractional values. Also note that since there are no clear specifications of the actual encoding of the sign, mantissa, and characteristic into a machine word for the different representations we will only present the signed mantissa and the characteristic—true to their specified lengths and in the same number system but not packed into a machine representation.

a. IBM long format. IBM long format—hex-based (base 16) with no leading digit, characteristic bias 64, 14 digit mantissa.

   We need to convert the value to use an exponent base of 16 and then adjust it to get a normalized form for this representation. A normalized IBM representation has its most-significant (non-zero) digit immediately to the right of the decimal point. We also need to convert the exponent to correspond to base 16—this will fall out as we perform the conversion. Thus, shifting the mantissa for the value `+2.0000...0`$\times 2^{+15}$ right one power of 2—one bit—(don't forget to adjust the exponent the right direction!) we get `+1.0000...0`$\times 2^{+16}$ which can be rewritten using a hex-based exponent as `+1.0000...0`$\times 16^{+4}$.

   We now have the right base for the exponent but do not have a valid IBM floating-point value. To get this we need to shift the mantissa right by one hex digit—four bits—resulting in `+0.1000...0`$\times 16^{+5}$. This is a valid hex-based mantissa that has a non-zero leading digit. Note that the leading three bits of this representation are zero and represent wasted digits as far as precision is concerned—they could have been much more useful holding information at the other end of the representation.

   Next, in order to get the characteristic for this value we need to add in the bias value to the exponent—which is 64. This results in a characteristic of +69 (in a hex representation we have `45`. Finally, we need to limit the result to 14 hex digits—a total of 56 bits although the loss of

precision in the leading digit makes this comparison less exact—which gives us a mantissa of `+0.10000000000000` with a characteristic value of `45`.

If rounding is not taken into account the (truncated) mantissa would be `+0.FFFFFFFFFFFFF` with a characteristic value of `44`

b. IEEE short format. IEEE short format—binary based, leading (hidden) 1 to the left of the decimal point, characteristic bias 127, 23 bit mantissa (not including the hidden 1).

As before, we need to adjust the value to get it into a normalized form for this representation. A normalized IEEE (short or long) has a leading (hidden) 1 to the left of the decimal point. Thus, shifting the mantissa right one bit we get `+1.0000...0`$\times 2^{+16}$ which, miraculously, is normalized!. The key point to note is that the *representation* of mantissa is all zeros despite the fact that the mantissa itself is non-zero. This is due to the fact that the leading 1 is a hidden value saving one bit of the mantissa and resulting in a greater precision for the physical storage. This may be written as `+(1).0000...0`$\times 2^{+16}$ to make the hidden 1 value explicit.

The question arises, if the mantissa (as stored) is all zeros, how do we distinguish this value from the actual floating point value is 0.00? This is really quite easy—although the mantissa is zero, the representation of the value as a whole is non-zero due to a non-zero characteristic. For simplicity and compatibility we define the representation of all zeros—both mantissa and characteristic—to *be* the value 0.00. This makes sense for two reasons. First, it allows simple testing for zero to be performed—floating point and integer representations for zero are the same value. Second, it maps relatively cleanly into the range of values represented in the floating point system—and, due to the use of the characteristic instead of a normal exponent, is smaller than the smallest representable value. Now, we need to compute the characteristic and convert the value to binary—both fairly straightforward.

Using the IEEE short format values of bias and mantissa size, we get

$$+(1).00000000000000000000000$$

for the mantissa and a value of 143 (or *10001111*) for the characteristic.

If rounding is not taken into account the (truncated) mantissa would be

$$+(1).11111111111111111111111$$

with a characteristic of 142 (or *10001110*).

c. IEEE long format. IEEE long format—binary based, leading (hidden) 1 to the left of the decimal point, characteristic bias 1023, 52 bit mantissa (not including the hidden 1).

As in the previous case, we get a normalized mantissa by shifting the mantissa right one bit resulting in `+1.0000...0`$\times 2^{+16}$. Now, we need to compute the characteristic and convert the value to binary—both fairly straightforward but using different values for both the bias and number of bits in the mantissa from the short form. This was done for the IEEE representation to ensure that both the range and precision of the values representable were scaled comparably as the size of the representation grew. Previously, only the mantissa had grown as the representation size was increased. This did not prove to be an effective use of bits.

Now, using the IEEE long format values of bias and mantissa size, we get

$$+(1).0000000000000000000000000000000000000000000000000000$$

for the mantissa and a value of 1039 (or *10000001111*) for the characteristic.

If rounding is not taken into account the (truncated) mantissa would be

$$+(1).1111111111111111111111111111111111111111111111111111$$

with a characteristic of 1038 (or *10000001110*).

Problem 1.9

Represent the decimal numbers (i) +123, (ii) −4321, and (iii) +00000 (zero is represented as a positive number) as:

a. Packed decimal format (in hex).

b. Unpacked decimal format (in hex).

Assume a length indicator (in bytes) is specified in the instruction. Show lengths for each case.

First, let's break down these numbers into sequences of BCD digits—from there we can easily produce the packed or unpacked forms.

(i) $+123 \rightarrow \{+, 1, 2, 3\}$

(ii) -4321 $\rightarrow \{-, 4, 3, 2, 1\}$

(iii) $+00000 \rightarrow \{+, 0, 0, 0, 0, 0\}$

Second, the answers will be presented in big-endion sequence for simplicity of reading. This is neither an endorsement of big-endion nor a rejection of little-endion as an architectural decision—only as a presentation mechanism.

Third, note that the hex notations for the sign differ between packed and unpacked representations. For the packed representation the 4-bit ("nibble") values are '`A`' for '`+`' and '`B`' for '`-`', while for the unpacked representation the byte values are '`2B`' for '`2D`' and '`B`' for '`-`'. The latter is standard ASCII text encoding (although we would typically write the sign bit first in text).

a. Packed decimal format (in hex).

For packed values we can put two digits in one byte. Recall that there must be an odd number of digits followed by a trailing sign field, and thus we must pad case (ii) which doesn't have this property. Thus, we get:

(i) {`12`, `3A`}, 2 bytes

(ii) {`04`, `32`, `1B`}, 3 bytes

(iii) {`00`, `00`, `0A`}, 3 bytes

b. Unpacked decimal format (in hex).

For unpacked values we put only one digit in a byte so that there are no restrictions with respect to length. Thus we get:

(i) {`31`, `32`, `33`, `2B`}, 4 bytes

(ii) {`34`, `33`, `32`, `31`, `2D`}, 5 bytes

(iii) {`30`, `30`, `30`, `30`, `30`, `2B`}, 6 bytes

Problem 1.10

(a) R/M machine

In this solution, the assumption made is that there are no unnecessary memory addresses used. The necessary memory addresses are `Addr`, `Bddr`, and `Cddr` to hold the coefficients for the quadratic equation and `ROOT1` and `ROOT2` to hold the solutions.

And the `R/M` version:

LD	R1, BASE	; load offset address
LD.D	F5, Bddr[R1]	; F5 $\leftarrow$ B
MPY.D	F5, F5	; F5 $\leftarrow$ B^2
LD.D	F6, #4.0	; F6 $\leftarrow$ 4
MPY.D	F6, Addr[R1]	; F6 $\leftarrow$ 4A
MPY.D	F6, Cddr[R1]	; F6 $\leftarrow$ 4AC
SUB.D	F5, F6	; F5 $\leftarrow$ $B^2 - 4AC$
SQRT.D	F5	; F5 $\leftarrow$ $\sqrt{B^2 - 4AC}$
LD.D	F7, Baddr[R1]	; F7 $\leftarrow$ B
MPY.D	F7, #-1	; F7 $\leftarrow$ $-B$
MOV.D	F8, F7	; F8 $\leftarrow$ $-B$
SUB.D	F7, F5	; F7 $\leftarrow$ $-B - \sqrt{B^2 - 4AC}$
ADD.D	F8, F5	; F8 $\leftarrow$ $-B + \sqrt{B^2 - 4AC}$
LD.D	F9, Addr[R1]	; F9 $\leftarrow$ A
SHL.D	F9, #1	; F9 $\leftarrow$ 2A
DIV.D	F7, F9	; F7 $\leftarrow$ $\frac{-B-\sqrt{B^2-4AC}}{2A}$
DIV.D	F8, F9	; F8 $\leftarrow$ $\frac{-B+\sqrt{B^2-4AC}}{2A}$
ST.D	ROOT1[R1], F7	; ROOT1 $\leftarrow$ $\frac{-B-\sqrt{B^2-4AC}}{2A}$
ST.D	ROOT2[R1], F8	; ROOT1 $\leftarrow$ $\frac{-B+\sqrt{B^2-4AC}}{2A}$

(b) L/S machine

LD	R1,BASE	; load offset address
LD.D	F2,Bddr[R1]	; F2 $\leftarrow$ B
LD.D	F3,Cddr[R1]	; F3 $\leftarrow$ C
LD.D	F4,Addr[R1]	; F4 $\leftarrow$ A
MPY.D	F5,F2,F2	; F5 $\leftarrow$ B^2
MPY.D	F6,F3,F4	; F6 $\leftarrow$ AC
SHL.D	F6,#2	; F6 $\leftarrow$ 4AC
SUB.D	F6,F5,F6	; F6 $\leftarrow$ $B^2 - 4AC$
SQRT.D	F6,F6	; F6 $\leftarrow$ $\sqrt{B^2 - 4AC}$
MPY.D	F2,F2,#-1	; F2 $\leftarrow$ $-B$
SUB.D	F7,F2,F6	; F7 $\leftarrow$ $-B - \sqrt{B^2 - 4AC}$
ADD.D	F8,F2,F6	; F8 $\leftarrow$ $-B + \sqrt{B^2 - 4AC}$
SHL.D	F9,#1	; F4 $\leftarrow$ 2A
DIV.D	F7,F7,F9	; F7 $\leftarrow$ $\frac{-B-\sqrt{B^2-4AC}}{2A}$
DIV.D	F8,F8,F9	; F8 $\leftarrow$ $\frac{-B+\sqrt{B^2-4AC}}{2A}$
ST.D	ROOT1[R1],F7	; ROOT1 $\leftarrow$ F7
ST.D	ROOT2[R1],F8	; ROOT2 $\leftarrow$ F8

Chapter 2. The Basics

Problem 2.1

A four segment pipeline implements a function and has the following delays for each segment: ($b = .2$)

Segment #	Maximum delay
1	17
2	15
3	19
4	14

Where $c = 2\,\text{ns}$,

a. What is the cycle time that maximizes performance without allocating multiple cycles to a segment?

   Since the maximum stage delay is 19 ns, this is the shortest possible delay time that does not require multiple cycles for any given stage. Thus, $19 + 2 = 21\,\text{ns}$ is the minimum cycle time for this pipeline.

b. What is the total time to execute the function through all stages?

   There are four stages, each of which is clocked at 21 ns. $4 \times 21 = 84\,\text{ns}$ is thus the total delay (latency) through the pipeline.

c. $S_{\text{opt}} = \sqrt{\frac{(1-0.2)(17+15+19+14)}{(0.2)(2)}} \approx 11.$

   Let $T_{\text{seg}} = 7\,\text{ns}$

   $S = 11$

   $G = \frac{1}{1+10\times.2}\frac{1}{7+2} = 37.0$ MIPS

   Let $T_{\text{seg}} = 7.5\,\text{ns}$

   $S = 10$

   $G = \frac{1}{1+9\times.2}\frac{1}{7.5+2} = 37.6$ MIPS

   Let $T_{\text{seg}} = 8.5\,\text{ns}$

   $S = 9$

   $G = \frac{1}{1+8\times.2}\frac{1}{8.5+2} = 36.6$ MIPS

   Let $T_{\text{seg}} = 9.5\,\text{ns}$

   $S = 8$

   $G = \frac{1}{1+7\times.2}\frac{1}{9.5+2} = 36.2$ MIPS

   The cycle time that maximizes performance $= 7.5 + 2 = 9.5\,\text{ns}$

Problem 2.2

Repeat problem 1 if there is a 1 ns clock skew (uncertainty of $\pm 1\,\text{ns}$) in the arrival of each clock pulse.

a. What is the minimum cycle time without allocating multiple cycles to a segment?

Since the maximum stage delay is 21 ns, the minimum cycle time should be $21 + 2 = 23$ ns.

b. What is the total time to execute the function through all stages?

There are four stages, each of which is clocked at 23 ns. $4 \times 23 = 92$ ns is thus the total delay (latency) through the pipeline.

c. Now $c = 4$.

$$S_{\text{opt}} \sqrt{\frac{(1 - .2)(76)}{(.2)4}} = 8 \text{ or } 9$$

Use 8:

$$\text{Cycle time} = \frac{7.6}{8} + 4 = 13.5 \text{ ns}.$$

Problem 2.3

a. No uncontrolled clock skew allowed

Minimum clock cycle time = largest $P_{\max} + C = 16 + 3 = 19$ ns

Latency = 4 (19) = 76 ns

b. Wave pipelining allowed

$\Delta T_{\min}$ = largest $(P_{\max} - P_{\min}) + C = 7 + 3 = 10$ ns

Latency = $(14 + 3) + (12 + 3) + (16 + 3) + (11 + 3) = 65$ ns

$CS_1 \uparrow = 17 \mod 10 = 7$ ns (or -3 ns)
$CS_2 \uparrow = 32 \mod 10 = 2$ ns
$CS_3 \uparrow = 51 \mod 10 = 1$ ns
$CS_4 \uparrow = 65 \mod 10 = 5$ ns

Problem 2.5

$T = 120$ ns, $C = 5$ ns, $b = 0.2$, $S_{\text{opt}} = \sqrt{\frac{(1-b)(1+k)T}{bC}}$.

k	S_{opt}
0.05	10.04
0.08	10.18
0.11	10.32
0.14	10.46
0.17	10.60
0.20	10.73

Optimum segments vs. overhead is shown in figure 1.

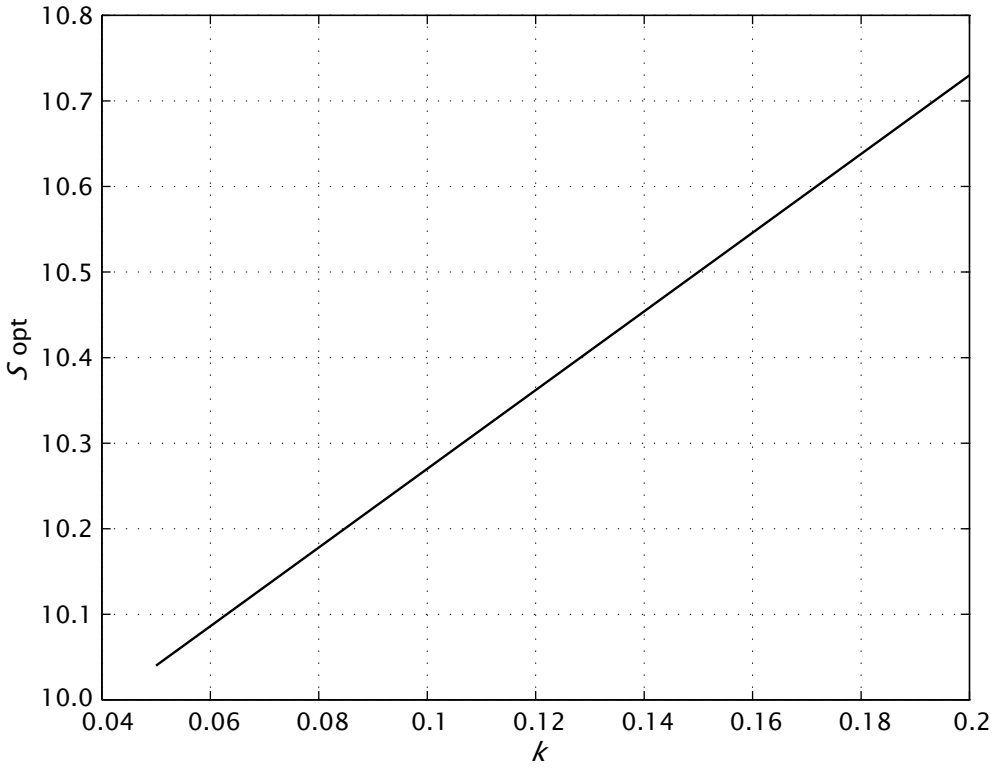


Figure 1: Problem 2-5.

Problem 2.6

a. $G = \frac{1}{(1+b(S-a))(C+T(1+k)/S)}$

Let $\frac{dG}{dS} = 0$

$-b(CS^2 + T(1+k)S) + T(1+k)(1+b(S-a)) = 0$

$-bCS^2 + T(1-ba)(1+k) = 0$

$S_{\text{opt}} = \sqrt{\frac{(1-ba)(1+k)T}{bC}}$

Problem 2.7

a. It is relatively easy to see how a clock skew of δ increases the clocking overhead in a traditionally clocked system by 2δ. Assume we have two latches and each of them is controlled by clock 1 and clock 2. Then, it is possible for clock 1 for the first latch to tick late by δ, and clock 2 for the second latch to tick early by δ. In this case, the cycle time is increased by 2δ.

b. It is a little harder to show that ΔT = largest $(P_{\max} - P_{\min})$ + oldC + 4δ. Once again, uncertainty in the arrival of data determines the clock cycle. Both the clock before and the clock after the stage are affected by the uncontrollable clock skew.

