### 1-1

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>A\cdot B \cdot C</th>
<th>(A\cdot B \cdot C)'</th>
<th>A'</th>
<th>B'</th>
<th>C'</th>
<th>A'\cdot B + C'</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### 1-2

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>A \oplus B</th>
<th>A \oplus B \oplus C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

### 1-3

(a) \(A + A \cdot B = A (1 + B) = A\)

(b) \(A \cdot B + A \cdot B' = A (B + B') = A\)

(c) \(A' \cdot B + A \cdot C = C (A' \cdot B + A) = C (A' + A) (B + A) = (A + B) C\)

(d) \(A' \cdot B + A \cdot B \cdot C' + A \cdot B \cdot C = A' \cdot B + A \cdot B (C' + C) = A' \cdot B + A \cdot B = B (A' + A) = B\)

### 1-4

(a) \(A \cdot B + A \cdot (C \cdot D + C \cdot D') = A \cdot B + A \cdot C (D + D') = A \cdot B + C\)

(b) \((B \cdot C' + A' \cdot D) (A \cdot B' + C \cdot D') = \)

\[= AB' \cdot C' + A' \cdot A' B' D + B C' \cdot C' D' + A' C' D' = 0\]

\[\text{with terms: } 0, 0, 0, 0\]
1-5
(a) \((A+B)'(A'+B')' = (A'B')(AB) = 0\)
(b) \(A + A'B + A'B' = A + A'(B+B') = A + A' = 1\)

1-6
\[ F = x'y + xy'z' \]
\[ F' = (x + y') (x' + y' + z) = x'y' + xy' + y' + xy + y'z \\
\quad = y'(1 + x' + x + z) + xz = y' + xz \]
\[ F \cdot F' = (x' + y + y'z') (y' + yz) = 0 \cdot 0 + 0 = 0 \]
\[ F + F' = x'y + xy'z' + y' + xz(y + y') \\
\quad = y'y + xz(y' + x) + y'(1 + x) = x'y + xy + y' \\
\quad = y'(x + x) + y' = y' + y' = 1 \]

1-7
(a) \[\begin{array}{c|c|c|c|c|}
\hline
x & y & z & F \\
\hline
0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 \\
0 & 1 & 0 & 0 \\
0 & 1 & 1 & 1 \\
1 & 0 & 0 & 0 \\
1 & 0 & 1 & 1 \\
1 & 1 & 0 & 0 \\
1 & 1 & 1 & 1 \\
\hline
\end{array}\]
(b) \[ F = xy'z' + xy'z' + xyz' \]

(c) \[ F = xy'z' + xy'z' + xyz' \\
\quad = y'z(x + x') + xz(y + y') \\
\quad = y'z + xz \]

(d) Same as (a)
\( F = x'y' + x'y \)

\( F = y + x'z \)

\( F = xy' + xz + yz \)

\( F = c' + a'b \)

\( F = BCD + A'B'D' \)

\( F = C'D + ABC + ABD \)

\( F = A'c' + A'b'd' + ACD + \left\{ \frac{A'B'D}{BCD} \right\} \)

\( F = B'D + B'D' + \left\{ \frac{A'B}{A'D'} \right\} \)
(1) \( F = x'y'z' \)

(2) \( F' = x'z + y'z \)

\[
F = (x + z')(y + z')
\]

(1) \( F = A'C' + C'D + B'D \)

(2) \( F = (A + D')(C'D) (A + B' + C) \)

\[
F = B'D' + A'B' + AC
\]
(a) \( F = x_3' + w_3' \)

(b) \( = (x' + z)(w' + z') \)

\[ s = x'y_3' + x'y_3' + xy_3' + xy_3 \]
\[ = x'(y_3' + y_3') + x(y_3' + y_3) \]
\[ = x'(y \oplus 3) + x(y \oplus 3) \]
\[ = x \oplus y \oplus 3 \]

\[ F = xy + x_3 + y_3 \]

\[ A = xy + x_3 + y_3 \]
\[ F = x \oplus y \oplus 3 \]

By inspection
When \( D = 0 \); \( J = 0, K = 1; Q \rightarrow 0 \)
When \( D = 1 \); \( J = 1, K = 0; Q \rightarrow 1 \)

1-18

see text; Section 1-6 for derivation.

1-19

\[ D_A = x'y + xA ; \quad D_B = x'B + xA ; \quad \beta = B \]

![Circuit Diagram](image)

(b) Present State | Inputs | Next State | Output
--- | --- | --- | ---
\( A \) \( B \) | \( x \) \( y \) | \( A \) \( B \) | \( \beta \)
00 | 00 | 00 | 0
00 | 01 | 10 | 0
00 | 10 | 00 | 0
00 | 11 | 00 | 0
01 | 00 | 01 | 1
01 | 01 | 11 | 1
01 | 10 | 01 | 1
01 | 11 | 11 | 1
10 | 00 | 00 | 0
10 | 01 | 00 | 0
10 | 10 | 10 | 0
10 | 11 | 11 | 0
11 | 00 | 01 | 1
11 | 01 | 01 | 1
11 | 10 | 11 | 1
11 | 11 | 11 | 1
\[ J_A = K_A = x \]
\[ J_B = K_B = \overline{x} \]

1-21 Count up-down binary counter with enable \( E \)

<table>
<thead>
<tr>
<th>Present State</th>
<th>Inputs</th>
<th>Next State</th>
<th>Flip-flop inputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A ) ( B ) ( E )</td>
<td>( A ) ( B )</td>
<td>( J_A ) ( K_A )</td>
<td>( J_B ) ( K_B )</td>
</tr>
<tr>
<td>00 00</td>
<td>00</td>
<td>0 x 0</td>
<td>0 x 0</td>
</tr>
<tr>
<td>00 01</td>
<td>00</td>
<td>0 x 0</td>
<td>0 x 0</td>
</tr>
<tr>
<td>00 10</td>
<td>11</td>
<td>1 x 1</td>
<td>1 x 1</td>
</tr>
<tr>
<td>01 10</td>
<td>01</td>
<td>0 x 1</td>
<td>0 x 1</td>
</tr>
<tr>
<td>01 01</td>
<td>01</td>
<td>0 x 0</td>
<td>0 x 0</td>
</tr>
<tr>
<td>01 11</td>
<td>00</td>
<td>0 x 1</td>
<td>0 x 1</td>
</tr>
<tr>
<td>01 10</td>
<td>10</td>
<td>1 x 1</td>
<td>1 x 1</td>
</tr>
<tr>
<td>10 00</td>
<td>10</td>
<td>x 1 0</td>
<td>x 1 0</td>
</tr>
<tr>
<td>10 01</td>
<td>10</td>
<td>x 1 0</td>
<td>x 1 0</td>
</tr>
<tr>
<td>10 11</td>
<td>01</td>
<td>x 0 1</td>
<td>x 0 1</td>
</tr>
<tr>
<td>11 00</td>
<td>11</td>
<td>x 2 0</td>
<td>x 2 0</td>
</tr>
<tr>
<td>11 01</td>
<td>11</td>
<td>x 2 0</td>
<td>x 2 0</td>
</tr>
<tr>
<td>11 10</td>
<td>10</td>
<td>x 0 1</td>
<td>x 0 1</td>
</tr>
<tr>
<td>11 11</td>
<td>00</td>
<td>x 1 1</td>
<td>x 1 1</td>
</tr>
</tbody>
</table>

\[ J_A = (B \overline{x} + B' x') E \]

\[ K_A = (B x + B' x') E \]

\[ J_B = E \]

\[ K_B = E \]
(a) Inverters - 2 pins each \[ 12/2 = 6 \text{ gates} \] 7404
(b) 2-input XOR - 3 pins each \[ 12/3 = 4 \text{ gates} \] 7486
(c) 3-input OR - 4 pins each \[ 12/4 = 3 \text{ gates} \] 7421
(d) 4-input AND - 5 pins each \[ 12/5 = 2 \text{ gates} \] 74260
(e) 5-input NOR - 6 pins each \[ 12/6 = 2 \text{ gates} \] 7430
(f) 8-input NAND - 9 pins \[ 1 \text{ gate} \] 74107
(g) JK flip-flop - 6 pins each \[ 12/6 = 2 \text{ FFs} \] 74107

2-2

(a) 74155 - Similar to two decoders as in Fig 2-2.
(b) 74157 - Similar to multiplexers of Fig. 2-5.
(c) 74194 - Similar to register of Fig. 2-9.
(d) 74163 - Similar to counter of Fig. 2-11.

2-3
2-4 \begin{align*}
D_0 &= (A_1 + A_0 + E')' = A_1' A_0 E \\
D_1 &= (A_1' A_0' + E')' = A_1' A_0 E \\
D_2 &= (A_1' + A_0 + E')' = A_1 A_0' E \\
D_3 &= (A_1' + A_0' + E')' = A_1 A_0 E
\end{align*}

2-5 Remove the inverter from the E input in Fig. 2.2(a).

2-6 If all inputs equal 0 or if only $D_0 = 1$:

- the outputs $A_2 A_1 A_0 = 000$
- Needs one more output to recognize the all zeros input condition.

2-7
2-9
"Parallel load"
Clock pulses

When the parallel load input = 1, the clock pulses go through the AND gate and the data inputs are loaded into the register. When the parallel load input = 0, the output of the AND gate remains at 0.

2-10
The buffer gate does not perform logic. It is used for signal amplification of the clock input.

2-11

One stage of Register Fig. 2-7

Function table

<table>
<thead>
<tr>
<th>S</th>
<th>S0</th>
<th>YAYB</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>A0 B0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>A1 B1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>A2 B2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>A3 B3</td>
</tr>
</tbody>
</table>
2-12  Input bits  4-bit register
   0  →  1110
   1  →  0111
   1  →  1011
   0  →  1101
   1  →  0110

2-13  
serial transfer: One bit at a time by shifting.
Parallel transfer: All bits at the same time.
Input serial data by shifting, output data in parallel.
Input data with parallel load, output data by shifting.

2-14  
1000 → 0100 → 0010 → 0001

2-15

2-16  (a) 4; (b) 9

2-17
Count Load clear clock

Fig. 2-11  Output carry Load  to next stage
After the count reaches \( N-1 = 1001 \), the register loads 0000 from inputs.

\[
\begin{align*}
(2) \quad 2^k \times 16 &= 2^n \times 16 \\
& \text{Address lines} \quad \frac{11}{16} \\
(3) \quad 64K \times 8 &= 2^{16} \times 8 \\
& \text{Data lines} \quad \frac{16}{8} \\
(4) \quad 16M \times 32 &= 2^{24} \times 32 \\
& \quad \frac{24}{32} \\
(5) \quad 4G \times 64 &= 2^{32} \times 64 \\
& \quad \frac{32}{64}
\end{align*}
\]

\[
\begin{align*}
(2) \quad 2^k \times 2 &= 4K = 4096 \text{ bytes} \\
& \quad \text{(b) } 64K \times 1 = 64K = 2^{16} \text{ bytes} \\
& \quad \text{(c) } 2^{24} \times 4 = 2^{26} \text{ bytes} \\
& \quad \text{(d) } 2^{32} \times 8 = 2^{35} \text{ bytes}
\end{align*}
\]

\[
\frac{4096 \times 16}{128 \times 8} = \frac{2^{12} \times 2^{4}}{2^{7} \times 2^{3}} = 2^{6} = 64 \text{ chips}
\]

5 inputs common to all chips

\[
\begin{align*}
\text{2-23} & \quad 12 \text{ data inputs} + 2 \text{ enable inputs} + 8 \text{ data outputs} + 2 \text{ for power} = 24 \text{ pins}
\end{align*}
\]
CHAPTER 3

3-1
\[
(101110)_2 = 32 + 8 + 4 + 2 = 46 \\
(11010101)_2 = 64 + 32 + 16 + 4 + 1 = 117 \\
(110110100)_2 = 256 + 128 + 32 + 16 + 4 = 436
\]

3-2
\[
(12121)_3 = 3^4 + 2 \times 3^3 + 3^2 + 2 \times 3 + 1 = 81 + 54 + 9 + 6 + 1 = 151 \\
(4310)_5 = 4 \times 5^3 + 3 \times 5^2 + 1 = 500 + 75 + 5 = 580 \\
(50)_7 = 5 \times 7 = 35 \\
(118)_12 = 12^2 + 1 \times 12 + 8 = 114 + 12 + 8 = 262
\]

3-3
\[
\begin{align*}
(1231)_{10} &= 1024 + 128 + 64 + 15 = 2^{10} + 2^7 + 2^6 + 2^3 + 2 + 1 = (10011001111)_2 \\
(673)_{10} &= 512 + 128 + 32 + 1 = 2^9 + 2^7 + 2^5 + 1 = (1010100001)_2 \\
(1998)_{10} &= 1024 + 512 + 256 + 128 + 64 + 8 + 4 + 2 = 2^{10} + 2^9 + 2^8 + 2^7 + 2^6 + 2^3 + 2 + 1 = (1111001110)_2
\end{align*}
\]

3-4
\[
\begin{align*}
(7562)_{10} &= (16612)_8 \\
(1938)_{10} &= (792)_{16} \\
(175)_{10} &= (10101111)_2
\end{align*}
\]

3-5
\[
(F3A7C2)_{16} = (1111 \ 00111 \ 0100 \ 0111 \ 1100 \ 0010)_2 \\
= (74723702)_8
\]

3-6
\[
(x^2 - 10x + 31)_y = [(x - 5)(x - 8)]_{10} \\
= x^2 - (5 + 8)x + (40)_{10}x \\
\text{Therefore: } (10)_y = (13)_{10} \quad y = 13
\]
\[
\text{Also } (31)_y = 3 \times 13 + 1 = (40)_{10} \quad (y = 13)
\]
3-7 \( (215)_{10} = 128 + 64 + 16 + 7 = (11010111)_{2} \)

(a) \(0000\ 1101\ 0111\) Binary
(b) \(000\ 011\ 010\ 111\) Binary coded octal
(c) \(0000\ 1101\ 0111\) Binary coded hexadecimal
(d) \(0010\ 0001\ 0101\) Binary coded decimal

3-8 \( (29.5)_{10} = 256 + 32 + 7 = (100100111)_{2} \)

(a) \(0000\ 0000\ 0000\ 0001\ 0010\ 0111\)
(b) \(0000\ 0000\ 0000\ 0010\ 1001\ 0101\)
(c) \(10110010\ 00111001\ 00110101\)

3-10 JOHN DOE

3-11 87650123; 99019899; 09990048; 99999999.