Clock 2 after the stage clocks the data out of the stage. This must tick before $P_{\min}$ and after $P_{\max}$ (plus setup and delay time). Because of its uncertainty, it must be moved d earlier than $P_{\min}$ and δ later than $P_{\max}$.

The clock 1 before the stage causes additional uncertainty in $P_{\min}$ and $P_{\max}$. (You can think of $P_{\min}$ decreasing and $P_{\max}$ increasing.) Thus, it must be moved an additional δ earlier and a δ later.

Problem 2.8

Delay $R \rightarrow$ALU$\rightarrow R = 16$ ns (assume it breaks into 5-5-6 ns)

Instruction or data cache data access = 8 ns

a. Compute S_{opt}, assuming b = .2

$T = 4 + 6 + 8 + 3 + 12 + 9 + 3 + 6 + 8 + 3 + 16 + 2 = 80$ ns

$k = .05$

$C = 4$ ns

$S_{\text{opt}} = \sqrt{\frac{(1-b)(1+K)T}{bC}} = \sqrt{\frac{.8 \times 1.05 \times 80}{.2 \times 4}} = 9.16$, round to 9

b. Partition into approximately S_{opt} segments

(i) First partitioning:

Stage	Actions	Time
IF1	PC→MAR (4), cache dir (6)	10
IF2	cache data (8)	8
ID1	data transfer to IR (3),decode (6)	9
ID2	decode (6)	6
AG	address generate (9)	9
DF1	address→MAR (3), cache dir (6)	9
DF2	cache data access (8)	8
EX1	data→ALU (3),execute (5)	8
EX2	execute (5)	5
PA	execute (6), put away (2)	8

$T_{\text{seg}} = 10$ ns

10 stages

$\Delta T = (1 + K)T_{\text{seg}} + C = (1.05)(10\,\text{ns}) + 4\,\text{ns} = 14.5$ ns

$T_{\text{inst}} = \text{stages} \times \Delta T = 145$ ns

(ii) Second partitioning:

Stage	Actions	Time
IF1	PC→MAR (4), cache dir (6)	10
IF2	cache data (8)	8
ID1	data transfer to IR (3),decode (6)	9
ID2	decode (6)	6
AG	address generate (9)	9
DF1	address→MAR (3), cache dir (6)	9
DF2	cache data access (8),data→ALU (3)	11
EX	execute (10)	10
PA	execute (6)put away (2)	8

$T_{\text{seg}} = 11$ ns

9 stages

$\Delta T = (1 + K)T_{\text{seg}} + C = (1.05)(11\,\text{ns}) + 4\,\text{ns} = 15.55$ ns

$T_{\text{inst}} = \text{stages} \times \Delta T = 139.95$ ns

(iii) Third partitioning:

Stage	Actions	Time
IF1	PC→MAR (4), cache dir (6)	10
IF2	cache data (8), data transfer (3)	11
ID	instruction decode (12)	12
AG	address generate (9), MAR (3)	12
DF1	cache dir (6)	6
DF2	cache data (8),data→ALU (3)	11
EX1	execute (10)	10
EX2	execute (6),put away (2)	8

$T_{\text{seg}} = 12\,\text{ns}$

8 stages

$\Delta T = (1+K)T_{\text{seg}} + C = (1.05)(12\,\text{ns}) + 4\,\text{ns} = 16.6\,\text{ns}$

$T_{\text{inst}} = \text{stages} \times \Delta T = 8(16.6) = 132.8\,\text{ns}$

(iv) Fourth partitioning:

Stage	Actions	Time
IF1	PC→MAR (4), cache dir +data (14)	18
ID	data transfer (3), decode (12)	15
AG	AG (9), MAR (3), cache dir (6)	18
DF2	cache data (8),data→ALU (3)	11
PA	execute (16),put away (2)	18

$T_{\text{seg}} = 18\,\text{ns}$

5 stages

$\Delta T = (1+K)T_{\text{seg}} + C = (1.05)(18\,\text{ns}) + 4\,\text{ns} = 22.9\,\text{ns}$

$T_{\text{inst}} = \text{stages} \times \Delta T = 5(22.9) = 114.5\,\text{ns}$

c. Performance

- $G = \frac{1}{1+(S-1)b} \times \frac{1}{(1+k)(T_{\text{seg}})+C}$
- $b = .2$

$G(T_{\text{seg}} = 10\,\text{ns}) = \frac{1}{1+9\times.2} \times \frac{1}{14.5\,\text{ns}} = 24.6$ MIPS

$G(T_{\text{seg}} = 11\,\text{ns}) = \frac{1}{1+8\times.2} \times \frac{1}{15.55\,\text{ns}} = 24.7$ MIPS

$G(T_{\text{seg}} = 12\,\text{ns}) = \frac{1}{1+7\times.2} \times \frac{1}{16.6\,\text{ns}} = 25.1$ MIPS

$G(T_{\text{seg}} = 18\,\text{ns}) = \frac{1}{1+4\times.2} \times \frac{1}{22.9\,\text{ns}} = 24.3$ MIPS

Of the pipelines considered here, with the above assumptions, the 8-stage ($T_{\text{seg}} = 12\,\text{ns}$) pipeline has the best performance. However, the 5-stage pipeline also has decent performance and would be easier to implement.

Problem 2.9

$b = .25$, $C = 2\,\text{ns}$, $k = 0$.

Function	Delay
A	6
B	8
C	3
D	7
E	9
F	5

$T = 6 + 8 + 3 + 7 + 9 + 5 = 38$.

a. Optimum number of pipeline segments, ignoring quantization

$S_{\text{opt}} = \sqrt{\frac{(1-.25)(1+0)38}{.25 \times 2}} = 7.55$, round down to 7.

b. Cycle time

Since $S_{\text{opt}} = 7$, one function unit can be split into 2 stages. Split function unit E since it has the longest delay.

$\Delta T = 8 + 2 = 10\,\text{ns}$

c. Performance

$G = \frac{1}{1+(S-1)b} \times \frac{1}{T_{\text{seg}}+C} = \frac{1}{1+6\times.25}\frac{1}{8+2\text{ ns}} = 40$ MIPS.

Performance $= \frac{1}{1+(7-1)0.25} = 0.4$ inst/cycle.

d. Can you find better performance?

Assume that each function unit cannot be merged with neighboring function units.

A 6
B 8 Can be divided into 4, 4 ns stages
C 3
D 7 Can be divided into 3.5, 3.5 ns stages
E 9 Can be divided into 4.5, 4.5 ns stages
F 5

(i) Do not split any function units.
$T_{\text{seg}} = 9\,\text{ns}$
$S = 6$
$G = \frac{1}{1+5\times.25}\frac{1}{9+2\text{ ns}} = 40.4$ MIPS

(ii) Split function units B and E.
$T_{\text{seg}} = 7\,\text{ns}$
$S = 8$
$G = \frac{1}{1+7\times.25}\frac{1}{7+2\text{ ns}} = 40.4$ MIPS

(iii) Split function units B, D and E.
$T_{\text{seg}} = 6\,\text{ns}$
$S = 9$
$G = \frac{1}{1+8\times.25}\frac{1}{6+2\text{ ns}} = 41.7$ MIPS
$T_{\text{seg}} = 6\,\text{ns}$ gives the best performance.

e. The adjusted cycle time $= 10 + 2(1) = 12\,\text{ns}$.

Problem 2.10

a. Without aspect-mismatch adjustment,

- Bits per line = 256 bits/line
- Total number of lines = 32KB/32B = 1024 lines
- Tag bits = 20

Data bits = $32\,\text{KB} \times 8$ bits per byte $= 32 \times 1024 \times 8 = 262,144$ bits

Tag bits = 1024 lines $\times$ 20 bits per byte = 20480 bits

Cache size = $195 + .6(262,144 + 20,480) = 169,769.4$ **rbe**

b. With aspect-mismatch adjustment

With aspect-ratio mismatch, 10% of our area is wasted, so the actual area which is taken up is only 90% of the total area. We divide by .9 to find the corrected area,

$$\text{Area} = \frac{169,769.4}{.9} = 188,632.7 \textbf{ rbe}$$

Problem 2.11

$A = (1.4 \text{ cm})^2 = 1.96 \text{ cm}^2$

a. 6-inch wafer = 15.24-cm wafer

$N = \frac{\pi}{4A}(d - \sqrt{A})^2 = \frac{\pi}{4 \times 1.4^2}(15.24 - 1.4)^2 \approx 76$

Year	ρ_D	Y	N_G	Wafer cost	Effective die cost
1	1.5	.053	4	\$5000	\$1250
2	1.325	.074	5	\$4625	\$925
3	1.15	.105	7	\$4250	\$607
4	.975	.148	11	\$3875	\$352.30
5	.8	.21	16	\$3500	\$218.70

b. 8-inch wafer = 20.32-cm wafer

$N = \frac{\pi}{4A}(d - \sqrt{A})^2 = \frac{\pi}{4 \times 1.4^2}(20.32 - 1.4)^2 \approx 143$

Year	ρ_D	Y	N_G	Wafer cost	Effective die cost
1	1.5	.053	7	\$10000	\$1428.60
2	1.325	.074	10	\$9125	\$912.50
3	1.15	.105	15	\$8250	\$550.00
4	.975	.148	21	\$7375	\$351.20
5	.8	.21	30	\$6500	\$216.70

It is more effective to use an 8-inch wafer since the effective die cost is lower for the last 4 years.

Problem 2.12

$A = -\frac{\ln(\text{yield})}{\rho_D} = -\frac{\ln(.1)}{1} = 2.3 \text{ cm}^2$

20% of die area is reserved for pads, etc., so we have left = 1.84 cm^2

10% of die area is reserved for sense amps, etc., so we have left = 1.61 cm^2

$f = 1\mu m = 2\lambda$

Cell size = $135\lambda^2 = 33.75\,\mu m^2$

Capacity = $\frac{1.61}{33.75\times10^{-8}} = 4.770 \times 10^6$ bits

Of this, we can only use 4 Mb = 2^{22} = 4,194,304 bits

New cell area = 1.61 $cm^2 \times \frac{4194304}{4.770\times10^6} = 1.42$ cm^2 which is 70% of die area.

Total die area = 1.42 $cm^2 \times \frac{100}{70} = 2.03$ cm^2

New Yield = $e^{-1\times2.03} = 13.1$ %

(We can also assume that the 10% overhead for sense amps, etc. is calculated from the total cell area. In this case, the capacity is 4.9478×10^6 bits. The new total die area = 1.95 cm^2 and the new yield = 14.2 %)

Problem 2.13

$f = 1\mu m = 2\lambda$

Cell size = $135\lambda^2 = 33.75\,\mu m^2$

Total cell area = $1024 \times 1024 \times 33.75\,\mu m^2 = 0.354$ cm^2

Total die area = $0.354 \times \frac{100}{70} = 0.506$ cm^2

Yield = $e^{-1\times0.506} = 0.603$

$N = \frac{\pi}{4A}(d - \sqrt{A})^2 = \frac{\pi}{4\times0.506}(21 - \sqrt{0.506})^2 \approx 638$

$N_G = 638 \times 0.603 \approx 384$

Cost of a $1^M \times 1^b$ die = $\frac{\$5000}{384} \approx \$13$

(Can also assume that the 10% overhead for sense amps, etc. is calculated from the total cell area. In this case, the total die area = 0.487 cm^2 and the cost ≈ $12.30)

Problem 2.14

$A = 2.3$ cm^2

Yield (.5 defects/cm^2) = $e^{-\rho_D \times A} = e^{-.5\times2.3} = 31.7\%$

Yield (1 defects/cm^2) = $e^{-\rho_D \times A} = e^{-1\times2.3} = 10.0\%$

Die (15 cm) = $\frac{\pi}{4A}(d - \sqrt{A})^2 = \frac{\pi}{4\times2.3} \times (15 - 1.52)^2 = 62$

Die (20 cm) = $\frac{\pi}{4A}(d - \sqrt{A})^2 = \frac{\pi}{4\times2.3} \times (20 - 1.52)^2 = 116$

$N_G(d = 15, \rho_D = .5) = 62 \times .317 = 19.7$, cost/good die = $\frac{\$5000}{19.7} = \$254$

$N_G(d = 15, \rho_D = 1) = 62 \times .100 = 6.2$, cost/good die = $\frac{\$5000}{6.2} = \$806$

$N_G(d = 20, \rho_D = .5) = 116 \times .317 = 36.8$, cost/good die = $\frac{\$8000}{36.8} = \$217$

$N_G(d = 20, \rho_D = 1) = 116 \times .100 = 11.6$, cost/good die = $\frac{\$8000}{11.6} = \$690$

For either defect density, the larger wafer is more cost effective for the given die size and wafer costs.

Problem 2.15

a. Using one chip:

$A = 280 \text{ mm}^2 \times \frac{10}{8} = 350 \text{ mm}^2 = 3.5 \text{ cm}^2$

$N = \frac{\pi}{4 \times 3.5}(15 - \sqrt{3.5})^2 = 38.68$

$Y = e^{-\rho_D A} = e^{-1 \times 3.5} = .03$

$N_G = N \cdot Y = 38.68 \times .03 = 1.16 \approx 1$

Effective cost = \$4000

Cost for one processor = \$4000 + \$50 = \$4050

b. Using two chips:

$A = 1.75 \text{ cm}^2$

$N = \frac{\pi}{4 \times 1.75}(15 - \sqrt{1.75})^2 = 83.95$

$Y = e^{-\rho_D A} = e^{-1 \times 1.75} = .174$

$N_G = N \cdot Y = 83.95 \times .174 = 14.6 \approx 14$

Effective cost = $\frac{\$4000}{14} = \$285.70$

Cost for one processor = $\$285.70 \times 2 + \$50 \times 2 = \$671.40$

c. Using four chips:

$A = .875 \text{ cm}^2$

$N = \frac{\pi}{4 \times .875}(15 - \sqrt{.875})^2 = 177.56$

$Y = e^{-\rho_D A} = e^{-1 \times .875} = .417$

$N_G = N \cdot Y = 177.56 \times .417 \approx 74$

Effective cost = $\frac{\$4000}{74} = \$54.05$

Cost for one processor = $\$54.05 \times 4 + \$50 \times 4 = \$416.20$

d. Using eight chips:

$A = .4375 \text{ cm}^2$

$N = \frac{\pi}{4 \times .4375}(15 - \sqrt{.4375})^2 = 369$

$Y = e^{-\rho_D A} = e^{-1 \times .4375} = .646$

$N_G = N \cdot Y = 369 \times .646 \approx 238$

Effective cost = $\frac{\$4000}{238} = \$16.80$

Cost for one processor = $\$16.80 \times 8 + \$50 \times 8 = \$534.40$

Using four chips shows the smallest cost per processor.

Chapter 3. Data

Problem 3.1

From Tables 3.2, 3.3, and 3.4:

LS: total bytes $= 200 * 4B = 800B$
R/M scientific: total bytes $= 180 * 3.2B = 576B$, $576/800 = .72$
R/M commercial: total bytes $= 180 * 3.8B = 684B$, $684/800 = .86$
R+M scientific: total bytes $= 120 * 4B = 480B$, $480/800 = .60$
R+M commercial: total bytes $= 120 * 3.8B = 456B$, $456/800 = .57$

Problem 3.2

From Table 3.3: R/M $\rightarrow$ 180 total instructions. From Table 3.15: 1 of these is LDM, 1 of these is STM. Data from section 3.3.1: LDM $\rightarrow$ 3 registers $\rightarrow$ 2 cycles, STM $\rightarrow$ 4 registers $\rightarrow$ 3 cycles. Base CPI = 1.

CPI with LDM + STM $= ((178 * 1) + (1 * 2) + (1 * 3))/180 = 1.0167$.

$1.0167/1 \rightarrow 1.7\%$ slowdown.

Problem 3.5

Using data from Tables 3.4 and 3.14:

Frequencies:

FP total = 12%
Add/Sub = 54%
Mul = 43%
Div = 3%

Assuming that the extra cycles cannot be overlapped, then the penalties can be calculated as:

Add/Sub $= 0.54 \times 1$
Mul $= 0.43 \times 5$
Div $= 0.03 \times 11$

Total excess CPI $= (0.54 + 2.15 + 0.33) \times .12 = 0.362$

Problem 3.6

(See problem 3.5 solution for relative frequencies.) Extra floating point cycles can be overlapped. Penalty occurs only due to data dependency on the results. Using data from Table 3.19:

a. Add/Sub $= 0.403 \times 0.54 \times 1 = 0.218$ cycles per fpop

b. Mul $= (0.403 \times 5 + 0.147 \times 4 + 0.036 \times 3 + 0.049 \times 2 + 0.067 \times 1) \times 0.43 = 1.24$ cycles per fpop

c. Div $= (0.404 \times 11 + 0.147 \times 10 + 0.036 \times 9 + 0.049 \times 8 + 0.067 \times 7 + 0.017 \times 6 + 0.032 \times 5 + 0.025 \times 4 + 0.028 \times 3 + 0.022 \times 2 + 0.035 \times 1) \times .03 = 0.229$ cycles per fpop

Total excess CPI $= (0.218 + 1.24 + 0.229) \times .12 = 0.202$

Problem 3.10

From Table 3.4, fixed point operations account for 16 instructions per 100 HLL operations. From Table 3.14, fixed point multiply accounts for 15% of all fixed point instructions. Number of fixed point multiply operations per 100 HLL is $.15 \times 16 = 2.4$.