3-12 876100; 909343; 900000; 000000

3-13 01010001; 01111110; 01111110; 01111110; 01001010; 01111110; 10000000; 11111110; 00000000

3-14
(a) 5250 + 8679 = 13929
(b) 1753 + 1360 = 3113
(c) 920 + 9750 = 10670
(d) 1200 + 9750 = 10950

\[ \begin{array}{cccc}
5250 & +8679 & \downarrow 10's \ complement \downarrow & 13929 \\
1753 & +1360 & \downarrow 10's \ complement \downarrow & 3113 \\
920 & +9750 & & 10670 \\
1200 & +9750 & & 10950
\end{array} \]

3-15
(a) \(11010 + 10000 = 101010\)
(b) \(11010 + 00100 = 111110\)
(c) \(0001000 + 010000 = 0101000\)
(d) \(1010100 + 0101100 = 00100000\)

\[ \begin{array}{cccc}
11010 & +10000 & \downarrow 101010 & (26-16=10) \\
11010 & +00100 & \downarrow 011101 & (26-13=13) \\
0001000 & +010000 & \downarrow 0101000 & -101100 \\
1010000 & +0010100 & \downarrow 00000000 & (84-84=0)
\end{array} \]

\[ \begin{array}{cccc}
11010 & +10000 & \downarrow 101010 & (26-16=10) \\
11010 & +00100 & \downarrow 011101 & (26-13=13) \\
0001000 & +010000 & \downarrow 0101000 & -101100 \\
1010000 & +0010100 & \downarrow 00000000 & (84-84=0)
\end{array} \]
\[
\begin{align*}
3-16 \\
+42 &= 0101010 \\
-42 &= 1010110 \\
\frac{(+42)}{} &= 0101010 \\
\frac{(-13)}{} &= 1110011 \\
\frac{(+29)}{} &= 0011101 \\
-13 &= 1110011 \\
\frac{(-42)}{} &= 1010110 \\
\frac{(+13)}{} &= 0001101 \\
\frac{(-29)}{} &= 1100011
\end{align*}
\]

3-17

\[
\begin{align*}
\text{01} & \leftarrow \text{last two carries} \rightarrow 10 \\
\text{+70} & \text{ 01000110} \\
\text{+80} & \text{ 01010000} \\
\text{+150} & \text{ 10010110} \\
\text{greater} & \text{ than 127}
\end{align*}
\]

3-18

\[
\begin{align*}
2) (-638) & \quad 9362 \\
\text{+785} & \quad \rightarrow 0785 \\
\text{+147} & \quad \rightarrow 0147 \\
\text{b) (-638)} & \quad 9362 \\
\text{(-185)} & \quad +9815 \\
\text{(-823)} & \quad \frac{9177}{9}
\end{align*}
\]

3-19

\[
\begin{align*}
\text{Mantissa} \\
\text{+ 26 bits} \\
\text{Largest: +0,1111...1} \\
1-2^{-26} \\
\text{Smallest: +0,0000...0} \\
\text{(normalized)} \quad 2^{-1} \\
\text{Exponent} \\
\text{8 bits} \quad \text{36 bits} \\
\text{+1111111} \\
\text{+255} \quad \text{(-2)}^{-26} \quad \text{+255} \\
\text{(-256)} \\
\text{2^{-256}}
\end{align*}
\]

3-20

\[
\begin{align*}
46.5 = 32 + 8 + 4 + 2 + 0.5 &= (101100.1)_2 \\
\text{Sign} \\
\text{0 1011101000000000} \\
\text{24-bit mantissa} \\
\text{00000110} \\
\text{8-bit exponent (+6)}
\end{align*}
\]
### 3-21

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Gray code</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>11000</td>
</tr>
<tr>
<td>17</td>
<td>11001</td>
</tr>
<tr>
<td>18</td>
<td>11011</td>
</tr>
<tr>
<td>19</td>
<td>11010</td>
</tr>
<tr>
<td>20</td>
<td>11110</td>
</tr>
<tr>
<td>21</td>
<td>11111</td>
</tr>
<tr>
<td>22</td>
<td>01001</td>
</tr>
<tr>
<td>23</td>
<td>01000</td>
</tr>
<tr>
<td>24</td>
<td>01010</td>
</tr>
<tr>
<td>25</td>
<td>01011</td>
</tr>
<tr>
<td>26</td>
<td>01101</td>
</tr>
<tr>
<td>27</td>
<td>01110</td>
</tr>
<tr>
<td>28</td>
<td>01111</td>
</tr>
<tr>
<td>29</td>
<td>10001</td>
</tr>
<tr>
<td>30</td>
<td>10000</td>
</tr>
<tr>
<td>31</td>
<td></td>
</tr>
</tbody>
</table>

### 3-22

8620

(a) BCD 1000 0110 0010 0000

(b) XS-3 1011 1001 0101 0011

(c) 24 21 1110 1100 0010 0000

(d) Binary 10000110101100 (8192+256+128+32+8+4)

### 3-23

<table>
<thead>
<tr>
<th>Decimal</th>
<th>BCD with even parity</th>
<th>BCD with odd parity</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 00000</td>
<td>1 00000</td>
</tr>
<tr>
<td>1</td>
<td>1 00001</td>
<td>0 00001</td>
</tr>
<tr>
<td>2</td>
<td>1 00100</td>
<td>0 00100</td>
</tr>
<tr>
<td>3</td>
<td>0 00111</td>
<td>1 00111</td>
</tr>
<tr>
<td>4</td>
<td>1 01000</td>
<td>0 01000</td>
</tr>
<tr>
<td>5</td>
<td>0 01011</td>
<td>1 01011</td>
</tr>
<tr>
<td>6</td>
<td>0 01101</td>
<td>1 01101</td>
</tr>
<tr>
<td>7</td>
<td>1 01111</td>
<td>0 01111</td>
</tr>
<tr>
<td>8</td>
<td>1 10000</td>
<td>0 10000</td>
</tr>
<tr>
<td>9</td>
<td>0 10011</td>
<td>1 10011</td>
</tr>
</tbody>
</table>
3-24

\[ \begin{array}{cccc}
3984 &=& 0011 & 1111 & 1110 & 0100 \\
1100 & 0000 & 0001 & 1011 &=& 6015 \\
\end{array} \]

3-25

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>y = A \oplus B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>C</th>
<th>D</th>
<th>z = C \oplus D</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ y \oplus z = x \oplus y \oplus z \]

\[ \begin{array}{cccc}
A & B & C & D \\
000 & 00 & 01 & 01 \Rightarrow \begin{cases} 
AB = 00 \text{ or } 11 \\
CD = 01 \text{ or } 10 
\end{cases} \quad 0001, 0010, 1101, 1110 \\
011 & 11 & 01 & 10 \Rightarrow \begin{cases} 
AB = 01 \text{ or } 10 \\
CD = 00 \text{ or } 11 
\end{cases} \quad 0100, 0111, 1000, 1011 \\
\end{array} \]

Always odd number of 1's

3-26

Same as in Fig. 3-3 but without the complemented circles in the outputs of the gates.

\[ P = x \oplus y \oplus z \]

\[ \text{Error} = x \oplus y \oplus z \oplus P \]
4.1

4.2

4.3

\[ P: \quad R1 \leftrightarrow R2 \]

\[ P'Q: \quad R1 \leftrightarrow R3 \]

4.4

Connect the 4-line common bus to the four inputs of each register.

Provide a "load" control input in each register.

Provide a clock input for each register.

To transfer from register C to register A:

Apply \( S_1S_0 = 10 \) (to select C for the bus.)

Enable the load input of A

Apply a clock pulse.
4-6
(a) 4 selection lines to select one of 16 registers.
(b) 16 x 1 multiplexers
(c) 32 multiplexers, one for each bit of the registers.

4-7
(a) Read memory word specified by the address in AR into register R2.
(b) Write content of register R3 into the memory word specified by the address in AR.
(c) Read memory word specified by the address in R5 and transfer content to R5 (destroys previous value)

4-8
### Table: 4-12

<table>
<thead>
<tr>
<th>M</th>
<th>A</th>
<th>B</th>
<th>Sum</th>
<th>Carry</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0111</td>
<td>0110</td>
<td>1101</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1000</td>
<td>1001</td>
<td>0001</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1100</td>
<td>1000</td>
<td>0100</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0101</td>
<td>1010</td>
<td>1011</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0000</td>
<td>0001</td>
<td>1111</td>
<td>0</td>
</tr>
</tbody>
</table>
4-13 $A-1 = A + 2^{1}$'s complement of $1 = A + 1111$

4-14

4-15

<table>
<thead>
<tr>
<th>$S$</th>
<th>$C_{in}$</th>
<th>$X$</th>
<th>$Y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>A</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>A</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>A</td>
<td>B</td>
</tr>
</tbody>
</table>

$A - 1 = A + 1111$

$A + B$

$A + 1$

$A - 1$

$A - B$

$S_{in}$

$4 \times 1$ MUX

$X_0$

$Y_0$

$C_1$

$X_1$

$Y_1$

$C_2$

$D_0$

$D_1$

$D_2$

$D_3$

$D_4$

$D_5$

$D_6$

$D_7$

$D_8$
4-16

Binary value of $F_i$

4-17

4-18

(a) $A = 11011001$  
$B = 10110100$  
$A \oplus B = 01101101$

(b) $A = 11011001$  
$B = 11111101$  
$A \lor B = 11111101$

4-19

(a) $AR = 11110010$  
$BR = 11111111$  
$AR = 11110001; BR = 11111111$  
$CR = 10111001; DR = 1101010$

(b) $CR = 10111001$  
$DR = 11010101$  
$CR = 10101000; BR = 00000000$  
$AR = 11110001; DR = 1101010$

(c) $AR = 11110001$  
$CR = 10101000$  
$AR = 01001001; BR = 00000000; CR = 10101000; DR = 1101010$
4-20

$$R = 10011100$$

Arithmetic shift right: $11001110$
Arithmetic shift left: $00111000$ overflow because a negative number changed to positive

4-21

Logical shift left: $R = 11011101$
Circular shift right: $01011101$
Logical shift right: $00111110$
Circular shift left: $01011100$

4-22

$$S = 1 \text{ Shift left}$$

$$\begin{array}{cccc}
A_0 & A_1 & A_2 & A_3 \\
L & L & L & L
\end{array}$$

$$H = \underbrace{0 \ 0 \ 1 \ 0 \ 0 \ 0} \ \text{shift left}$$

4-23

(a) Cannot complement and increment the same register at the same time.

(b) Cannot transfer two different values ($R_2$ and $R_3$) to the same register ($R_1$) at the same time.

(c) Cannot transfer a new value into a register (PC) and increment the original value by one at the same time.
5-1

\[ 256K = 2^8 \times 2^{10} = 2^{18} \]
\[ 64 = 2^6 \]

(a) Address: 18 bits
   Register code: 6 bits
   Indirect bit: 1 bit

\[ \frac{25}{32} - 25 = 7 \text{ bits for opcode.} \]

(b) \[ \begin{array}{ccc}
I & \text{opcode} & \text{register} \\
1 & 7 & 6 \end{array} \]
\[ 18 = 32 \text{ bits} \]

(c) Data: 32 bits; address: 18 bits.

5-2

A direct address instruction needs two references to memory: (1) Read instruction; (2) Read operand.

An indirect address instruction needs three references to memory: (1) Read instruction; (2) Read effective address; (3) Read operand.

5-3

(a) Memory read to bus and load to IR: \( IR \leftarrow M[AR] \)
(b) TR to bus and load to PC: \( PC \leftarrow TR \)
(c) AC to bus, write to memory, and load to DR: \( DR \leftarrow AC, \ M[AR] \leftarrow AC \)
(d) Add DR (or INPR) to AC: \( AC \leftarrow AC + DR \)

5-4

\[ \begin{array}{cccc}
\text{Sr} & \text{Si} & \text{So} & \text{Load (1)} & \text{Memory} & \text{Adder} \\
\text{1} & \text{2} & \text{3} & \text{4} \\
\hline
\text{a) AR} & \text{PC} & 010 \ (\text{PC}) & \text{AR} & \text{Read} & \text{Transfer} \\
\text{b) IR} & M[AR] & 111 \ (\text{M}) & \text{IR} & \text{Write} & \text{DR to AC} \\
\text{c) M[AR]} & \text{TR} & 110 \ (\text{TR}) & \text{M} & \text{Write} & \text{DR to AC} \\
\text{d) DR} & \text{AC} & 100 \ (\text{AC}) & \text{DR} & \text{AC} & \text{Transfer} \\
\end{array} \]
(2) \[ IR = M[PC]\]  
PC cannot provide address to memory,  
Address must be transferred to AR first  
\[ AR = PC \]  
\[ IR = M[AR] \]

(b) \[ AC \leftarrow AC + TR \]  
Add operation must be done with  
\[ DR \] , Transfer TR to DR first.  
\[ DR \leftarrow TR \]  
\[ AC \leftarrow AC + DR \]

(c) \[ DR \leftarrow DR + AC \]  
Result of addition is transferred to  
\[ AC \] (not DR). To save value of AC  
its content must be stored temporarily  
in DR (or TR).  
\[ AC \leftarrow DR \] , \[ DR \leftarrow AC \]  
(See answer to Problem 5-4(b))  
\[ AC \leftarrow AC + DR \]  
\[ AC \leftarrow DR \] , \[ DR \leftarrow AC \]

5-6

(2) \[ \begin{array}{cccc}
0001 & 0000 & 0010 & 0100 \\
& ADD & \ldots & (024)_{16} \\
& ADD \ content \ of \ M[024] \ to \ AC & ADD & 024
\end{array} \]
\[ \begin{array}{cccc}
1011 & 0001 & 0010 & 0100 \\
I \sta & \text{(124)6} & \text{store \ AC \ in \ M[M[124]]} & \sta \text{I 124}
\end{array} \]
\[ \begin{array}{cccc}
0111 & 0000 & 0010 & 0000 \\
\text{Register} & \text{Increment \ AC} & \text{INC}
\end{array} \]

5-7

CLE Clear E
CME Complement E
**Diagram:**

- Clock
- $T_0$
- $T_1$
- $T_2$
- $T_3$
- $C_7 T_3$

$\uparrow$ SC goes to 0 causing $T_0 = 1$

---

**Table:**

<table>
<thead>
<tr>
<th>E</th>
<th>AC</th>
<th>PC</th>
<th>AR</th>
<th>IR</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Initial</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLA</td>
<td>0000</td>
<td>022</td>
<td>800</td>
<td>7800</td>
</tr>
<tr>
<td>CLE</td>
<td>0</td>
<td>A937</td>
<td>022</td>
<td>400</td>
</tr>
<tr>
<td>CMA</td>
<td>56C8</td>
<td>022</td>
<td>200</td>
<td>7200</td>
</tr>
<tr>
<td>CME</td>
<td>0</td>
<td>A937</td>
<td>022</td>
<td>100</td>
</tr>
<tr>
<td>CIR</td>
<td>49B</td>
<td>022</td>
<td>080</td>
<td>7080</td>
</tr>
<tr>
<td>CIL</td>
<td>526F</td>
<td>022</td>
<td>040</td>
<td>7040</td>
</tr>
<tr>
<td>INC</td>
<td>A938</td>
<td>022</td>
<td>020</td>
<td>7020</td>
</tr>
<tr>
<td>SFA</td>
<td>A937</td>
<td>022</td>
<td>010</td>
<td>7010</td>
</tr>
<tr>
<td>SNA</td>
<td>A937</td>
<td>023</td>
<td>008</td>
<td>7008</td>
</tr>
<tr>
<td>SZA</td>
<td>A937</td>
<td>022</td>
<td>004</td>
<td>7004</td>
</tr>
<tr>
<td>SZE</td>
<td>A937</td>
<td>022</td>
<td>002</td>
<td>7002</td>
</tr>
<tr>
<td>HLT</td>
<td>A937</td>
<td>022</td>
<td>001</td>
<td>7001</td>
</tr>
</tbody>
</table>
### 5-10