When implementing multiplication by shifts and adds, each fixed point multiply requires on average 13 shifts, 5 adds, and 22 branches. Total operations/multiply = 40.

Extra shifts/100 HLL = $13 \times 2.4 = 31.2$

Extra adds/100 HLL = $5 \times 2.4 = 12$

Extra branches/100 HLL = $22 \times 2.4 = 52.8$

Table 1: Instructions per 100 HLL ops

Move		107
Branch	$= 26 + 52.8 =$	78.8
Floating point	$= 24 - 2.4 =$	21.6
Fixed point	$= 16 + 12 =$	28
Shifts	$= 27 + 31.2 =$	58.2
Total		293.6

Expected instructions per 100 HLL operations = 293.6.

Problem 3.13

a. 256/256

$19\% + 42\% = 61\%$.

b. 384/128

For 384: capture rate $= (384 - 256) * ((22 - 19)/(512 - 256)) + 19 = 20.5\%$.

$20.5\% + 39\% = 59.5\%$.

c. 448/64

For 448: capture rate $= (448 - 256) * ((22 - 19)/(512 - 256)) + 19 = 21.25\%$.

$21.25\% + 35\% = 56.25\%c)448/64$.

d. 480/32

For 480: capture rate $= (480 - 256) * ((22 - 19)/(512 - 256)) + 19 = 21.625\%$.

$21.625\% + 32\% = 53.62\%$.

e. 496/16

For 496: capture rate $= (496 - 256) * ((22 - 19)/(512 - 256)) + 19 = 21.8125\%$.

$21.81\% + 32\% = 53.81\%$.

Problem 3.14

From Table 3.10, 80% of branches are conditional. From Table 3.4, 26 branch instructions per 100 HLL instructions. From Table 3.3, 180 total instructions per 100 HLL instructions. For conditional branch: if mean interval > 1, branch takes 1 cycle. If mean interval < 1, branch takes 2 cycles.

Assume all branches take 1 cycle: CPI $= 1$.

Assume all conditional branches take 2 cycles:

CPI $= ((26 * 0.8) * 2 + (180 - (26 * 0.8) * 1))/180 = 1.12$.

Problem 3.15

```
 1) ADD.W R7, R7, 4
 2) LD.W R1, 0(R7)
3) MUL.W R2, R1, R1
4) ADD.W R3, R3, R2
 5) LD.W R4, 2(R7)
6) SUB.W R5, R2, R3
7) ADD.W R5, R5, R4
```

Address interlocks have a 4 cycle delay, and execution interlocks have a 2 cycle delay, all when interlocked by the preceding instruction.

I2 has an address interlock from I1 through R7. This requires 4 extra cycles.

I3 has an execution interlock on I2 through R1. This requires 2 extra cycles.

I4 has an execution interlock on I3 through R2. This requires 2 extra cycles.

I5 requires R7 from I1, but due to a previous interlock, this result is already available.

I6 has an execution interlock on I4 through R3. Because I6 is 2 instructions away, this requires 1 extra cycle. I6 does not interlock on R2 because of a previous interlock.

I7 has an execution interlock on I6 through R5. This requires 2 extra cycles.

Total cycles (issue) $= 7$ cycles

Total extra cycles $= 4 + 2 + 2 + 1 + 2 = 11$ cycles

Total time to execute $= 18$ cycles.

Chapter 4. Pipelined Processor Design

Problem 4.1

Additional assumptions beyond those stated in the question:

- For Amdahl V-8 and MIPS R2000, the operations have to be aligned to their respective phases (e.g., the IF must start at phase 2 and the Decode must occur at phase 2.)
- The instruction setting the CC is immediately before the branch instruction for the analysis.

For this problem, we need to calculate the penalty cycles for executing

a. Branch instruction.

b. Conditional branch instruction that branched in-line.

c. Conditional branch instruction that branched to the target.

d. The effect of address dependencies.

For branch instruction, the instruction fetch of the instruction following the branch instruction is delayed since it is fetched during the data fetch of the branch instruction. For BC-inline, the decode is delayed until the conditional code is set. For BC-target, we must consider both the instruction immediately following BC-target (BC-target+1) as well as the instruction following it (BC-target+2). The instruction fetch for BC-target+1 is delay till the data fetch of the BC-target instruction and the instruction fetch for the BC-target+2 is delayed till the condition code is set. For the EX/LD instruction sequence, the address calculation of the LD instruction is delayed until the end of the execution cycles. For the LD/LD instruction, the address calculation is delayed until the end of data fetch cycles.

The equation for CPI is:

$$\text{CPI} = \text{BaseCPI} + \text{RunOnCPI} + \text{BrCPI} + \text{BcCPI} + \text{ExLdCPI} + \text{LdLdCPI}.$$

$$\begin{aligned}
\text{BaseCPI} &= 1. \\
\text{RunOnCPI} &= 0.6 \text{ (from study 4.3).} \\
\text{BrCPI} &= (0.05) * \text{BrPenalty}. \\
\text{BcCPI} &= (0.15) * (0.5) * (\text{BC-inlinePenalty} + \text{BC-targetPenalty}). \\
\text{ExLdCPI} &= (0.015) * \text{ExLdPenalty}. \\
\text{LdLdCPI} &= (0.04) * \text{LdLdPenalty}.
\end{aligned}$$

Base Pipeline Template

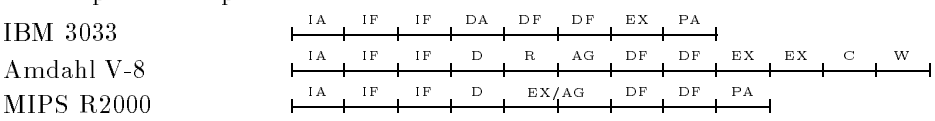


Table 2: Summary of IBM 3033

Instruction	Penalty	Execution Frequency
Br	2	5%
BC-inline	2	7.5%
BC-target	3	7.5%
EX/LD	1.54	1.5%
LD/LD	0.29	4.0%
CPI	2.2	

Instr	IBM 3033
1	Set CC
2	Branch
3	BR+1
3	BC-inline+1
3	BC-target+1
4	BC-target+2
3	Ex/Ld
3	Ld/Ld

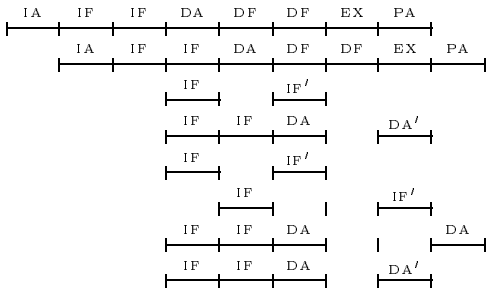


Table 3: Summary of Amdahl V-8

Instruction	Penalty	Execution Frequency
Br	4	5%
BC-inline	4	7.5%
BC-target	4	7.5%
EX/LD	2.17	1.5%
LD/LD	0.28	4.0%
CPI	2.54	

Instr	Amdahl V-8 (Phase)
1	
2	Instr
3	BR+1
3	BC-inline+1
3	BC-target+1
4	BC-target+2
3	Ex/Ld
3	Ld/Ld

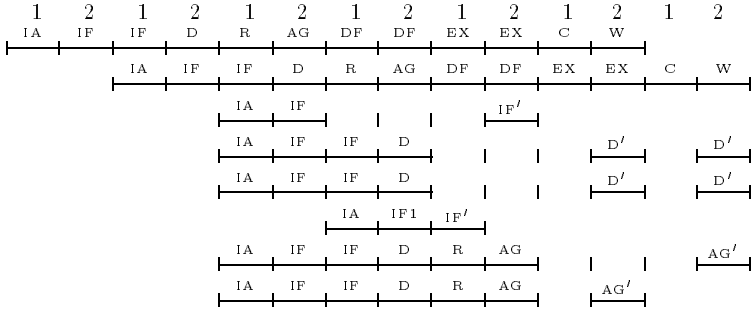
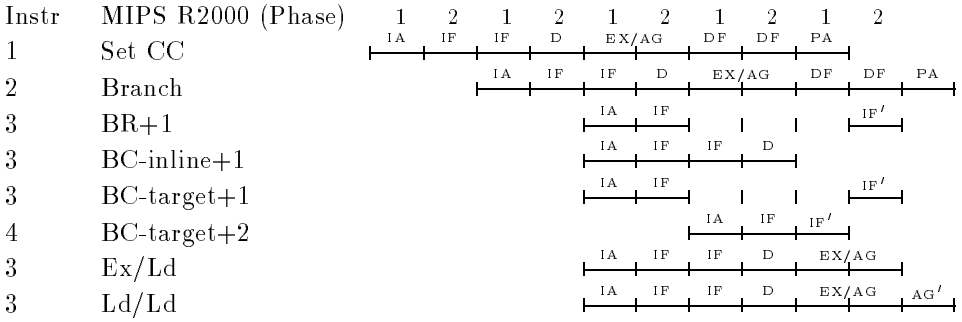


Table 4: Summary of MIPS R2000

Instruction	Penalty	Execution Frequency
Br	4	5%
BC-inline	2	7.5%
BC-target	3	7.5%
EX/LD	1.54	1.5%
LD/LD	0.29	4%
CPI	1.41	



Problem 4.3

For this problem, we need to calculate the following based on data from Chapter 3. Use pipeline template from problem 4.1.

a. EX/LD and LD/LD frequencies.

b. BR, BC-inline, BC target frequencies.

c. Calculate weighted penalty for EX/LD and LD/LD based on Tables 3.19 and 3.20.

a. We need to approximate the possibility of a Load instruction following a instruction that generates its data address. We use the data from Table 3.15.

   The occurrence of EX type instruction followed by a LOAD is $0.5 * 0.4 = 0.2$ for the R/M machines.

   The occurrence of a LOAD followed by a LOAD is 0.4 * 0.4 = 0.16 for the R/M machines.

   The occurrence of EX type instruction followed by a LOAD is $0.5 * 0.5 = 0.25$ for the L/S machines.

   The occurrence of a LOAD followed by a LOAD is $0.5 * 0.5 = 0.25$ for the L/S machines.

b. We use Table 3.10 for this calculation. To calculate the frequency of BR, BC-inline, and BC-target, we need to perform the following calculations. We also use R/M data for the IBM 3033 and AmdahlV-8 and L/S data for the MIPS R2000.

$$\begin{aligned} \text{BrFreq} &= \text{Type1Freq} * \text{AGFreq} * \text{UnconditionalFreq} = 1.8\% \\ \text{BC-inline} &= \text{Type1Freq} * \text{AGFreq} * \text{ConditionalFreq} * \text{InlineFreq} + \text{Type2Freq} * \text{InLineFreq} \\ &= 3.4\% \end{aligned}$$

$$\begin{aligned}\text{BC-target} &= \text{Type1Freq} * \text{AGFreq} * \text{ConditionalFreq} * \text{TargetFreq} \\ &\quad + \text{Type2Freq} * \text{TargetFreq} + \text{Type3Freq} \\ &= 7.9\end{aligned}$$

c. The weighted penalty is simply the sum of Prob (distance $= n$) $*$ penalty (distance $= n$), where n is the number of penalty cycles to 1.

Table 5: Summary of IBM 3033

Instruction	Penalty	Execution Frequency
BR	2	1.8%
BC-inline	2	3.4%
BC-target	3	7.9%
EX/LD	1.54	20%
LD/LD	0.29	16%
CPI	1.7	

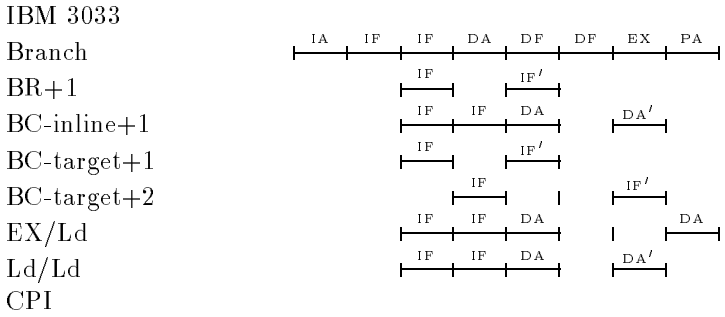


Table 6: Summary of Amdahl V-8

Instruction	Penalty	Execution Frequency
BR	4	1.8%
BC-inline	4	3.4%
BC-target	4	7.9%
EX/LD	2.17	20%
LD/LD	0.28	16%
CPI	2.0	

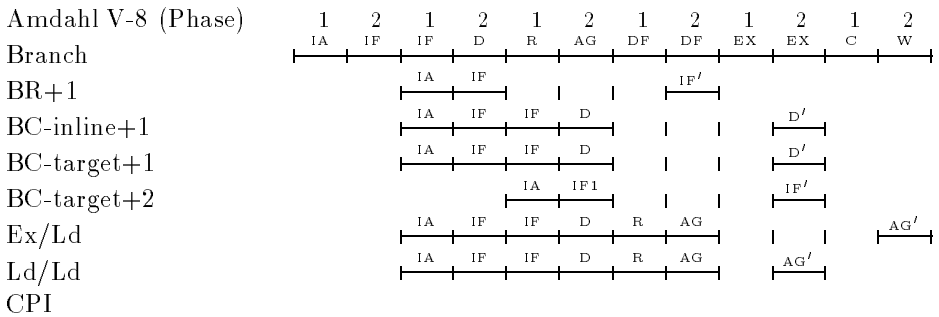
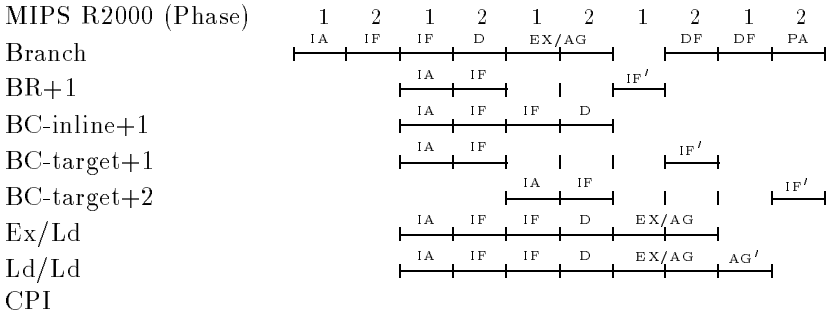


Table 7: Summary of MIPS R2000

Instruction	Penalty	Execution Frequency
BR	4	1.55%
BC-inline	2	3.0%
BC-target	3	6.8%
EX/LD	1.54	25%
LD/LD	0.29	25%
CPI	1.41	



Problem 4.5

This problem follows the same steps described in Study 4.4. The difference between early CC setting and delay branch is:

- Early CC hides the time the CPU is stalled waiting for the condition code to set. As a result, early CC setting makes it possible to hide the delay of BC-inline completely, but it can only hide BC-target up to the BR delay.

- Delay branch hides both the time the CPU is stalled waiting for condition code and the time to fetch the target instruction by inserting the instruction after the BC itself. As a result, it is possible to hide both BC-inline and BC-target delays completely.

To compute the excess CPI due to branch instructions for these two schemes, we follow the procedure of problem 4.1, where

BcCPI = $(0.15) * (0.5) *$ (BC-inline penalty + BC-target penalty).

The effectiveness of either scheme depends on the compiler's ability to insert instructions for both schemes. The results of the analysis are shown in Tables 8–13.

(a) IBM 3033

BaseCpi	1
Run-on	0.6

Table 8: Using Early CC setting for MIPS R2000

BC	$n=0$	$n=1$	$n=2$	$n=3$	$n=4$
BC-inline penalty	4	3	2	1	0
BC-target penalty	4	4	4	4	4
ExcessCPI	0.6	0.525	0.45	0.375	0.3

Table 9: Using delay branch for MIPS R2000

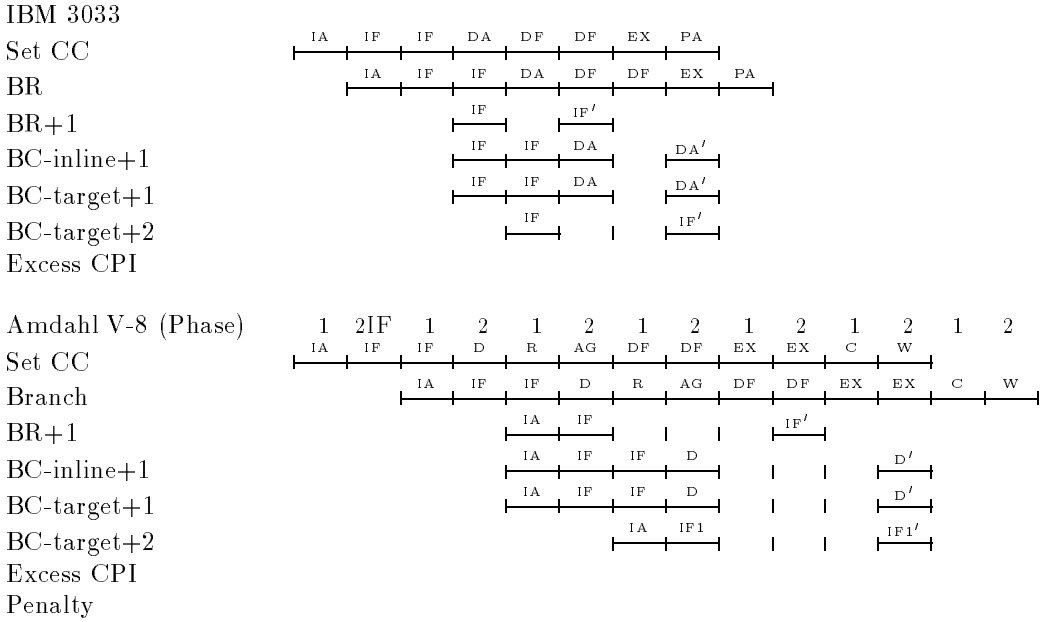
BC	$n=0$	$n=1$	$n=2$	$n=3$	$n=4$
BC-inline penalty	4	3	2	1	0
BC-target penalty	4	3	2	1	0
ExcessCPI	0.6	0.45	0.3	0.15	0

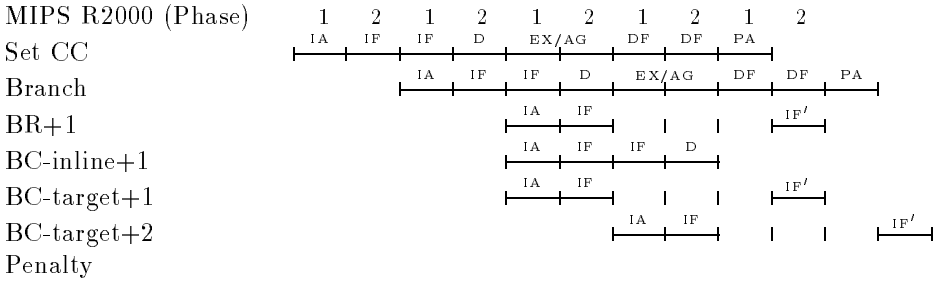
Table 10: Using Early CC setting for IBM 3033

BC	$n=0$	$n=1$	$n=2$	$n=3$
BC-inline penalty	2	1	0	0
BC-target penalty	3	2	2	2
ExcessCPI	0.375	0.225	0.15	0.15

Table 11: Using delay branch for IBM 3033

BC	$n=0$	$n=1$	$n=2$	$n=3$
BC-inline penalty	2	1	0	0
BC-target penalty	3	2	1	0
ExcessCPI	0.375	0.225	0.075	0





(b) Amdahl V-8

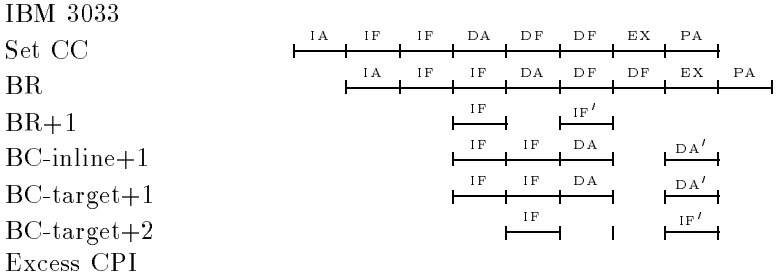
BaseCpi	1
Run-on	0.6

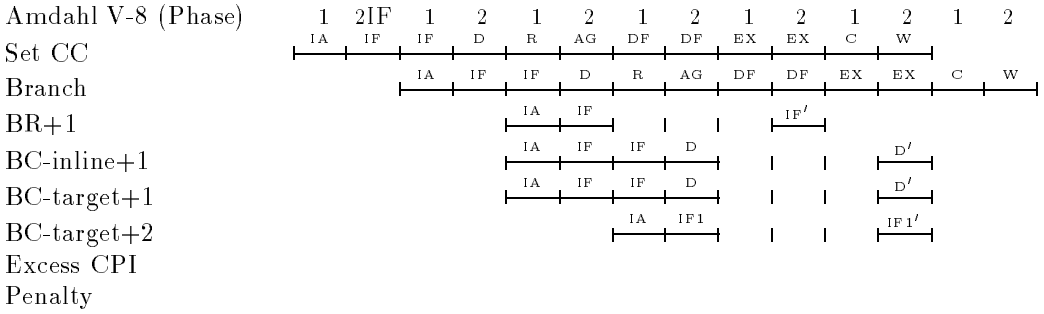
Table 12: Using Early CC setting for Amdahl-V8

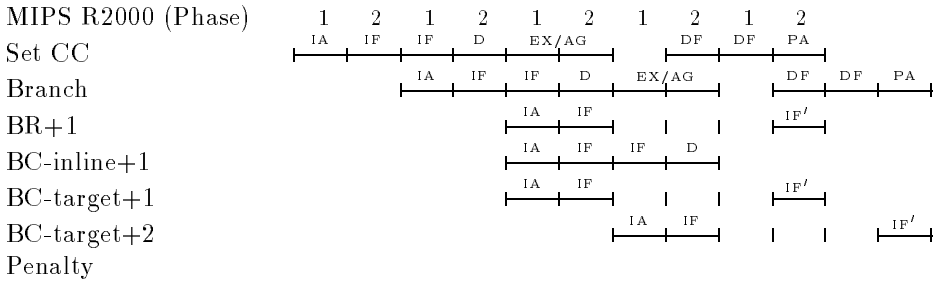
BC	$n=0$	$n=1$	$n=2$	$n=3$	$n=4$
BC-inline penalty	4	3	2	1	0
BC-target penalty	4	4	4	4	4
ExcessCPI	0.6	0.525	0.45	0.375	0.3

Table 13: Using delay branch for Amdahl-V8

BC	$n=0$	$n=1$	$n=2$	$n=3$	$n=4$
BC-inline penalty	4	3	2	1	0
BC-target penalty	4	3	2	1	0
ExcessCPI	0.6	0.45	0.3	0.15	0







Problem 4.7

From section 4.4.5, we have the following equation:

$$p = 1 + \left\lceil \frac{1}{m} \times \frac{\text{IF access time (cycles)}}{\text{instructions/IF}} \right\rceil,$$

where p is the number of in-line buffer words, and an instruction is decoded every m cycles.

a. IBM 3033

$m = 1$

IF access time = 2 cycles (from Figure 4.3)

Average instruction length = $\frac{3.2B+3.8B}{2} = 3.5B$ (from Table 3.2)

Instructions/IF = $\frac{\text{path width } w}{\text{instruction length}} = \frac{4}{3.5} = 1.14$ for $w = 4$

Instructions/IF = $\frac{8}{3.5} = 2.29$ for $w = 8$

$$p(w = 4B) = 1 + \left\lceil \frac{1}{1} \times \frac{2}{1.14} \right\rceil = 3$$

$$p(w = 8B) = 1 + \left\lceil \frac{1}{1} \times \frac{2}{2.29} \right\rceil = 2$$

We assume that we have branch prediction which may sometimes predict the target path. Therefore both the primary and target paths must be the same (minimum) size in order to avoid runout.

b. Amdahl V-8

The Amdahl V-8 is similar to the 3033 (executing the same instruction set), but it only decodes an instruction every other cycle. From Figure 4.4, we have $m = 2$, IF access time = 2 cycles.

$$p(w = 4B) = 1 + \left\lceil \frac{1}{2} \times \frac{2}{1.14} \right\rceil = 2$$

$$p(w = 8B) = 1 + \left\lceil \frac{1}{2} \times \frac{2}{2.29} \right\rceil = 2$$

Once again, we assume branch prediction, so both instruction buffers must be the same size.

c. MIPS R2000

$m = 2$ (Figure 4.5)

IF access time = 2 cycles (Figure 4.5)

Average instruction length = 4B (from Table 3.2)

$$p(w = 4B) = 1 + \left\lceil \frac{1}{2} \times \frac{2}{1} \right\rceil = 2$$
$$p(w = 8B) = 1 + \left\lceil \frac{1}{2} \times \frac{2}{2} \right\rceil = 2$$

Assuming we can sometimes predict taken, both buffers must be the same size. However, if we assume that the R2000 always predicted the in-line path (relying perhaps on delayed branches to reduce branch latency), then the target buffer could perhaps be as small as one entry (as per section 4.4.5.)

Problem 4.8

We use Chebyshev's inequality.

a. Without knowing the variance, we have to use Bound 1:

$$\begin{aligned} \text{Prob (overflow)} &= \text{Prob}\, q > BF = \text{Prob}\, q \geq BF + 1 \rightarrow p \geq \frac{Q}{BF+1} \\ Q &= \text{mean number of requests} = 2 \\ BF &= \text{buffer size} = 4 \\ p &\geq \frac{Q}{BF+1}, \text{ so } p \geq 2/5,\ p \geq 40\% \end{aligned}$$

b. Knowing the variance, we can compare Bound 2 and take the minimum:

s^2 = variance = 0.5

$p \geq \frac{s^2}{(BF+1-Q)^2}$

$p \geq .5/9 = 5.55\%$

This upper bound is lower than the value determined in the first part, so we can say that $\text{Prob}\, q > BF \geq 5.55\%$.

Chapter 5. Cache Memory

Problem 5.1

$$\overbrace{A_{23}\ldots A_{15}}^{\text{tag}}\overbrace{A_{14}\ldots A_6}^{\text{index}}\overbrace{A_5\ldots A_3}^{W/L}\overbrace{A_2\ldots A_0}^{B/W}$$

a. Address bits unaffected by translations

$A_{14} - A_0$

b. Bits to address the cache directories

$A_{14} - A_6$

c. Address bits compared to entries in the cache directory

$A_{23} - A_{15}$

d. Address bits appended to (b) to address cache array

$A_5 - A_3$

Problem 5.3

The effective miss rate for cache in 5.1:

$$\begin{aligned}\text{DTMR} &= .007 \text{ (128KB, 64B/L, Fully Assoc, Table A.1)}\\ \text{Adjustment factor} &= 1.05 \text{ (64B/L, 4-way, Table A.4)}\\ \text{Miss rate} &= .007 \times 1.05 = .00735 = .735\%\end{aligned}$$

Problem 5.6

Assume the cache of problem 5.1 with 16 B/L.

a. $Q = 20,000$

Miss rate from Figure A.9: .01 = .0947

Miss rate = .0508

b. The optimal cache size is the smallest size with a low miss rate. From Figure 5.26, the $Q = 20,000$ line flattens out around 32KB. From Table A.9 (which tabulates the same data), we see that the miss rate is essentially unchanged for caches of either size 32KB or 64KB and above.

Problem 5.7

a. L1 cache: 8 KB, 4-way, 16B/L

DTMR: from Table A.1, .075 = 7.5%

Adjustment for 4W associativity: from Table A.4, 1.04

Miss rate = $.075 \times 1.04 = .078 = 7.8\%$

L2 cache: 64KB, direct mapped, 64B/L

DTMR: from Table A.1, .011 = 1.1%

Adjustment for direct mapped cache: from Table A.4, 1.57

Miss rate = $.011 \times 1.57 = .0173 = 1.73\%$

b. Expected CPI loss due to cache misses

Refs/I = 1.5

We need to determine how many of the references are reads and how many are writes. For this problem, we consider L/S machines in a scientific environment. L/S machines typically have one instruction reference per instruction (as in Table 3.2, where we see that all instructions are 4 bytes long, which would be one reference on a 4-byte memory path.) This leaves .5 remaining data references. Conveniently enough, Table 5.7 shows .31 data reads and .19 data writes per instruction for an L/S machine in a scientific environment.

I-Reads/I = 1.0

D-Reads/I = .31

D-Writes/I = .19

Miss penalty (MP) for L1 = 3 cycles

MP for L1 + MP for L2 = 10 cycles

MP for L2 = $10 - 3 = 7$ cycles

We make the assumption of statistical inclusion, that is, if we miss in L2 we also miss in L1, so we can use the DTMR (solo) as global miss rates. We can make this assumption because L2 is significantly larger than L1.

Note that since the L1 cache is WTNWA, we need only consider read misses. Assume that writes can be considered L1 cache hits when calculating miss penalties if they miss in L1 but hit in L2.

Assume 30 % of lines in L2 integrated cache are dirty (from Section 5.6).

Expected CPI loss due to cache miss:

$$\begin{aligned} &= (\text{I-reads} + \text{D-reads})/I \times MR_{L1} \times MP_{L1} + \\ &\quad (\text{I-reads} + \text{D-reads} + \text{D-writes})/I \times MR_{L2} \times MP_{L2} \times (1 + w) \\ &= (1.31) \times .078 \times 3 + (1.5) \times (.0173) \times 7 \times 1.3 = .54 \text{ CPI} \end{aligned}$$

c. Will all lines in L1 always reside in L2?

No, because L1 is 4-way associative and L2 is direct mapped. Considering the criteria from 5.12.1:

(i) Number of L2 sets $\geq$ Number of L1 sets

Number of L2 sets $= \frac{64KB}{64B/L} = 1024$ sets

Number of L1 sets $= \frac{8KB}{\text{4-way associative} \times 16B/L} = 128$ sets

$1024 \geq 128$, so this criterion holds.

(ii) L2 assoc $\geq$ L1 assoc

L2 assoc = 1 < L1 Assoc = 4

This condition does not hold, so logical inclusion does not apply.

Problem 5.8

L1: 4KB direct-mapped, WTNWA

L2: 8KB direct-mapped, CBWA

16BL for both caches

a. Yes
   L2 sets (512) > L1 sets (256)
   L2 associativity (1) = L1 associativity (1)
   Also, since L1 is write through, L2 has the updated data of L1.

b. No
   L2 sets (128) < L1 sets (256)

c. No
   L2 associativity < L1 associativity
   Also, since L1 is copyback, L2 may not have the updated data of L1.

Problem 5.9

Assume there is statistical inclusion.

$MR_{L1} = 10\%$

$MR_{L2} = 2\%$

$MP_{L1} = 3$ cycles

$MP_{L1+L2} = 10$ cycles

$MP_{L2} = 10 - 3 = 7$ cycles

1 ref/I

$$\begin{aligned} \text{Excess CPI due to cache misses} &= 1 \text{ ref/I} \times (MR_{L1} \times MP_{L1} + MR_{L2} \times MP_{L2}) \\ &= .1 \times 3 + .02 \times 7 = .44 \text{ CPI} \end{aligned}$$

Problem 5.13

a. 12KB, 3-way set associative
   $32B = 2^5 B$ line size. There are 5 bits for line address.
   There are $\frac{12KB}{3 \times 32B} = 128 = 2^7$ sets. There are 7 bits for index.
   There are $26 - 7 - 5 = 14$ bits for tag.

$$\text{Address: } \overbrace{A_{25} \ldots A_{12}}^{\text{tag}} \overbrace{A_{11} \ldots A_5}^{\text{index}} \overbrace{A_4 \ldots A_0}^{\text{line}}$$

$$\text{Directory entry: } \overbrace{14bits}^{\text{tag}_1} \overbrace{14bits}^{\text{tag}_2} \overbrace{14bits}^{\text{tag}_3}$$

b. DTMR for 8KB: 0.05 (Table A.1)

DTMR for 16KB: 0.035

Interpolate: DTMR for 12KB (fully associative): $\frac{0.05+0.035}{2} = 0.0425$

Adjustment for 2-way set associativity (8KB): 1.13 (Table A.4)

Adjustment for 2-way set associativity (16KB): 1.13

Adjustment for 4-way set associativity (8KB): 1.035

Adjustment for 4-way set associativity (16KB): 1.035

Interpolate:

Adjustment for 2-way set associativity (12KB): 1.13

Adjustment for 4-way set associativity (12KB): 1.035

Adjustment for 3-way set associativity (12KB): $\frac{1.13+1.035}{2} = 1.0825$

DTMR for 3-way set associative cache (12KB): $1.0825 \times 0.0425 = 4.6\%$

c. Since 12Kb is not a power of 2, the usual way of dividing the address bits to access the cache will not work.

Observe that a 8KB direct-mapped cache uses the least significant 13 bits of address to access the cache and a 16KB direct-mapped cache uses the least significant 14 bits. For a 12Kb direct-mapped cache, when bits [13:12] are 00, 01 or 10, this maps well into the cache, but when bits [13:12] are 11, we will have to decide where to put the data.