<table>
<thead>
<tr>
<th>PC</th>
<th>AR</th>
<th>DR</th>
<th>AC</th>
<th>IR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial</td>
<td>021</td>
<td>—</td>
<td>—</td>
<td>A937</td>
</tr>
<tr>
<td>AND</td>
<td>022</td>
<td>083</td>
<td>B8F2</td>
<td>A32</td>
</tr>
<tr>
<td>ADD</td>
<td>022</td>
<td>083</td>
<td>B8F2</td>
<td>6229</td>
</tr>
<tr>
<td>LDA</td>
<td>022</td>
<td>083</td>
<td>B8F2</td>
<td>88F2</td>
</tr>
<tr>
<td>STA</td>
<td>022</td>
<td>083</td>
<td>—</td>
<td>A937</td>
</tr>
<tr>
<td>BUN</td>
<td>083</td>
<td>083</td>
<td>—</td>
<td>A937</td>
</tr>
<tr>
<td>BSA</td>
<td>084</td>
<td>084</td>
<td>—</td>
<td>A937</td>
</tr>
<tr>
<td>TSZ</td>
<td>022</td>
<td>083</td>
<td>B8F3</td>
<td>A937</td>
</tr>
</tbody>
</table>

### 5-11

<table>
<thead>
<tr>
<th>PC</th>
<th>AR</th>
<th>DR</th>
<th>IR</th>
<th>SC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial</td>
<td>7FF</td>
<td>—</td>
<td>—</td>
<td>0</td>
</tr>
<tr>
<td>T0</td>
<td>7FF</td>
<td>7FF</td>
<td>—</td>
<td>1</td>
</tr>
<tr>
<td>T1</td>
<td>800</td>
<td>7FF</td>
<td>EA9F</td>
<td>2</td>
</tr>
<tr>
<td>T2</td>
<td>800</td>
<td>A9F</td>
<td>EA9F</td>
<td>3</td>
</tr>
<tr>
<td>T3</td>
<td>800</td>
<td>C35</td>
<td>EA9F</td>
<td>4</td>
</tr>
<tr>
<td>T4</td>
<td>800</td>
<td>C35</td>
<td>FFFF</td>
<td>EA9F</td>
</tr>
<tr>
<td>T5</td>
<td>800</td>
<td>C35</td>
<td>0000</td>
<td>EA9F</td>
</tr>
<tr>
<td>T6</td>
<td>801</td>
<td>C35</td>
<td>0000</td>
<td>EA9F</td>
</tr>
</tbody>
</table>

### 5-12

(a) \( q = (1001)_2 \)

\[ \frac{1001}{\text{ADD}} \quad \text{ADD I 32E} \]

(b) \(\text{AC}=7EC3 \quad (\text{ADD})\)

\[\text{DR} = \frac{8B9F}{0A62}\]

(c) \(\text{PC} = 3AF + 1 = 3BO\)

\(\text{AR} = 9AC\)

\(\text{DR} = 8B9F\)

\(\text{AC} = 0A62\)

\(\text{IR} = 932E\)

\(\text{E} = 1\)

\(\text{I} = 1\)

\(\text{SC} = 0000\)

Memory

<table>
<thead>
<tr>
<th>3AF</th>
<th>932E</th>
</tr>
</thead>
<tbody>
<tr>
<td>32E</td>
<td>09AC</td>
</tr>
<tr>
<td>9AC</td>
<td>8B9F</td>
</tr>
</tbody>
</table>

\(\text{AC} = 7EC3\)
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
</table>
| XOR     | D0 T4: \( DR \leftarrow M[AR] \)  
           D0 T5: \( AC \leftarrow AC + DR, \ SC \leftarrow 0 \) |
| ADM     | D1 T4: \( DR \leftarrow M[AR] \)  
           D1 T5: \( DR \leftarrow AC, AC \leftarrow AC + DR \)  
           D1 T6: \( M[AR] \leftarrow AC, AC \leftarrow DR, \ SC \leftarrow 0 \) |
| SUB     | D2 T4: \( DR \leftarrow M[AR] \)  
           D2 T5: \( DR \leftarrow AC, AC \leftarrow DR \)  
           D2 T6: \( AC \leftarrow \overline{AC} \)  
           D2 T7: \( AC \leftarrow AC + 1 \)  
           D2 T8: \( AC \leftarrow AC + DR, \ SC \leftarrow 0 \) |
| XCH     | D3 T4: \( DR \leftarrow M[AR] \)  
           D3 T5: \( M[AR] \leftarrow AC, AC \leftarrow DR, \ SC \leftarrow 0 \) |
| SEQ     | D4 T4: \( DR \leftarrow M[AR] \)  
           D4 T5: \( TR \leftarrow AC, AC \leftarrow AC + DR \)  
           D4 T6: If \( (AC = 0) \) then \( (PC \leftarrow PC + 1) \), \( AC \leftarrow TR \), \( SC \leftarrow 0 \) |
| BPA     | D5 T4: If \( (AC = 0 \land AC(15) = 0) \)  
           then \( (PC \leftarrow AR) \), \( SC \leftarrow 0 \) |

5-14

Converts the ISZ instruction from a memory-reference instruction to a register-reference instruction. The new instruction ICSZ can be executed at time \( T_3 \) instead of time \( T_6 \), a saving of 3 clock cycles.
Modify Fig. 5-9:

Same as Fig. 5-9

\[ T_2 \]

\[ =1 \]

\[ =0 \]

Same as Fig. 5-9

Indirect = 1

\( I = 0 \) (Direct)

\[ T_3 \]

\[ AR(12-15) \leq 0 \]

\[ AR \leftarrow PC, PC \leftarrow PC + 1 \]

\[ T_4 \]

\[ AR \leftarrow M[AR] \]

Execute memory-reference instruction

(a)

<table>
<thead>
<tr>
<th>15</th>
<th>PC</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>AR</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>IR</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td>IR</td>
<td></td>
</tr>
</tbody>
</table>

Memory
64K x 8

| 7 | AC | 0 |
| 7 | DR | 0 |

(b)

Memory

<table>
<thead>
<tr>
<th>off code</th>
<th>8-address</th>
<th>8-address</th>
</tr>
</thead>
<tbody>
<tr>
<td>operand</td>
<td>8-bits</td>
<td></td>
</tr>
</tbody>
</table>

(c)

\( T_0: IR \leftarrow M[PC], PC \leftarrow PC + 1 \n\)

\( T_1: AR(0-7) \leftarrow M[PC], PC \leftarrow PC + 1 \n\)

\( T_2: AR(8-15) \leftarrow M[PC], PC \leftarrow PC + 1 \n\)

\( T_3: DR \leftarrow M[AR] \n\)
IR = "opcode 1 | Address 1 | opcode 2 | Address 2 |

Decoder 1

Decoder 2

1. Read 40-bit double instruction from memory to IR and then increment PC.
2. Decode opcode 1.
3. Execute instruction 1 using address 1.
4. Decode opcode 2.
5. Execute instruction 2 using address 2.
6. Go back to step 1.

5-18
(a) BUN 2300
(b) JON BUN 0 I (Branch indirect with address 0)

5-19

Memory

AC

MUXs

Data

Address

Output

n inputs

READ

WRITE

X_1

X_3

X_1': READ

X_3': WRITE

X_1'X_3': R ← AC

Clock

Load

X_2

Select

= 0 (AC)

= 1 (memory)
5-20 \[ J_F = xT_3 + zT_2 + wT_5 \] \[ K_F = yT_1 + zT_2 + wT_5 \]

\[ \begin{array}{c}
\text{clock} \\
C \\
K \\
J \\
Q \\
F
\end{array} \]

5-21 From Table 5-6: \( z_{dr} = 1 \) if \( DR = 0 \); \( z_{ac} = 1 \) if \( AC = 0 \)

\[ \text{INP} (PC) = R'T_1 + RT_2 + D_2 T_2 Z_{dr} + pB_q(FGI) + pB_8(FGO) + yB_4(A_{15}) + yB_3(A_{15}) + yB_2 Z_{ac} + yB_1 E' \]

\[ \text{LD} (PC) = D_4 T_4 + D_5 T_5 \]

\[ \text{CLR} (PC) = R T_1 \]

The logic diagram is similar to the one in Fig. 5-16.