One method is to leave all addresses with bits [13:12] = 11 out of the cache. Another method is to have theses addresses map to the same lines as addresses with bits [13:12] = 00, 01 or 10.

For either method, there are 5 bits for line address and $14 - 5 = 9$ bits for index (where the most significant two bits of the index have special significance). If we want to map addresses with bits [13:12] = 11 to the same cache lines as addresses with bits [13:12] = 01, bit 13 needs to be in the tag as well as the index. In this case, there are $26 - 13 = 13$ tag bits. If the addresses with bits [13:12] = 11 are left out of the cache, there are $26 - 14 = 12$ tag bits.

$$\text{Address for } 11\rightarrow 01: \overbrace{A_{25}\ldots A_{14}}^{\text{tag}}\ \overbrace{A_{13}}^{\text{tag/index}}\ \overbrace{A_{12}\ldots A_5}^{\text{index}}\overbrace{A_4\ldots A_0}^{\text{line}}$$

$$\text{Directory entry for } 11\rightarrow 01: \overbrace{13bits}^{\text{tag}}$$

d. From (b), DTMR for 12Kb fully associative cache: 0.0425

Adjustment for direct-mapped (8KB): 1.35

Adjustment for direct-mapped (16KB): 1.38

Adjustment for direct-mapped (12KB): $\frac{1.35+1.38}{2} = 1.365$

DTMR for 12KB direct-mapped cache: $1.365 \times 0.0425 = 5.8\%$

The actual miss rate will probably be worse than this, since this assumes that the addresses are evenly distributed throughout the 12KB, but due to the implementation limitations, this even distribution cannot be achieved.

Problem 5.14

8 KB integrated level 1 cache (direct mapped, 16 B lines)

128 KB integrated level 2 cache (2 way, 16 B lines)

Solo Miss Rate for L2 cache:

The solo miss rate for L2 cache is same as the global miss rate.

From Table A.1 and A.4, $.02 \times 1.17 = .0234$.

Local Miss Rate for L2 cache:

The miss rate of an 8 KB level 1 cache is $.075 \times 1.32 = .099$ from Tables A.1 and A.4, assuming an R/M machine. The number of memory access/I for R/M architecture in scientific environment is:

.73 (instruction) + .34 (data read) + .21 (data write) = 1.23 (pages 31–5).

Missed memory access/I for L1 = $1.23 \times .099 = .1218$.

From solo miss rate, we know that we the miss rate for L2 cache is .0234.

Missed memory access/I for L2 = $1.23 \times .0234 = .0288$.

So, L2 local miss rate = $\frac{.0288}{.099} = .291$.

Problem 5.18

a. CPI lost due to cache misses

User-only, R/M environment

I-Cache:

8KB, direct-mapped

64B lines

DTMR: 2.4 % (Table A.3)

Adjustment for direct-mapped: 1.46 (Table A.4)

I-cache MR = $.024 \times 1.46 = .035, 3.5\%$

I-cache MP = $5 + 1$ cycle$/4B \times 64B = 21$ cycles

$$\begin{aligned} \text{CPI lost due to I-misses} &= \text{I-Refs/I} \times \text{MR}_{\text{I-cache}} \times \text{MP}_{\text{I-cache}} \\ &= 1.0 \times .035 \times 21 \\ &= .735 \end{aligned}$$

D-cache:

4KB, direct-mapped

64B lines

CBWA

w = % dirty = 50%

DTMR: 5.5% (Table A.2)

Adjustment for direct-mapped: 1.45 (it says .45 in Table A.4, should be 1.45)

D-Cache $MR = .055 \times 1.45 = .0798 = 7.98\%$

D-cache $MP = 5 + 1\ /4B \times 64B = 21$ cycles

$$\begin{aligned} \text{CPI lost due to D-misses} &= \text{D-refs/I} \times MR_{\text{D-cache}} \times (1 + \%\ \text{dirty}) \times MP_{\text{D-cache}} \\ &= .5 \times .0798 \times 1.5 \times 21 \\ &= 1.26 \end{aligned}$$

Total CPI loss = .735 + 1.26 = 2.0 (optional)

b. Find the number of I and D-directory bits and corresponding **rbe** (area) for both directories.

I-Cache:

Tag size = $26b - \log_2(8KB) = 13b$

Control bits = valid bit = $1b$

Directory bits = $14 \times (8KB/64B) = 14 \times 128 = 1792b \times .6 = 1075.2$ **rbe**

D-Cache:

Tag size = $26b - \log_2(4KB) = 14b$

Control bits = valid bit + dirty bit = $2b$

Directory bits = $16 \times (4KB/64B) = 1024b \times .6 = 614.4$ **rbe**

c. Find the number of color bits in each cache

Since both caches are direct-mapped, we are going to have to worry about the 8KB I-cache, which is larger than the page size.

Page offset = $\log_2 4096 = 12$

I-Cache:

$$\begin{aligned} \text{Cache index + block offset} &= \log_2(8192/64) + \log_2(64) \\ &= 7 + 6 = \log_2 8192 = 13 \text{ bits} \\ \text{Color bits for I-cache} &= 13b \text{ (index + offset)} - 12b \text{ (page offset)} \\ &= 1 \text{ bit} \end{aligned}$$

D-cache:

$$\begin{aligned} \text{Cache index + block offset} &= \log_2(4096/64) + \log_2(64) \\ &= 6 + 6 = \log_2 4096 = 12 \text{ bits} \\ \text{Color bits for D-cache} &= 12b \text{ (index +offset)} - 12b \text{ (page offset)} \\ &= 0 \text{ bits} \end{aligned}$$

$\rightarrow$ No page coloring is necessary for D-cache.

The operating system must be able to ensure that text (i.e., code) pages match $V = R$ in the first bit of the page address. This requires $2^1 = 2$ free page lists.

Chapter 6. Memory System Design

Problem 6.1

The memory module uses 64 $4^M \times 1^b$ chips for 32MB of data and 8 $4^M \times 1^b$ chips for ECC. This allows 64 bits + 8 bits ECC to be accessed in parallel, forming a physical word.

T_{access}/module	=	120 ns
T_c	=	120 ns
T_{nibble}	=	40 ns up to four accesses
Physical word	=	64 bits + 8 bits ECC
Bus transit	=	20 ns (one way)
ECC	=	40 ns

a. Memory system access time

$$\begin{aligned} &= \text{Bus transit} + T_{\text{access}}/\text{module} + T_{\text{ECC}} + \text{Bus transit} \\ &= 20 + 120 + 40 + 20 \\ &= 200\,\text{ns} \end{aligned}$$

b. Maximum memory data bandwidth

Since we are allowed to have multiple buses and ECC units, we think of 4 memory modules as one very wide memory with 256 bits (64 bits × 4) wide datapath. That is, 256 bits can be fetched in parallel. The only limiting factor here is T_c for each module. For random access, each module cannot be accessed faster than 120 ns. So, the maximum bandwidth is $\frac{256 \text{ bits}}{120\,\text{ns}} = 267 \times 10^6 Bps = 254 MBps$.

c. From above, 256 bits can be fetched in parallel. We can use nibble mode for 4 consecutive accesses. Thus we can fetch $4 \times 256 = 1024$ bits every $120 + (4-1) \times 40 = 240$ ns. So the maximum bandwidth is $\frac{1024 \text{ bits}}{240\,\text{ns}} = 533 \times 10^6 Bps = 509 MBps$.

The low-order bits of the address are divided up as follows:

bits 0–2:	byte offset
bits 3–4:	module address
bits 5–6:	nibble address

Problem 6.2

a. For a page mode, the best organization will place a single page on a single module; this takes advantage of locality of reference as consecutive references to a single (2K) page will be able to take advantage of page mode. This is better than nibble mode, where only references to the same four words can take advantage of the optimized T_{nibble} time. Thus, the lower 11 bits of the word address are used as a (DRAM) page offset, and the address is broken up as follows:

bits 0–2: byte offset

bits 3–13: page offset

bits 14–15: module address

The advantage to interleaving is that accesses to different pages can be overlapped.

b. This system will perform significantly better than nibble mode for:
   1. Non-sequential access patterns.
   2. Access patterns that exhibit locality within a DRAM page.
   3. Access patterns that exhibit locality within multiple pages.
   4. Access patterns that sequentially traverse pages.

Problem 6.3

Hamming Code Design

Data size $(m) = 18$ bits

$2^k \geq m + k + 1$

Solve for k, and get $k = 5$

Total message length $= 18 + 5 = 23$ bits

Each correction bit covers its group bits (m)

k_1: 1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23

k_2: 2–3, 6–7, 10–11, 14–15, 18–19, 22–23

k_3: 4–7, 12–15, 20–23

k_4: 8–15

k_5: 16–23

1. Code for SEC (single error correction):

f	f	f	f	f	f	f	f	f	f	f	f	f	f	f	f	f	f	f	f	f	f	f
k	k	m	k	m	m	m	k	m	m	m	m	m	m	m	k	m	m	m	m	m	m	m

The logic equation for each k is just XOR of each k's group bits.

2. Code for DEC (double error correction):

To detect double bit errors, we must add a final parity bit for the entire SEC code.

f	f	f	f	f	f	f	f	f	f	f	f	f	f	f	f	f	f	f	f	f	f	f	f
k	k	m	k	m	m	m	k	m	m	m	m	m	m	m	k	m	m	m	m	m	m	m	k

Problem 6.4

- 40 MIPS
- 0.9 Instruction refs/Inst
- 0.3 Data read/Inst and 0.1 Data write/Inst
- $T_a = 200\,\text{ns}$
- $T_c = 100\,\text{ns}$

- Use open queue model

a. The allocation of modules to instruction and data

i) Memory modules for instruction

MAPS for instruction $= 0.9 \times 40 = 36$ MAPS

$\rho = \lambda_s/m \times T_c = 36 \times 10^6/m \times 100 \times 10^{-9} = 3.6/m$

Use $m = 8$, $\rho = .45$ So, 8 modules are necessary.

ii) Memory modules for data

MAPS for data $= .4 \times 40 = 16$ MAPS

$\rho = \frac{16 \times 10^6}{m} \times 10^{-7} = \frac{1.6}{m}$

Use $m = 4$, $\rho = .4$ So, 4 modules are necessary.

12 modules are necessary in total.

b. Effective T_w per reference (overall)

Assume this is $M_B/D/1$

$p = \frac{1}{m}$

$T_w = \frac{1}{\mu} \times \frac{\rho - p}{2(1-\rho)} = T_c \times \frac{\rho - \frac{1}{m}}{2(1-\rho)}$

For instruction memory,

$$T_w = 100 \times 10^{-9} \frac{.45 - .125}{2(1 - .45)} = 29.55\,\text{ns}$$

For data memory,

$$T_w = 100 \times 10^{-9} \frac{.4 - .25}{2(1 - .4)} = 12.5\,\text{ns}$$

Overall $T_w = \frac{.9 \times 29.55\,\text{ns} + .4 \times 12.5\,\text{ns}}{.4 + .9} = 24.3\,\text{ns}$

c. Queue size

For instruction memory, $Q_{ot} = \lambda \times T_w$ for Inst $= 36 \times 10^6 \times 29.55 \times 10^{-9} = 1.06$

For data memory, $Q_{ot} = \lambda \times T_w$ for data $= 16 \times 10^6 \times 12.5 \times 10^{-9} = .2$

Overall $Q_{ot} = 1.06 + .2 = 1.26$

d. Comparison to a single integrated I and D memory system

MAPS $= 1.3 \times 40$ MIPS $= 52$ MAPS

$\rho = \frac{\lambda_s}{m} \times T_c = \frac{52 \times 10^6}{m} \times 100 \times 10^{-9} = \frac{5.2}{m}$

Use $m = 16$, $\rho = .325$

$T_w = T_c \frac{\rho - \frac{1}{m}}{2(1-\rho)} = 100\,\text{ns}\ \frac{.325 - \frac{1}{16}}{2(1 - .325)} = 19.4\,\text{ns}$

$Q_{ot} = T_w \times \lambda = 19.4\,\text{ns} \times 52 \times 10^6$ per sec $= 1.009$

Type	Number of modules	T_w	Q_{ot}
Split	12	24.3 ns	1.26
Integrated	16	19.4 ns	1

Problem 6.5

Use $M_B/D/1$ closed queue model for realistic results

$T_c = 100\,\text{ns}, T_a = 120\,\text{ns}$

Two references to memory in each memory cycle: $n = 2$

Eight interleaved memory modules: $m = 8$

a. Expected waiting time

$\rho_a = 1 + \frac{2}{8} - \frac{1}{2\times 8} - \sqrt{(1 + \frac{2}{8} - \frac{1}{2\times 8})^2 - \frac{2\times 2}{8}} = .233$

$T_w = T_c \frac{\rho_a - \frac{1}{m}}{2(1-\rho_a)} = 100\,\text{ns}\frac{.233-\frac{1}{8}}{2(1-.233)} = 7.04\,\text{ns}$

b. Total access time

$T_a + T_w = 120\,\text{ns} + 7.04\,\text{ns} = 127.04\,\text{ns}$

c. Mean total number of queued (waiting) requests

$n - B = n - m \times \rho_a = 2 - 8 \times .233 = .136$

d. Offered memory bandwidth

Offered $= n/T_c = 2/100\,\text{ns} = 20$ MAPS

e. Achieved memory bandwidth

$B(m,n)/T_c = 8 \times .233/100\,\text{ns} = 18.64$ MAPS

f. Achieved bandwidth using Strecker's model

$B(m,n) = m(1 - (1 - \frac{1}{m})^n) = 8(1 - (1 - \frac{1}{8})^2) = 1.875$

Bandwidth $= \frac{B}{T_c} = \frac{1.875}{100\times 10^{-9}} = 18.75$ MAPS

Problem 6.7

Integrated Memory with $m = 8$

$C_P = 2$ (Instruction source and data source; assumes no independent writes from a data buffer. If data buffer is assumed, then $C_p = 3$.)

$Z = C_P \times \frac{T_c}{\Delta T} = 3 \times \frac{100(\text{ns})}{1/(40\times 10^6)} = 12$

$\delta = \frac{n}{Z} = \frac{1.3}{12} \times \frac{100(\text{ns})}{1/(40\times 10^6)} = .433$

$$\begin{aligned} B(m,n,\delta) &= m + n - \tfrac{\delta}{2} - \sqrt{(m + n - \tfrac{\delta}{2})^2 - 2nm} \\ &= 8 + 5.2 - \tfrac{.433}{2} - \sqrt{(8 + 5.2 - \tfrac{.433}{2})^2 - 2 \times 5.2 \times 8} \\ &= 3.74 \end{aligned}$$

$\text{Perf}_{\text{ach}} = \frac{3.74}{5.2} \times 40$ MIPS $= 28.8$ MIPS.

Problem 6.8

$T_a = 200\,\text{ns}$, $T_c = 100\,\text{ns}$, $m = 8$, $T_{\text{bus}} = 25\,\text{ns}$, $L = 16$; Copyback with write allocate.

a. Compute $T_{\text{line.access}}$.

$L > m$ and $T_c < m \times T_{\text{bus}}$

$$\begin{aligned} T_{\text{line.access}} &= T_a + (L-1) \times T_{\text{bus}} \\ &= 200 + 15 \times 25 \\ &= 575\,\text{ns} \end{aligned}$$

b. Repeat for $m = 2$ and $m = 4$.

$m = 2, m < L$ and $T_c > m \times T_{\text{bus}}$

$$\begin{aligned} T_{\text{line.access}} &= T_a + T_c(\lceil \tfrac{L}{m} \rceil) + T_{\text{bus}} \times ((L-1) mod\ m) \\ &= 200 + 100(7) + 25(1) \\ &= 925\,\text{ns} \end{aligned}$$

$m = 4, m < L$ and $T_c \leq m \times T_{\text{bus}}$

$$\begin{aligned} T_{\text{line.access}} &= T_a + (L-1)T_{\text{bus}} \\ &= 200 + 15(25) \\ &= 575\,\text{ns} \end{aligned}$$

c. Nibble mode is now introduced:

$T_{\text{nibble}} = 50\,\text{ns}, m = 2, v = 4, T_v = 50\,\text{ns}$

$$\begin{aligned} T_{\text{line.access}} &= T_a + T_c(\lceil \tfrac{L}{m \times v} \rceil - 1) + T_{\text{bus}}(L - \tfrac{L}{m \times v}) \\ &= 200 + 100(1) + 25(16-2) \\ &= 650\,\text{ns} \end{aligned}$$

Problem 6.9

CBWA with $w = .5$, $T_a = 200\,\text{ns}$, $T_c = 100\,\text{ns}$, $m = 8$, $T_{\text{bus}} = 25\,\text{ns}$, $L = 16$

a. Unbuffered line transfer starting at line address

$T_{\text{m.miss}} = (1+w) \times T_{\text{line.access}} = 1.5 \times 575\,\text{ns} = 863\,\text{ns}$

$T_{\text{c.miss}} = (1+w) \times T_{\text{line.access}} = 1.5 \times 575\,\text{ns} = 863\,\text{ns}$

$T_{\text{busy}} = 0\,\text{ns}$

b. Write buffer, line transfer starting at line address

$T_{\text{m.miss}} = (1+w) \times T_{\text{line.access}} = 1.5 \times 575\,\text{ns} = 863\,\text{ns}$

$T_{\text{c.miss}} = T_{\text{line.access}} = 575\,\text{ns}$

$T_{\text{busy}} = w \times T_{\text{line.access}} = 288\,\text{ns}$

c. Access first word

$T_{\text{m.miss}} = (1+w) \times T_{\text{line.access}} = 1.5 \times 575\,\text{ns} = 863\,\text{ns}$

$T_{\text{c.miss}} = T_a = 200\,\text{ns}$

$T_{\text{busy}} = T_{\text{m.miss}} - T_{\text{c.miss}} = 663\,\text{ns}$

Problem 6.13

A processor without a cache accesses every t-th element of a k element vector. Each element is 1 physical word. Assuming $T_a = 200\,\text{ns}$, $T_c = 100\,\text{ns}$, and $T_{\text{bus}} = 25\,\text{ns}$, plot the average access time per element for an 8-way, low-order interleaved memory for $t = 1$ to 12 and $k = 100$.