5-22 \[ \text{Write} = D_3 T_4 + D_5 T_4 + D_6 T_6 + RT_1 \quad \text{\( (M[AR] \leftarrow xx) \)} \]

5-23 \( (T_0 + T_1 + T_2)'(IEN)(FGI + FGO) \): \( R \leftarrow 1 \)

\[ RT_2 : R \leftarrow 0 \]
x_2 \text{ places } DC \text{ onto the bus, From Table 5-6:}

\begin{align*}
R'T_0 & : \text{AR} \leq PC \\
R'T_0 & : \text{TR} \leq PC \\
D_5 T_4 & : M[AR] \leftarrow PC
\end{align*}

\[ x_2 = R'T_0 + R'T_0 + D_5 T_4 = (R'R) T_0 + D_5 T_4 = T_0 + D_5 T_4 \]

From Table 5-6:

\[ \text{CLR}(SC) = R T_2 + D_7 T_3 (I' + I) + (D_6 + D_1 + D_2 + D_5) T_5 \]

+ \( D_3 + D_4 \) \( T_4 + D_6 T_6 \)
6-1

```
010 CLA 0000 011 7800
011 ADD 016 C1A5 012 1016
012 BUN 014 C1A5 014 4014
013 HLT 8184 014 7001
014 AND 017 8184 015 0017
015 BUN 013 8184 013 4013
016 C1A5
017 73C6
```

\[
(C1A5)_{16} = \frac{1100\ 0001\ 1010\ 0101}{\text{AND}}
\]

\[
(93C6)_{16} = \frac{1001\ 0011\ 1100\ 0110}{1000\ 0001\ 1000\ 0100} = (8184)_{16}
\]

6-2

```
100 5103 ← BSA 103
101 7200 ← CHA FFFE ← Answer
102 7001 ← HLT
103 0000 ← 5101 ← Answer
104 7800 ← CLA 0000
105 7020 ← INC 0001
106 C103 ← BUN 103 I
```

6-3

```
CLA STA SUM \{ SUM = 0 \\
LDA SUM \{ SUM = SUM + A + B \\
ADD A \\
ADD B \\
STA SUM \\
LDA C \\
CHA \\
INC \\
ADD DIF \\
STA DIF \\
LDA DIF \\
LDA SUM \\
ADD DIF \\
STA SUM \{ SUM = SUM + DIF
```

A more efficient compiler will optimize the machine code as follows:

```
LDA A \\
ADD B \\
STA SUM \\
LDA C \\
CHA \\
INC \\
ADD DIF \\
STA DIF \\
ADD SUM \\
STA SUM
```
A line of code such as: LDA I is interpreted by the assembler (Fig. 6-2) as a two symbol field with I as the symbolic address. A line of code such as: LDA I I is interpreted as a three symbol field. The first I is an address symbol and the second I as the Indirect bit.

Answer: Yes, it can be used for this assembler.

The assembler will not detect an ORG or END if the line has a label; according to the flow chart of Fig. 6-1. Such a label has no meaning and constitutes an error. To detect the error, modify the flow chart of Fig. 6-1:

---

6-6 (a) memory word

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>DE</td>
<td>C space</td>
<td>-3</td>
<td>5 CR</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>characters</th>
<th>Hex</th>
<th>binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>DE</td>
<td>44 45</td>
<td>0100 0100 0100 0101</td>
</tr>
<tr>
<td>C space</td>
<td>43 20</td>
<td>0100 0011 0010 0000</td>
</tr>
<tr>
<td>-3</td>
<td>2D 33</td>
<td>0010 1101 0011 0011</td>
</tr>
<tr>
<td>5 CR</td>
<td>35 0D</td>
<td>0011 0101 0000 1101</td>
</tr>
</tbody>
</table>

(b) $(35)_{10} = (0000 0000 0010 0011)_2$

$-35 \rightarrow 1111 1111 1101 1101 = (FFDD)_{16}$
6-7 (c) LOP 105 \( (100)_{10}^{16} = (0000000000000000)_{2}^{16} \)
ADS 10B \((-100)_{10}^{16} = (1111111111001100)_{2}^{16} = (FF9C)_{16}^{16} \)
PTR 10C \((75)_{10}^{16} = (0000000001001011)_{2}^{16} = (0048)_{16}^{16} \)
NBR 10D \((23)_{10}^{16} = (0000000000010111)_{2}^{16} = (0017)_{17}^{16} \)
CTR 10E 
SUM 10F 

(b) 

<table>
<thead>
<tr>
<th>Loc</th>
<th>Hex</th>
<th>ORG 100</th>
<th>Loc</th>
<th>Hex</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>210B</td>
<td>LDA ADS</td>
<td>10B</td>
<td>0150</td>
</tr>
<tr>
<td>101</td>
<td>310C</td>
<td>STA PTR</td>
<td>10C</td>
<td>00000</td>
</tr>
<tr>
<td>102</td>
<td>210D</td>
<td>LDA NBR</td>
<td>10D</td>
<td>FF9C</td>
</tr>
<tr>
<td>103</td>
<td>310E</td>
<td>STA CTR</td>
<td>10E</td>
<td>00000</td>
</tr>
<tr>
<td>104</td>
<td>7800</td>
<td>CLA</td>
<td>10F</td>
<td>00000</td>
</tr>
<tr>
<td>105</td>
<td>910C</td>
<td>LOP ADD PTR</td>
<td>10F</td>
<td>00000</td>
</tr>
<tr>
<td>106</td>
<td>610C</td>
<td>ISZ PTR</td>
<td>150</td>
<td>0048</td>
</tr>
<tr>
<td>107</td>
<td>610D</td>
<td>ISZ CTR</td>
<td>:</td>
<td>:</td>
</tr>
<tr>
<td>108</td>
<td>4105</td>
<td>BUN LOP</td>
<td>:</td>
<td>:</td>
</tr>
<tr>
<td>109</td>
<td>310F</td>
<td>STA SUM</td>
<td>1B3</td>
<td>0017</td>
</tr>
<tr>
<td>10A</td>
<td>7001</td>
<td>HLT</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

6-8 Modify flow chart of Fig. 6-1

6-9

6-10 LC=1 A, BSS 10

\[
\begin{align*}
\text{scan next line} & \quad \text{store symbol in symbol table} \\
\text{yes} & \quad \text{BSS} \\
\text{no} & \quad \text{LC=LC+N} \\
\text{LC=LC+1} & \\
\end{align*}
\]

get op-code

\[
\begin{align*}
\text{search symbol table for address symbol} & \quad \text{yes} \\
\text{is symbol in table?} & \quad \text{error message} \\
\text{no} & \quad \text{increment LC} \\
\text{yes} & \quad \text{scan next line} \\
\end{align*}
\]

\[
\begin{align*}
\text{same as Fig. 6-2}, & \\
\text{same as Fig. 6-2} & \\
\text{same as Fig. 6-2} & \\
\text{same as Fig. 6-2} & \\
\end{align*}
\]

35
(a) MRI Table

<table>
<thead>
<tr>
<th>Memory Word</th>
<th>Symbol</th>
<th>HEX</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>AND</td>
<td>41 4D</td>
</tr>
<tr>
<td>2</td>
<td>D.space</td>
<td>44 20</td>
</tr>
<tr>
<td>3</td>
<td>Value</td>
<td>00 00</td>
</tr>
<tr>
<td>4</td>
<td>ADD</td>
<td>41 44</td>
</tr>
<tr>
<td>5</td>
<td>D.space</td>
<td>44 20</td>
</tr>
<tr>
<td>6</td>
<td>Value</td>
<td>10 00</td>
</tr>
<tr>
<td></td>
<td>etc.</td>
<td></td>
</tr>
</tbody>
</table>