Figure 2: Problem 6-13.

Problem 6.14

$\lambda_s = 75$ MAPS and $T_s = 100\,\text{ns}$

$\rho = \frac{\lambda_s}{m} T_s = \frac{75 \times 10^6}{m} 100 \times 10^{-9} = \frac{7.5}{m}$

Since ρ should be around .5, use $m = 16$. Then, $\rho = \frac{7.5}{16} = .469$.

$Q_o = \frac{\rho(\rho - p)}{2(1-\rho)} = \frac{.469(.469 - \frac{1}{16})}{2(1-.469)} = .1795$

$Q_{ot} = 16 \times .1795 = 2.87$

a. Using Chebyshev

   Prob (q > BF) $\leq .01$

   Prob (q $\geq$ BF +1) $\leq .01$

   $\frac{Q_o}{BF+1} \leq .01$

   BF $+1 \geq \frac{Q_o}{P} = \frac{.1795}{.01} = 17.95 \approx 18$

   BF ≥ 17

   Total BF (TBF) $= 17 \times 16 = 272$

b. Using M/M/1

Prob (overflow) $\leq .01$

Solve for $\rho^{(\mathrm{TBF}/m)+2} = .01$

$.469^{(\mathrm{TBF}/m)+2} = .01$

TBF ≈ 66

Problem 6.15

You are to design the memory for a 50 MIPS processor $(1\tilde{/}\mathrm{I})$ with 1 instruction and 0.5 data references per instruction. The memory system is to be 16MB. The physical word size is 4B. You are to use 1M x 1b chips with $\mathrm{T}_c = 40\,\mathrm{ns}$. Draw a block diagram of your memory including address, data, and control connections between the processor, DRAM controller, and the memory. Detail what each address bit does. If $\mathrm{T}_a = 100\,\mathrm{ns}$, what are the expected memory occupancy, waiting time, total access time, and total queue size? Discuss the applicability of the Flores model in the analysis of this design?

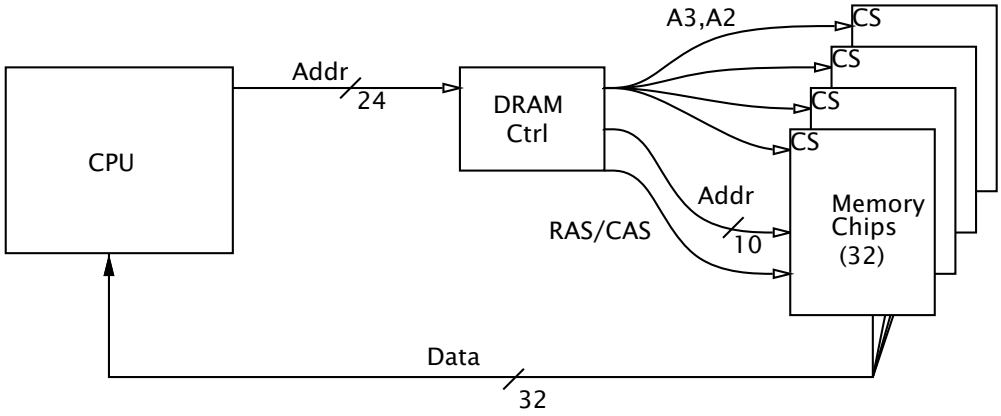


Figure 3: Problem 6-15.

$$
\begin{aligned}
\lambda &= 75MAPS \\
\mu &= m/Tc \\
\rho &= \lambda/\mu = 3/m \\
m &= 4 \\
\rho &= 0.75 \\
T_w &= T_c \frac{\rho - 1/m}{2(1-\rho)} = T_c = 40\,\mathrm{ns} \\
Q_o &= \frac{\rho(\rho - 1/m)}{2(1-\rho)} = 0.75 \\
Q_{o-t} &= 3
\end{aligned}
$$

Because of the high occupancy of the memory, the Flores model is not particularly well suited to this design.

Problem 6.17

IF/cycle = .5

DF/cycle = .3

DS/cycle = .3

$m = 8$ and $T_s = 100\,\text{ns}$(memory)

Processor cycle time $(\Delta T) = 20\,\text{ns} \rightarrow 50$ MIPS

$\lambda = .9 \times 50 \times 10^6 = 45$ MAPS

$n = (.5 + .3 + .1) \times \frac{100}{20} = 4.5$

$z = 3 \times \frac{100}{20} = 15$

$\delta = \frac{n}{z} = \frac{4.5}{15} = .3$

$\rho = \frac{n}{m} = \frac{4.5}{8} = .563$

$$\begin{aligned}
B(m,n,\delta) &= 8 + 4.5 - \frac{.3}{2} - \sqrt{(8 + 4.5 - \frac{.3}{2})^2 - 2 \times 8 \times 4.5} \\
&= 3.377 \\
Bw &= \frac{3.377}{10^{-7}} = 33.77 MAPS \\
\rho_a &= \frac{B}{m} = \frac{3.377}{8} = .422 \\
\text{MIPS}_{\text{ach}} &= \frac{\rho_a}{\rho} 50 \text{ MIPS} = \frac{.422}{.563} = 37.48 \text{ MIPS} \\
T_w &= \frac{n - B}{B} T_s = \frac{4.5 - 3.377}{3.377} \times 100(\text{ ns}) = 33.25(\text{ ns}) \\
Q_{ct} &= n - B = 4.5 - 3.377 = 1.123
\end{aligned}$$

Problem 6.18

a. Line size = 16B

This is already calculated in study 6.3. $\text{Perf}_{\text{rel}} = .78$

b. Line size = 8B

Miss rate = .07

L = $8B/4B = 2$

$$\begin{aligned}
T_{\text{line access}} &= T_a + T_c(\left\lceil \frac{L}{m} \right\rceil - 1) + T_{\text{bus}}((L-1) \bmod m) \\
&= 120 + 100(1-1) + 40((2-1) \bmod 2) \\
&= 120 + 40 = 160\,\text{ns} \\
T_{\text{m.miss}} &= (1+w) T_{\text{line access}} = 1.5 \times 160\,\text{ns} = 240\,\text{ns} \\
\text{Perf}_{\text{rel}} &= \frac{1}{1 + f\lambda_p T_{\text{m.miss}}} \\
&= \frac{1}{1 + .07 \times \frac{1}{40 \times 10^{-9}} \times 240 \times 10^{-9}} \\
&= .704
\end{aligned}$$

c. Line size = 32B

Miss rate = .02

L = $\frac{32B}{4B} = 8$

$$\begin{aligned} T_{\text{line access}} &= 120 + 100(\left\lceil \frac{8}{2} \right\rceil - 1) + 40((8-1) \bmod 2) \\ &= 460 \\ T_{\text{m.miss}} &= (1+w) \times 460\,\text{ns} = 1.5 \times 460\,\text{ns} = 690\,\text{ns} \\ \text{Perf}_{\text{rel}} &= \frac{1}{1 + .02 \times 1/(40 \times 10^{-9}) \times 690 \times 10^{-9}} = .743 \end{aligned}$$

In conclusion, the cache with 16B line size shows the best performance.

Chapter 7. Concurrent Processors

Problem 7.1

VADD VR3, VR1, VR2

VMPY VR5, VR3, VR4

Vector size = 64

$$\begin{aligned} \text{Unchained time} &= 8 \text{ (add startup)} + 64 \text{ (elements/VR)} + 8 \text{ (mpy startup)} + 64 \text{ (elements)} \\ &= 144 \text{ cycles.} \\ \text{Chained time} &= 8 \text{ (add startup)} + 8 \text{ (mpy startup)} + 64 \text{ (elements/VR)} = 80 \text{ cycles.} \end{aligned}$$

a. Implied (average) instruction memory bandwidth

$$= \frac{2 \text{ inst} \times 4 \text{ bytes/inst}}{144 \text{ cycles}} = .0556B/\text{cycle for unchained}$$

and

$$= \frac{2 \text{ inst} \times 4 \text{ bytes/inst}}{80 \text{ cycles}} = .1B/\text{cycle for chained}$$

b.

```
VLD   VR1,  add1
VLD   VR2,  add2
VLD   VR4,  add3
VADD  VR3,  VR1,  VR2
VMPY  VR5,  VR3,  VR4
VST   VR5,  add4
```

$$\begin{aligned} \text{Unchained time} &= (8+8+8) \text{ (load startup)} + 64 \text{ (load completion)} + 8 \text{ (add startup)} \\ &\quad +64 \text{ (add completion)} + 8 \text{ (multiply startup)} + 64 \text{ (multiply completion)} \\ &\quad +8 \text{ (store startup)} + 64 \text{ (store completion)} \\ &= 6 \times 8 + 4 \times 64 = 304 \text{ cycles} \end{aligned}$$

A total of 64 registers $\times$ 4 instructions $\times\ 8^B = 2048$ bytes are moved.

Implied (average) data bandwidth $= \frac{2048 \text{ bytes}}{304 \text{ cycles}} = 6.7\text{B/cycles}$

$$\begin{aligned} \text{Chained time} &= (8+8+8) \text{ (load startup)} + 64 \text{ (load completion)} + 8 \text{ (add startup)} \\ &\quad +8 \text{ (multiply startup)} + 64 \text{ (multiply completion)} + 8 \text{ (store startup)} \end{aligned}$$

+64 (store completion)

$$= 6 \times 8 + 3 \times 64 = 240 \text{ cycles}$$

Implied (average) data bandwidth $= \frac{2048 \text{ bytes}}{240 \text{ cycles}} = 8.5\text{B/cycles}$

Problem 7.2

a. $F37B90_{16} = 330313232100_4$

$r = (0 - 0 + 1 - 2 + 3 - 2 + 3 - 1 + 3 - 0 + 3 - 3) \bmod 5 = 5 \bmod 5 = 0$

So, module address = 0

$$\begin{array}{rl} & 330313232100_4 \\ - & 000230211000_4 \\ \hline & 030023021100_4 = 30B250_{16} \end{array}$$

So, address in module = $30B250_{16}$

b. $AA3347_{16} = 222203031013_4$

$r = (3 - 1 + 0 - 1 + 3 - 0 + 3 - 0 + 2 - 2 + 2 - 2) \bmod 5 = 7 \bmod 5 = 2$

So, module address = 0

$$\begin{array}{rl} & 222203031013_4 \\ - & 202002210012_4 \\ \hline & 020200221001_4 = 220A41_{16} \end{array}$$

So, address in module = $220A41_{16}$

Problem 7.3

a.

m_3	m_2	m_1	m_0	m'_3	m'_2	m'_1	m'_0
0	0	0	0	0	0	1	0
0	0	0	1	0	1	1	1
0	0	1	0	1	0	0	0
0	0	1	1	1	1	0	1
0	1	0	0	0	1	1	0
0	1	0	1	0	0	1	1
0	1	1	0	1	1	0	0
0	1	1	1	1	0	0	1
1	0	0	0	1	0	1	0
1	0	0	1	1	1	1	1
1	0	1	0	0	0	0	0
1	0	1	1	0	1	0	1
1	1	0	0	1	1	1	0
1	1	0	1	1	0	1	1
1	1	1	0	0	1	0	0
1	1	1	1	0	0	0	1

Since each original address maps to a unique hashed address, the scheme works.

b. $F37B90_{16}$

The 4 least significant bits are 0000_2 which map to 0010_2, thus the module address is 2.

$AA3347_{16}$

The 4 least significant bits are 0111_2 which map to 1001_2, thus the module address is 9.

Problem 7.4

$\Delta T = 8\,\text{ns}$

Memory cycle time (T_c) = 64 ns

Two requests per ΔT

a. Offered memory bandwidth

$n = 2 \times \frac{64}{8} = 16$

$\lambda = \frac{n}{T_c} = \frac{16}{64 \times 10^{-9}} = 250$ MAPS

Assume 64 bit word size

$\lambda = 250$ MAPS $\times$ 8 bytes $= 1907$ MBps

b. Achieved bandwidth

Use $M_B/D/1$

$$\begin{aligned} B(m,n) &= 8 + 16 - \frac{1}{2} - \sqrt{(8+16-.5)^2 - 2 \times 8 \times 16} = 6.288 \\ \lambda_a &= B/n \times \lambda_{\text{offered}} \\ &= \frac{6.288}{16} \times 1907 \text{ MBps} \\ &= 750 \text{ MBps} \end{aligned}$$

c. Mean queue size

$Q_{c-t} = n - B = 16 - 6.288 = 9.712$

Problem 7.8

$\gamma = .5$, $m = 17$

$n = 16$ and $T_c = 64$ ns, from problem 7.4

Assume 64 bit word size.

$B(m,n,\gamma) = 17 + 16(1+.5) - .5 - \sqrt{(17+16(1+.5)-.5)^2 - 2 \times 16 \times 17(1.5)} = 11.79$

$\lambda_{\text{ach}} = \frac{B}{T_c} = \frac{11.79}{64 \times 10^{-9}} \approx 184$ MAPS $= 1405$ MBps

Problem 7.10

4 memory requests/ΔT

$\Delta T = 8$ ns

$T_c = 42$ ns

a. Minimum number of interleaving for no contention

$m > n$

$m > 4 \times \frac{42}{8} = 21$, or $m = 32$ if only powers of 2 are allowed.

b. $m = 64$

$$\begin{aligned} \gamma_{\text{opt}} &= \frac{n-1}{2m-2n} = \frac{21-1}{2 \times 64 - 2 \times 21} = .233 \\ \text{Mean TBF} &= n \times \gamma_{\text{opt}} = 21 \times .233 = 4.893 \\ B(m,n,\gamma) &= B(64, 21, .233) = 20.567 \\ \text{Relative perf} &= \frac{20.567}{21} = .979 \end{aligned}$$

Problem 7.11

Assume that both the vector processor and the uniprocessor have the same cycle time. Assume that the uniprocessor can execute one instruction every cycle. Assume that the vector processor can do chaining.

The vector processor can load 3 operands concurrently in advance of using them since there are 3 read ports to memory. It can also store a result concurrently since there is a write port to memory. With chaining, the vector processor can perform 2 arithmetic operations concurrently if there are enough functional units.

Thus, the vector processor can perform 6 operations per cycle and the maximum speedup over a uniprocessor is 6.

Problem 7.14

```
I1   DIV.F   R1,R2,R3
I2   MPY.F   R1,R4,R5
I3   ADD.F   R4,R5,R6
I4   ADD.F   R5,R4,R7
I5   ST.F    ALPHA,R5
```

a. Improved control flow.

Assume there is no reorder buffer.

Assume floating point divide, multiply, and add take 8,4, and 3 separately.

Cycle 1: Decoder Issues I_1 →DIV unit
R2,R3 →DIV Res Stn
TAG_DIV →R1
Cycle 2: Divide begins DIV.F
Cycle 9: Divide completes
Divide requests permission to broadcast result in the next cycle
Cycle 10: Divide Result →R1
Cycle 11: Decoder Issues I_2 →MPY unit
R4,R5 →MPY unit
TAG_MPY →R1
Cycle 12: MPY begins
Decoder issues I_3 →ADD unit
R5,R6 →ADD unit
ADD_TAG1 →R4
Cycle 13: ADD begins
Decoder issues I_4 →ADD unit
ADD_TAG1 →ADD unit
R7 →ADD unit
ADD_TAG2 →R5
Cycle 14: ADD2 waits
Decoder issues I_5 →store unit
ADD_TAG2 →store units
Cycle 15: I_2, I_3 completes, requests for broadcast
(I_2 is granted, I_3 is not granted)
Cycle 16: MPY →R1
Cycle 17: ADD →R4, ADDER2's Res Stn

Cycle 18: ADD (I_4) begins
Cycle 20: I_4 completes and requests for broadcast (granted)
Cycle 21: ADD →R5, store units
Cycle 22: I_5 begins

b. Data flow with value holding reservation station.

Assume reorder buffer.

Cycle 1:	Decoder issues	I_1 →DIV unit
		R3,R2 →DIV Res Stn
		TAG_DIV →R1
Cycle 2:	Divides begins	DIV.F
	Decoder issues	I_2 →MPY unit
		R4,R5 →MPY Res Stn
		TAG_MPY →R1
Cycle 3:	Begin	MPY.F
	Decoder issues	I_3 →ADD unit
		R5,R6 →ADD Res Stn
		TAG_ADD1 →R4
Cycle 4:	Begin	ADD.F
	Decoder issues	I_4 →ADD unit
		TAG_ADD1 →ADD Res Stn
		R7 →ADD Res Stn
		TAG_ADD2 →R5
Cycle 5:	ADDER for I_4 waits	
	Decoder issues	I_5 →Store unit
	TAG_ADD2 →store buffer	
Cycle 6:	MPY completes and requests for broadcast (granted)	
	ADD completes and requests for broadcast (not granted)	
Cycle 7:	Multiply unit →R1	
Cycle 8:	ADD unit (1st unit) →R4,2nd Res Stn of Adder	
Cycle 9:	ADD (I_4) begins	
	DIV completes and requests for broadcast (granted)	
Cycle 10:	DIV.F broadcasts but ignored	
Cycle 11:	ADD completes and request for broadcast (granted)	
Cycle 12:	ADD unit →R5, store buffer	
Cycle 13:	STORE begins	

c. Control flow with shadow register.

Assume reorder buffer.

In this case, it would be better to rewrite the code using register renaming.

DIV.F	R1,R2,R3	renamed
MPY.F	R9,R4,R5	R1→R9
ADD.F	R10,R5,R6	R4→R10
ADD.F	R11,R10,R7	R5→R11
ST.F	ALPHA,R11	R5→R11

Cycle 1: Decoder issues I_1 →DIV unit
R3,R2 →DIV Res Stn
TAG_DIV →R1
Cycle 2: Divides begins DIV.F
Decoder issues I_2 →MPY unit
R4,R5 →MPY Res Stn
TAG_MPY →R9
Cycle 3: Begin MPY.F
Decoder issues I_3 →ADD unit
R5,R6 →ADD Res Stn
TAG_ADD1 →R10
Cycle 4: Begin ADD.F
Decoder issues I_4 →ADD unit
TAG_ADD1 →ADD Res Stn
R7 →ADD Res Stn
TAG_ADD2 →R11
Cycle 5: ADDER for I_4 waits
Decoder issues I_5 → Store unit
TAG_ADD2 → store buffer
Cycle 6: MPY completes and requests for broadcast (granted)
ADD completes and requests for broadcast (not granted)
Cycle 7: Multiply unit → R9
Cycle 8: ADD unit (1st unit) → R10, 2nd Res Stn of Adder
Cycle 9: ADD (I_4) begins
DIV completes and requests for broadcast (granted)
Cycle 10: DIV.F → R1
Cycle 11: ADD completes and request for broadcast (granted)
Cycle 12: ADD unit → R11, store buffer
Cycle 13: STORE begins

Chapter 8. Shared Memory Multiprocessors

Problem 8.1

a.

$$\begin{aligned}
\lambda &= 2/\text{sec} \\
\rho_A &= \lambda_a T_{sa} = \lambda p T_{sa} = 2/\text{sec} \times .1(\text{ sec}) \times p = .2p \\
\rho_B &= \lambda_b T_{sb} = \lambda(1-p) T_{sb} = .4(1-p) \\
T_{wa} &= \frac{\rho_A}{1-\rho_A} T_s = \frac{.2p}{1-.2p} \times .1(\text{sec})(FromM/M/1model) \\
T_{wb} &= \frac{.08(1-p)}{1-.4(1-p)} \\
T_{\text{A-total}} &= T_{wa} + T_{sa} = \frac{.02p}{1-.2p} + .1 \\
T_{\text{B-total}} &= T_{wb} + T_{sb} = \frac{.08(1-p)}{1-.4(1-p)} + .2 \\
T_{\text{total}} &= pT_{\text{A-total}} + (1-p)T_{\text{B-total}}
\end{aligned}$$

We have to minimize T_{total} because we need to minimize average response time.

$$\begin{aligned}
T_{\text{total}} &= T(p) \\
&= p(\frac{.02p}{1-.2p} + .1) + (1-p)(\frac{.08(1-p)}{1-.4(1-p)} + .2) \\
&= \frac{p}{10-2p} + \frac{1-p}{3+2p}
\end{aligned}$$

Within the range of $0 \le p \le 1$, $T'(p) < 0$. This means that $T(p)$ is a decreasing function. So, we get the minimum at $p = 1$:

$$T(p=1) = \frac{1}{10-2} - \frac{1-1}{3+2} = .125 \text{ (sec)}.$$

b. $\lambda = 6$ per sec

$T(p) = \frac{p}{10-6p} + \frac{1-p}{6p-1}$

$T'(p) = \frac{10}{(10-6p)^2} - \frac{5}{(6p-1)^2}$

With $p = .788$, $T(p)$ has its minimum value.

So, $T(p = .788) = .2063$ sec.

Problem 8.3

a. With no cache miss

1 instruction per cycle with no cache miss

b. With cache miss

CPI loss due to cache miss = (1.5 refs/I) × (.04 miss/refs) × (8 cycles/miss)
= .48 CPI

Total CPI for all four processors= 4 CPI_{base} + .48 $\text{CPI}_{\text{penalty}}$
= 4.48 CPI

CPI for four-processor ensemble = 1.12 CPI

Problem 8.4

a. Single pipelined processor performance

Base CPI = 1

CPI loss due to branch = $.2 \times 2 = .4$

CPI loss due to cache miss = $.01 \times 8 \times 1.5 = .12$

CPI loss due to run-on delay = $.1 \times 3 = .3$

$CPI_{\text{Total}} = 1 + .4 + .12 + .3 = 1.82$

b. How effective in the use of parallelism must be?

CPI for SRMP from problem 8.3 is 1.12

Speedup of SRMP = $\frac{1.82}{1.12} = 1.65$

The speedup of 1.8 can be achieved only when we can find 4 independent instructions that can be concurrently executed on SRMP. So, to provide overall speedup over single processor, we should at least be able to find $4/1.65 = 2.42$ independent instructions on average. In another words, $2.42/4 = 60.6$ % utilization of SRMP.

Problem 8.5

Note: we assume the processor halts on cache miss and bus is untenured.

m processors

Time to transfer a line = 8 cycles

$w = .5$

Miss rate = 2%

1.5 refs/inst

a. Bus occupancy, ρ

Bus Transaction time (per 100 inst)

= 100 (inst) $\times$ .02 $\times$ 1.5 (refs/inst) $\times$ (1 +.5) $\times$ 8 cycles = 36 cycles

Processor time (per 100 inst) = 100 cycles since CPI = 1

$\rho = \frac{36}{100+36} \approx .265$ for a single processor

b. Number of processors allowed before bus saturates

$n = \frac{100}{36} = 2.78$

Only two processors can be placed before saturation occurs.

c. To find ρ_a, solve the following two equations by iteration:

$B(n) = n\rho_a = 1 - (1-a)^n$

$a = \frac{\rho}{\rho + \frac{\rho_a}{\rho}(1-\rho)}$

$\rho_a = .243$ by iteration

$B(n=2) = 2 \times \rho_a = 2 \times .243 = .486$

$T_w = \frac{n\rho - B(n)}{B(n)} T_s = \frac{2 \times .265 - .486}{.486} T_s = .09$ (bus cycles)

Problem 8.6

Note: assume processor blocks on a cache miss and bus is untenured.

$n = 4$

Bus time = 8 bus cycles = 8 processor cycles

$B(n=4) = 4\rho_a = 1 - (1 - \frac{\rho}{\rho + \frac{\rho_a}{\rho}(1-\rho)})^4$

$\rho_a = .198$

$B(n=4) = 4 \times .198 = .792$

$T_w = \frac{n\rho - B(n)}{B(n)} T_s = \frac{4 \times .265 - .792}{.792} \times 8$ bus cycles $= 2.707$ bus cycles

For every 100 instructions, $100 \times .02 \times 1.5 \times 1.5 = 4.5$ bus transactions occur. That is, 22.22 instructions are executed between two bus transactions. This is processor execution time.

a. Find λ.

λ_a= 1/(Processor execution time + bus time + T_w)
= $\frac{1}{22.22+8+2.707}$
=.0304/bus cycle

b. Performance
$\frac{\lambda_a}{\lambda} = \frac{22.22+8}{22.22+8+2.707} = .918$
Achieved performance = .918× original performance

Problem 8.8

a. With resubmission: (This is already done in 8.5 c.)
$B(n=2) = .486$, $\frac{T_w}{T_s} = .09$

b. Without resubmission:
$\rho = .265$
$B(n=2) = 1 - (1-\rho)^n = 1 - (1 - .265)^2 \approx .460$
$\rho_a = \frac{B(n=2)}{2} = .23$
$\frac{T_w}{T_s} = \frac{n\rho - B}{B} = \frac{2 \times .265 - .460}{.460} = .15$

Problem 8.9

Baseline network: 4×4 switch, $k = 4$
$N = 1024$
$l = 200$ bits (both requests and reply)
$w = 16$ bits
$h = 1$

Assume we ignore service time:

a. $n = \lceil \log_k N \rceil = \lceil \log_4 1024 \rceil = 5$

b. Without network contention:
$T_{\text{dynamic}} = T_c = n + \frac{l}{w} + h = 5 + \frac{200}{16} + 1 = 18.5$
$T_{\text{total(request+reply)}} = 2 \times 18.5 = 37$ cycles

c. $m = .015$, $\rho = m\frac{l}{w} = .015 \times \frac{200}{16} = .188$
$T_w = \frac{\rho \frac{l}{w}(1-\frac{1}{k})}{2(1-\rho)} = \frac{.188 \times \frac{200}{16} \times (1-\frac{1}{4})}{2(1-.188)} = 1.085$ cycles
$T_{\text{dynamic}} = 18.5 + 1.085 = 19.59$ cycles
$T_{\text{total (request reply)}} = 2 \times 19.59 = 39.17$ cycles

Problem 8.11

Direct static network

Nodes $= 32 \times 32 -$ 2D Torus

$l = 200$

$w = 32$

$h = 1$

a. Expected distance
$nk_d = n \times \frac{k}{4} = 2 \times \frac{32}{4} = 16$

b. $T_{\text{static}} = T_c = h \times n \times k_d + \frac{l}{w} = 16 + \frac{200}{32} = 22.25$ cycles
$T_{\text{total(request+reply)}} = 22.25 \times 2 = 44.5$ cycles

c. $m = .015$

$\rho = \frac{mk_d}{2}\frac{l}{w} = \frac{.015 \times 8}{2} \times \frac{200}{32} = \frac{.75}{2} = .375$

$T_w = \frac{\rho}{1-\rho}\frac{l}{k_d \times w}(1 + \frac{1}{n}) = \frac{.375}{1-.375}\frac{200}{8 \times 32}(1 + .5) = .703$

d. $T_{\text{total}} = 2 \times (T_c + T_w) = 2 \times (22.25 + .703) = 45.9$ cycles

Problem 8.14

Note: In this problem, we ignore the assumption on low occupancy although this might simplify the problem. So, the graph is valid through a relatively large occupancy (or m). Another important point is that a hypercube is likely to have more network requests from neighboring nodes than a grid, since more nodes are connected to a node. So, for fair comparison, the network latency in a grid should be compared with a 16 times larger value of m in a hypercube.

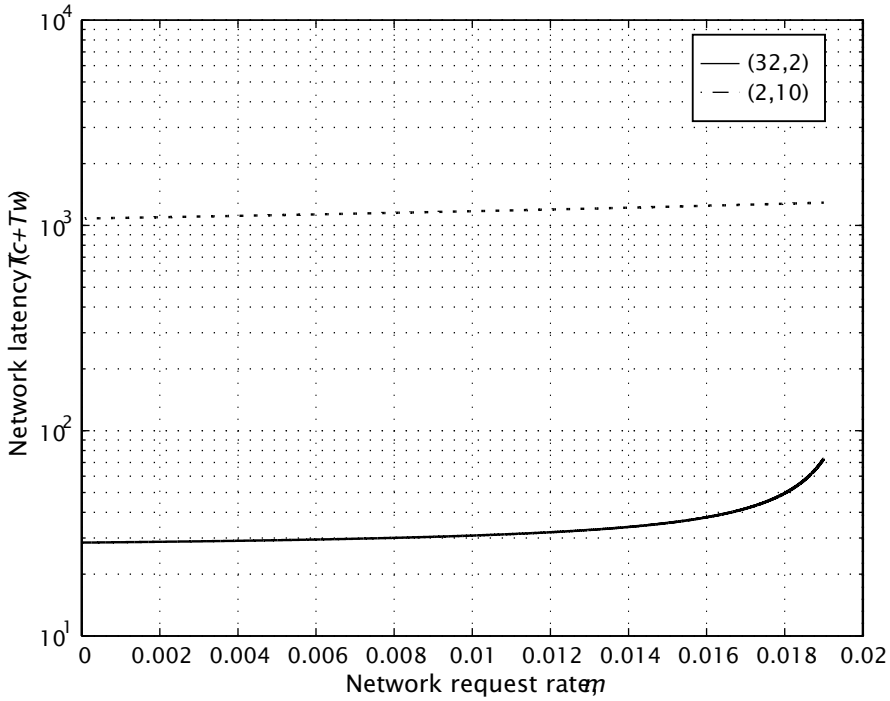


Figure 4: Problem 8-14.

a. $(k, n) = (32, 2)$-grid

$T_{ch} = C_1(32^{\frac{2}{2}-1}) = C_1$ s

$l/w = 200/16 = 12.5$

Fan-in + Fan-out $= 64 = 2nw \rightarrow w = \frac{64}{2\times 2} = 16$

$k_d = \frac{k}{4} = \frac{32}{4} = 8$

$T_c = nk_d + \frac{l}{w} = 2 \times 8 + 12.5 = 28.5$ cycles

$\rho = \frac{mk_d l}{2w} = m\frac{8}{2} \times 12.5 = 50m$

$T_w = \frac{\rho}{1-\rho}\frac{l}{w}\frac{1}{k_d}(1 + \frac{1}{n}) = \frac{50m}{1-50m} \times 12.5 \times \frac{1}{8} \times 1.5 = \frac{117.188m}{1-50m}$ cycles

$\frac{T}{C_1} = \frac{(T_c+T_w)T_{ch}}{C_1} = 28.5 + \frac{117.188m}{1-50m}$

b. (2,10)

$T_{ch} = C_1(2^{\frac{10}{2}-1}) = 16C_1$ s

$l/w = 200/3.2 = 62.5$

Fan-in + Fan-out $= 64 = 2nw \rightarrow w = \frac{64}{2\times 10} = 3.2$

$k_d = \frac{k}{4} = \frac{2}{4} = .5$

$T_c = nk_d + \frac{l}{w} = 10 \times .5 + 62.5 = 67.5$ cycles

$\rho = \frac{16mk_d l}{2w} = 250m$

$T_w = \frac{\rho}{2(1-\rho)}\frac{l}{w} = \frac{250m}{2(1-250m)} \times 62.5 = \frac{7812.5m}{1-250m}$ cycles

$\frac{T}{C_1} = \frac{(T_c+T_w)T_{ch}}{C_1} = 16 \times 67.5 + \frac{7812.5m}{1-250m} = 1080 + \frac{7812.5m}{1-250m}$

Problem 8.17

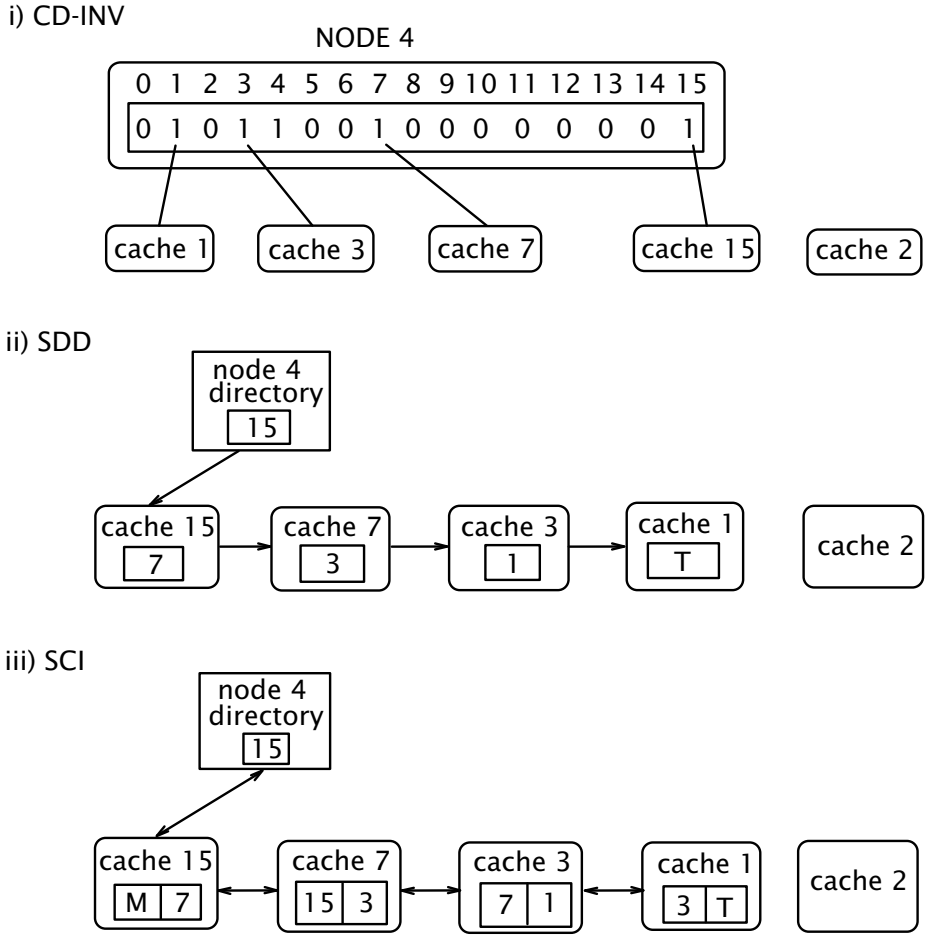


Figure 5: Problem 8-17(a): Immediately before the write takes place.