(b) Non-MRI Table

<table>
<thead>
<tr>
<th>Memory Word</th>
<th>Symbol</th>
<th>HEX</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CLR</td>
<td>43 4C</td>
</tr>
<tr>
<td>2</td>
<td>A.space</td>
<td>41 20</td>
</tr>
<tr>
<td>3</td>
<td>Value</td>
<td>78 00</td>
</tr>
<tr>
<td>4</td>
<td>CLE</td>
<td>43 4C</td>
</tr>
<tr>
<td>5</td>
<td>E.space</td>
<td>45 20</td>
</tr>
<tr>
<td>6</td>
<td>Value</td>
<td>74 00</td>
</tr>
<tr>
<td></td>
<td>etc.</td>
<td></td>
</tr>
</tbody>
</table>

6-11

LDA B
CMA
INC
ADD A
/ Form A-B
S PA
/ skip if AC positive.
BUN N10
/(A-B)<0, go to N10
SZA
/ skip if AC=0
BUN N30
/(A-B)>0, go to N30
BUN N20
/(A-B)=0, go to N20

6-12 (a) The program counts the number of 1's in the number stored in location WRD. Since WRD = (62C1)\(_{16}\) = (0110 0010 1100 0001)\(_{2}\), number of 1's is 6; so CTR will have (0006)\(_{16}\)

(b)

<table>
<thead>
<tr>
<th>Line</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>ORG 100</td>
</tr>
<tr>
<td>101</td>
<td>CLE</td>
</tr>
<tr>
<td>102</td>
<td>STA CTR</td>
</tr>
<tr>
<td></td>
<td>/ Initialize counter to zero</td>
</tr>
<tr>
<td>103</td>
<td>LDA WRD</td>
</tr>
<tr>
<td>104</td>
<td>SZA</td>
</tr>
<tr>
<td>105</td>
<td>BUN ROT</td>
</tr>
<tr>
<td>106</td>
<td>BUN STP</td>
</tr>
<tr>
<td></td>
<td>/ Word is zero; stop with CTR=0</td>
</tr>
<tr>
<td>107</td>
<td>ROT, CIL</td>
</tr>
<tr>
<td></td>
<td>/ Bring bit to E</td>
</tr>
<tr>
<td>108</td>
<td>SZE</td>
</tr>
<tr>
<td>109</td>
<td>BUN AGN</td>
</tr>
<tr>
<td></td>
<td>/ bit = 1, go to count it</td>
</tr>
<tr>
<td>10A</td>
<td>BUN ROT</td>
</tr>
<tr>
<td></td>
<td>/ bit = 0, repeat</td>
</tr>
<tr>
<td>10B</td>
<td>AGN, CLE</td>
</tr>
<tr>
<td>10C</td>
<td>ISZ CTR</td>
</tr>
<tr>
<td></td>
<td>/ Increment counter</td>
</tr>
</tbody>
</table>
6-12 (b) Continued

10D 7004 SZA   /check if remaining bits = 0
10E 4107 BUN ROT /No; rotate again
10F 7001 STP, HLT /Yes; stop
110 0000 CTR, HEX 0
111 62C1 WRD, HEX 62C1
END

6-13 (100)₁₀ = (256)₁₀  500 to 5FF → (256)₁₀ locations

ORG 100
LDA ADS
STA PTR   /Initialize pointer
LDA NBR
STA CTR   /Initialize counter to -256
CLA

LOP, STA PTR I   /store zero
ISZ PTR
ISZ CTR
BUN LOP
HLT

ADS, HEX 500
PTR, HEX 0
NBR, DEC -256
CTR, HEX 0
END

6-14

LDA A       /Load multiplier
SZA
BUN NZR    /Is it zero?
HLT
NZR, CMA
INC
STA CTR    /Store -A in counter
CLA

LOP, ADD B   /Add multiplicand
ISZ CTR
BUN LOP    /Repeat loop A times
HLT

A, DEC -     /multiplier
B, DEC -     /multiplicand
CTR, HEX 0   /counter
END
The first time the program is executed, location CTR will go to 0. If the program is executed again starting from location (100)$_{16}$, location CTR will be incremented and will not reach 0 until it is incremented $2^{16} = 65,536$ times, at which time it will reach 0 again.

We need to initialize CTR and P as follows:

```
LDA NBR
STA CTR
CLA
STA P
```

Program

```
NBR, DEC -8
CTR, HEX 0
P, HEX 0
```

Multiplicand is initially in location XL. Will be shifted left into XH (which has zero initially). The partial product will contain two locations PL and PH (initially zero). Multiplier is in location Y, CTR=-16

```
LDA Y
CLR
STA Y
SZE
BUN ONE
BUN ZRO
```

```
ONE, LDA XL
ADD PL
STA PL
CLA
CIL
ADD XH
ADD PH
STA PH
CLE
```

same as beginning of Program in Table 6-14

Double-precision add

```
P ← X + P
```

Same as program in Table 6-15

Continued next page
6-16 continued

ZRO, LDA XL  \[ \text{Double-precision left-shift } XH + XL \]
CIL
STA XL
LDA XH
CIL
STA XH
ISZ CTR
BUN LOP
HLT

Repeat 16 times

6-17

If multiplier is negative, take the 2's complement of multiplier and multiplicand and then proceed as in Table 6-14 (with CTR=7).

Flow-Chart:

```
start
CTR ← 7
P ← 0

Plus \[ Y \]
minus
check sign of multiplier

Y ← Y + 1
2's complement multiplier

X ← X + 1
2's complement multiplicand

Proceed as in Table 6-14
```
C ← A-B
CLE
LDA BL
CMA
INC
ADD AL
STA CL

To form a double-precision 2’s complement of subtrahend \(BH+BL\), a 1’s complement is formed and 1 added once.

Thus, BL is complemented and incremented while BH is only complemented.

Location TMP saves the carry from E while BH is complemented.

\[ z = x \oplus y = xy' + x'y = [(xy')' \cdot (x'y')']' \]

LDA Y
CMA
AND X
CMA
STA TMP
LDA X
CMA
AND Y
CMA

AND TMP
CMA
STA Z
HLT

\[ x_1, — \]
\[ y_2, — \]
\[ z_3, — \]
\[ \text{TMP}, — \]

LDA X
CLE
CIL
SIZE BUN ONE
SPA BUN OVF
BUN EXT
ONE, SNA BUN OVF
EXT, HLT

/ zero to low order bit; sign bit in E

AC(1) = 1
AC(1) = 0

= O
= E
= i

= 0
OCF

= 0
= K, O

40
Calling Program

BSA SUB
HEX 1234 /Subroutine
HEX 4321 /minuend
HEX 0 /difference

Subroutine

SUB, HEX 0
LDA SUB I
CMA
INC
ISZ SUB
ADD SUB I
ISZ SUB
STA SUB I
ISZ SUB
BUN SUB I

6-22

Calling Program

BSA CMP
HEX 100 /starting address
DEC 32 /number of words

Subroutine

CMP, HEX 0
LDA CMP I
STA PTR
ISZ CMP
LDA CMP I

6-23

CR4, HEX 0
CIR
CIR
CIR
CIR
BUN CR4 I

6-24

LDA ADS
STA PTR
LDA NBR
STA CTR

LOP, BSA IN2 /Subroutine Table 6-20
STA PTR I
ISZ PTR
ISZ CTR

BUN LOP
HLT

ADS, HEX 400
PTR, HEX 0
NBR, DEC -512
CTR, HEX 0
LDA WRD AND MS1 STA CH1 LDA WRD AND MS2 CLE BSA SR8 /subroutine to shift right eight times 

start 

Initialize memory buffer

Load next double character from buffer into AC

AND AC with HEX 00FF

Compare AC with HEX 000D

Equal?

no

yes

Pack HEX 0D to LNE

END

STA CH2 HLT

WRD, HEX - CH1, HEX - CH2, HEX - MS1, HEX 00FF MS2, HEX FF00

Load double character again

AND AC with HEX FF00

Compare AC with HEX OD0D

Equal?