Problem 8.18

Line transmission takes 9 cycles

Invalidation or acknowledgement takes 1 cycle per hop

$15 \leftrightarrow 0 \leftrightarrow 1 \leftrightarrow 2 \leftrightarrow 3 \leftrightarrow 4 \leftrightarrow 5 \leftrightarrow 6 \leftrightarrow 7$.

a. CD-INV

Step 1: Node 2 sends write miss to node 4 (2 cycles)

Step 2: Node 4 replies to node 2 with invalidation count and data (9 + 1 cycles). Assume step 3 begins immediately after node 4 sends.

Step 3: We assume invalidation or acknowledgement sends out in both directions.

Cycle	1	2	3	4	5	6	7	8	9
	4→7 (I) 4→15 (I)			7→2 (A)		15→2 (A) done			done done
		4→1 (I)			1→2 (A) done				
			4→3 (I)	3→2 (A)					

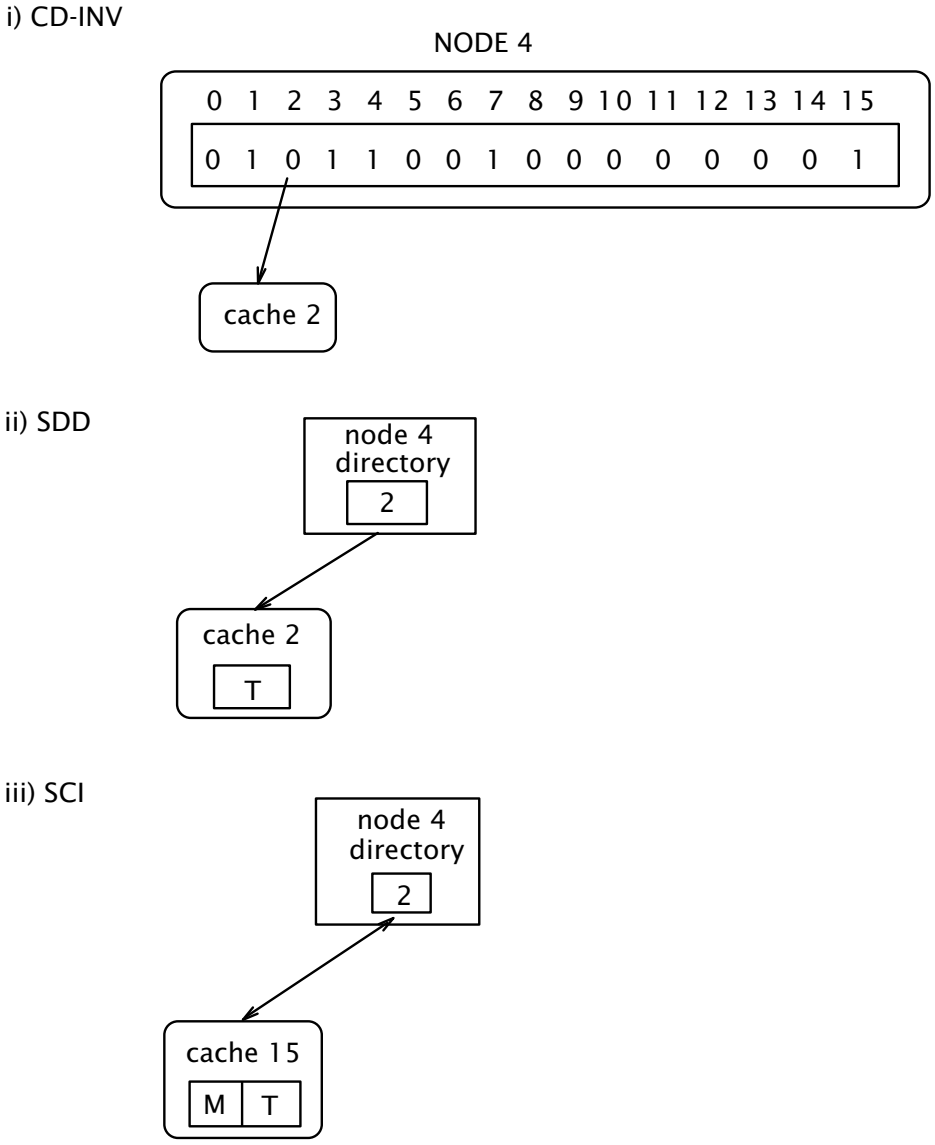


Figure 6: Problem 8-17(b): After the write takes place and is acknowledged.

I means Invalidation, and A means acknowledgement.

Total cycles = 2 + 10 + 9 = 21 cycles.

b. SCI

Step 1: cache 2 sends write miss to node 4 (2 cycles)
Step 2: node 4 sends cache 15 to cache 2 (2 cycles)
Step 3: cache 2 invalidates cache 15 (3 cycles)
Step 4: cache 15 acknowledges to cache 2 with cache 7 and data (3 + 9 cycles)
Step 5: cache 2 invalidates cache 7 (5 cycles)
Step 6: cache 7 acknowledges to cache 2 with cache 3 (5 cycles)
Step 7: cache 2 invalidates cache 3 (1 cycle)
Step 8: cache 3 acknowledges to cache 2 with cache 1 (1 cycle)
Step 9: cache 2 invalidates cache 1 (1 cycle)
Step 10: cache 1 acknowledges to cache 2 with tail (1 cycle)

Total network cycles = 33.

Problem 8.19

a. Immediately before the write takes place

(i) CD-UP Node 4 directory

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	0	1	0	0	0	1	0	0	0	0	0	0	0	1

(ii) DD-UP

node 4		cache 15		cache 7		cache 3		cache 1		cache 2
15	→	7	→	3	→	1	→	T		

b. After write takes place and is acknowledged

(i) CD-UP Node 4 directory

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	1	0	0	0	1	0	0	0	0	0	0	0	1

(ii) DD-UP

node 4		cache 2		cache 15		cache 7		cache 3		cache 1
2	→	15	→	7	→	3	→	1	→	T

Chapter 9. I/O and the Storage Hierarchy

Problem 9.1

A request is made to the Hitachi DK516 drive outlined in Table 9.4. The new track is 30 tracks distant from the current position. The request is to move 20 sections into a buffer (the transfer rate is 3 Mbytes). The starting block location is unknown but assumed to be uniformly distributed over possible locations on the track. What is the total time to access and transfer the requested data?

Seek Time:

From Table 9.5, $a = 3.0$, $b = 0.45$. The seek distance is 30 tracks. By the equation, seek time = 5.46 ms.

Rotation:

Uniformly distributed over [0, 16.67 ms]. Expected rotation delay = 8.33 ms.

Transfer:

20 blocks at 512 bytes at 3 Mbytes/sec = 3.41 ms.

Total = $5.46 + 8.33 + 3.41$ ms = 17.2 ms.

Problem 9.2

$\lambda = \frac{1}{33\text{ms}} = 30.3$ requests/sec

$\mu = 50$ requests/sec; in other words, $T_s = 20$ ms

$\rho = \frac{30.3}{50} = .606$

Using M/G/1,

a. Total response time
   $T_w = \frac{\rho}{2(1-\rho)}(1 + c^2)T_s = \frac{.606}{2(1-.606)}(1 + .5) \times 20$ ms = 23.07 ms
   $T_r = T_w + T_s = 23.07 + 20.0$ ms = 43.07 ms

b. Average number of requests queued at the disk awaiting service
   $Q = \frac{\rho^2}{2(1-\rho)} \times 1.5 = \frac{.606^2}{2(1-.606)} \times 1.5 = .699$

Problem 9.4

Repeat the first example in study 9.1 with the following changes: $c^2 = 0.5$, $T_{\text{user}} = 20$ ms (i.e., 200K user-state instructions before an I/O request), and $n = 3$.

First, look at the disk to determine how applicable the open-queue model really is. $\lambda = 1$ request per 30 ms (time for user process to generate an I/O and system to process request) = 33.3 requests/sec. Thus, $\rho = 2/3$ and $T_w = 30$ ms. So with the open-queue model, the CPU generates an I/O request; 30 ms later (on average) the disk begins service on the request; 20 ms later the request is complete.

The 50 ms it takes to perform the disk operation for the first task is less than the time the CPU will spend processing the other two jobs in memory (60 ms), thus as an approximation, the open-queue model will apply.

Since we are dealing with statistical behavior, there will be some instances in which the disk queueing time and disk service time will exceed the time the CPU is processing the two other tasks. We will use the closed-queue asymptotic model to better estimate the delays.

$T_u = 30$ ms.

$T_s = 20$ ms.

$T_c = 50$ ms.

$n = 3$.

Now, the rate of requests to the disk = rate of requests serviced by the CPU = $\lambda = \min(1/T_u, n/T_c) =$ 33.3 requests/sec.

Since $T_u > T_s$, we need to use the inverted server model. What this really means is that the CPU, not the disk, is the queueing bottleneck. Therefore, it is the CPU queueing delays which will ultimately lower the peak system throughput. Thus,

$T'_u = 20$ ms.

$T'_s = 30$ ms.

$T_c = 50$ ms.

$n = 3$.

Notice, however, that the service rates of the disk and CPU are closer than they were in example 9.1. We should therefore expect the relative queueing delays at the disk to be potentially higher and our model to be less accurate.

We compute $r = T'_u/T'_s = 2/3$. Since we have a small n, we use the limited population correction for the CPU utilization. Notice that the correction for ρ in section 9.4.2 depends on the distribution of service time being exponential. Since the service time we are concerned with is the CPU's service time, we need to assume that the CPU's $c^2 = 1.0$. With that assumption, $\rho_a = 0.975$ (by applying the equation for $n = 3$). Finally, $\lambda_a = 32.5$ requests/second.

With this we can derive the utilizations and waiting times for the CPU and the disk. In most queueing systems, you want to find queueing delays, utilizations, throughputs, and response times. You should know how to find each of these values.

Problem 9.5

$T_{\text{user}} = \frac{200K}{40 \text{ MIPS}} = 5$ ms

$T_{\text{sys}} = 2.5$ ms

$n = 3$

$T_s = 20$ ms

Using a noninverted model,

$T_u = 7.5$ ms

$T_s = 20$ ms

$r = .375$

$\rho_a = \frac{1+r+\frac{r^2}{2}}{1+r+\frac{r^2}{2}+\frac{r^3}{6}} = .94$

$\lambda_a = \frac{.94}{.020} = 47.2$ instead of 50

Problem 9.6

Our job is to find the value of T_{user} at $n = 1$ that has the same user computation time per second ($\lambda_a T_{\text{user}}$) as $n = 2$.

At $n = 2$, user computation time ($\lambda_a T_{\text{user}}$) = 445 ms.

$T_u = T_{\text{user}} + T_{\text{system}}$

$\rho_a = \frac{1}{1+r}, r = \frac{T_u}{T_s}$ (inverted service case)

$\lambda_a = \frac{\rho_a}{max(T_u, T_s)}$

We can find T_{user} by taking an initial approximation and iterating several times.

Let's pick a T_{user} that makes $T_u = 20$ ms.

$T_u - T_{\text{system}} = 20 \text{ ms} - 2.5 \text{ ms} = 17.5$ ms.

$r = 1$

$\rho_a = \frac{1}{1+1} = .5$

$\lambda_a = \frac{.5}{.02} = 25$ transactions/sec

$\lambda_a T_{\text{user}} = 25 \times 17.5 \text{ ms} = 437.5$ ms

One more iteration:

Try $T_{\text{user}} = 18$ ms

$\lambda_a T_{\text{user}} = 24.7 \times 18 = 444.3$ ms

This is a close enough value.

In conclusion, for approximately $T_{\text{user}} = 18$ ms at $n = 1$ we have the same performance as $n = 2$.

Problem 9.7

In study 9.2, suppose we increase the server processor performance by a factor of 4, but all other system parameters remain the same. Find the disk utilization and user response time for $n = 20$ (assume $c^2 = 0.5$ for the disk).

We have the following expected service times:

$T_{\text{disk}} = 18.8$ ms.

$T_{\text{server}} = 10$ ms.

$T_{\text{network}} = 3.6$ ms.

Thus, the disk will be the bottleneck in this case. We will use the asymptotic model without the low population correction. We have the following parameters for our model:

$T_c = 1$ sec. Remember, the workstation user (if not slowed down by the rest of the system) will generate a disk operation every second.

$T_s = 18.8$ ms

$T_u = 981.2$ ms

$r = T_u/T_s = 52.2$

$f = T_s/T_c = 0.0188$

By equation 9.6,

$T_w/T_c = 0.0080835$

$T_{w\ \text{disk}} = 8.384$ ms

$\lambda_{a\ \text{disk}} = 19.83$

$\rho_{a\ \text{disk}} = 0.3728$.

Now, we can compute the expected waiting times and utilizations of the other nodes in the system using the open queue model:

$\rho_{a\ \text{net}} = \lambda_a T_{snet} = 0.0714$.

$T_{w\ \text{net}} = 0.138$ ms.

$\rho_{a\ \text{server}} = \lambda_a T_{s\ \text{server}} = 0.1983$

$T_{w\ \text{server}} = 1.24$ ms.

$T_{w\ \text{disk}} = 8.384$ ms.

$T_{w\ \text{total}} = 9.762$ ms.

Notice that although the service times of the server and the disk differed by less than 50%, the waiting times are nearly 50x different. Comparing the waiting time of the total system with the waiting time of the disk, the disk is responsible for about 85% of the waiting time. We should determine if the waiting time of the net and server impact the request rate:

$\lambda'_a = \lambda/(T_{w\ \text{disk}} + T_{w\ \text{net}} + T_{w\ \text{server}} + T_c) = 19.807$.

With this value, we should recompute the utilizations and waiting times. Since, however, this is very close to our initial estimate of the achieved request rate, it will not alter our results significantly.

Now for the workstation:

$\lambda = 20$ requests/sec.

$T_c = 1$ sec.

$T_w =$ sum of T_w for each node $= 9.76$ ms.

Thus, the response time is $T_w + T_{sdisk} + T_{sserver} + T_{snet} = 42.14$ ms.

Problem 9.12

Rotation speed = 3600 rpm

$$\text{Seek time} = \text{a} + \text{b}\sqrt{\text{seek tracks}} = 3.0 + 0.45\sqrt{23-1} = 5.11\,\text{ms}.$$

When the seek is complete, the head has moved $5.11/16.67 * 77 = 23.611$ sectors. The rotational delay for the rotation of the additional 60.39 sectors is 13.07 ms. The transfer takes $16*512/3*10^6 = 2.73ms$. The total elapsed time is $5.11 + 13.07 + 2.73 = 20.91$ ms.

Problem 9.13

The perceived delay is:

$$\frac{\lambda - \lambda_a}{\lambda_a} T_c$$

For (a), $\frac{20-18.9}{18.9} \times 70 = 4.07$ ms

For (b), $\frac{61.5-44.5}{44.5} \times 32.5 = 12.42$ ms

Problem 9.18

Without disk cache, $T_{\text{user}} = 40$ ms, $T_{\text{sys}} = 10$ ms.

With disk cache, $T_{\text{user}} = \frac{T_{\text{user (no disk cache)}}}{.3} = \frac{40\text{ ms}}{.3} = 133.3$ ms.

Then

$$\begin{aligned}
T_u &= 133.3\text{ ms} + 10\text{ ms} = 143.3\text{ ms} \\
T_s &= 20\text{ ms} \\
n &= 2 \\
r &= \frac{T_s}{T_u} = \frac{20}{143.3} = .14 \\
\rho_a &= \frac{1+r}{1+r+\frac{r^2}{2}} = .99 \\
\lambda_a &= \frac{.99}{.143} = 6.92 \text{ requests per second}
\end{aligned}$$

The maximum capability of the processor is 1 request each 143.3 ms. We achieved 99% of this capacity—i.e., $\frac{1}{6.92} = 144.5$ ms.