no

yes
<table>
<thead>
<tr>
<th>Location</th>
<th>Hex code</th>
<th>Location</th>
<th>Hex code</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>3213</td>
<td>3213</td>
<td>200</td>
</tr>
<tr>
<td>201</td>
<td>7080</td>
<td>7080</td>
<td>201</td>
</tr>
<tr>
<td>202</td>
<td>3214</td>
<td>3214</td>
<td>202</td>
</tr>
<tr>
<td>203</td>
<td>F200</td>
<td>F200</td>
<td>203</td>
</tr>
<tr>
<td>204</td>
<td>4209</td>
<td>4209</td>
<td>204</td>
</tr>
<tr>
<td>205</td>
<td>F800</td>
<td>F800</td>
<td>205</td>
</tr>
<tr>
<td>206</td>
<td>F400</td>
<td>F400</td>
<td>206</td>
</tr>
<tr>
<td>207</td>
<td>B215</td>
<td>B215</td>
<td>207</td>
</tr>
<tr>
<td>208</td>
<td>6215</td>
<td>6215</td>
<td>208</td>
</tr>
<tr>
<td>209</td>
<td>F100</td>
<td>F100</td>
<td>209</td>
</tr>
<tr>
<td>20A</td>
<td>420E</td>
<td>420E</td>
<td>20A</td>
</tr>
<tr>
<td>20B</td>
<td>A216</td>
<td>A216</td>
<td>20B</td>
</tr>
<tr>
<td>20C</td>
<td>F400</td>
<td>F400</td>
<td>20C</td>
</tr>
<tr>
<td>20D</td>
<td>6216</td>
<td>6216</td>
<td>20D</td>
</tr>
<tr>
<td>20E</td>
<td>2214</td>
<td>2214</td>
<td>20E</td>
</tr>
<tr>
<td>20F</td>
<td>7040</td>
<td>7040</td>
<td>20F</td>
</tr>
<tr>
<td>210</td>
<td>2213</td>
<td>2213</td>
<td>210</td>
</tr>
<tr>
<td>211</td>
<td>F080</td>
<td>F080</td>
<td>211</td>
</tr>
<tr>
<td>212</td>
<td>0000</td>
<td>0000</td>
<td>212</td>
</tr>
<tr>
<td>213</td>
<td>0000</td>
<td>0000</td>
<td>213</td>
</tr>
<tr>
<td>214</td>
<td>0000</td>
<td>0000</td>
<td>214</td>
</tr>
<tr>
<td>215</td>
<td>0000</td>
<td>0000</td>
<td>215</td>
</tr>
<tr>
<td>216</td>
<td>0000</td>
<td>0000</td>
<td>216</td>
</tr>
</tbody>
</table>

6-28

SRV, STA SAC
CIR
STA SE
LDA MOD /Check MOD
CHMA
SZA
BUN NXT / MOD ≠ all 1's
SKI
BUN NXT Service Input Device
INP
OUT
STA PTI I
ISZ PTI
BUN EXT / MOD ≠ 0

NXT, LDA MOD
SZA
.BUN EXT service
output
device LDA PT2 I
OUT
EXT, continue as in Table 6-23

43
A microprocessor is a small size CPU (computer on a chip). Microprogram is a program for a sequence of microoperations. The control unit of a microprocessor can be hardwired or microprogrammed, depending on the specific design. A microprogrammed computer does not have to be a microprocessor.

Hardwired control, by definition, does not contain a control memory.

Microoperation - an elementary digital computer operation. Microinstruction - an instruction stored in control memory. Microprogram - a sequence of microinstructions. Microcode - same as a microprogram.

The frequency of each clock is $\frac{1}{100 \times 10^{-9}} = \frac{1000}{100} \times 10^6 = 10 \text{ MHz}$.

If the data register is removed, we can use a single phase clock with a frequency of $\frac{1}{90 \times 10^{-9}} = 11.1 \text{ MHz}$. 
Control memory = \(2^{16} \times 32 = 32 \text{ bits}\)

Select | Address | Microoperations

(b) 4 bits
(c) 2 bits

Control memory = \(2^{12} \times 24\)

(a) 12 bits
(b) 12 bits
(c) 12 multiplexers, each of size 4-to-1 line.

\[
\begin{align*}
0001000 & = 8 \\
0101100 & = 44 \\
0111100 & = 60
\end{align*}
\]

Op code = 6 bits
Control memory = 00xxxxxx
Address = 11 bits

\[
\begin{array}{c}
\text{inputs} \\
\hline
2^n \times m \\
\hline
\text{ROM} \\
\hline
\text{outputs}
\end{array}
\]

The ROM can be programmed to provide any desired address from a given inputs from the instruction.

Either multiplexers, three-state gates, or gate logic (equivalent to a mux) are needed to transfer information from many sources to a common destination.
7-11

(a) 011 110 000
(b) 000 100 101
(c) 100 101 000

INC AC  INC DR  NOP
NOP  READ  INC PC
DRTAC  ACTDR  NOP

7-12

(a) READ  DR< M[AR]  F2 =100
DRTAC  AC< DR  F3 =101

(b) ACTDR  DR< AC  F2 =101
DRTAC  AC< DR  F1 =100

(c) ARTPC  PC< AR  F3 =110
DRTAC  AC< DR  F1 =100
WRITE  M[AR]< DR  F1 =111

Both use F1

7-13

If I=0, the operand is read in the first microinstruction and added to AC in the second.
If I=1, the effective address is read into DR and control goes to INDR2. The subroutine must read the operand into DR.

INDR2: DRTAR  U JMP  NEXT
READ  U RET

7-14

(2) Branch if S=0 and Z=0 (positive and non-zero AC) – See last instruction in Problem 7-16.

(b) 40: 000 000 000 10 00 1000 0000
41: 000 000 000 11 00 1000 0000
42: 000 000 000 01 01 1000 0111
43: 000 100 110 00 00 1000 0000
(a) 60: CLRAC, COM U JMP INDIRECT
61: WRITE, READ I CALL FETCH
62: ADD, SUB S RET 63 (NEXT)
63: DRTAC, INCDR Z MAP 60

(b)
60: Cannot increment and complement Ac at the same time. With a JMP to INDIRECT, control does not return to 61.
61: Cannot read and write at the same time. The CALL behaves as a JMP since there is no return from FETCH.
62: Cannot add and subtract at the same time. The RET will be executed independent of S.
63: The MAP is executed irrespective of Z or 60.

7-16

ORG 16
AND: ... NOP I CALL INDIRECT
READ U JMP NEXT
ANDOP: AND U JMP FETCH

ORG 20
SUB: NOP I CALL INDIRECT
READ U JMP NEXT
SUB U JMP FETCH

ORG 24
ADD: NOP I CALL INDIRECT
READ U JMP NEXT
DRTAC, ACTDR U JMP NEXT
ADD U JMP EXCHANGE+2 (Table 7-2)
ORG 28

BTCL:

NOP   I   CALL INDRCT
READ  U   JMP NEXT
DRTAC,ACTDR U   JMP NEXT
COM   U   JMP ANDOP

ORG 32

BZ:

NOP   Z   JMP ZERO
NOP   U   JMP FETCH

ZERO:

NOP   I   CALL INDRCT
ARTPC U   JMP FETCH

ORG 36

SEQ:

NOP   I   CALL INDRCT
READ  U   JMP NEXT
DRTAC,ACTDR U   JMP NEXT
XOR (N SUB) U   JMP BEQ1

ORG 69

BEQ1:

DRTAC,ACTDR Z   JMP EQUAL
NOP   U   JMP FETCH

EQUAL:

INCPC U   JMP FETCH

ORG 40

BPNZ:

NOP   S   JMP FETCH
NOP   Z   JMP FETCH
NOP   I   CALL INDRCT
ARTPC U   JMP FETCH
ISZ: NOP I CALL indirect
READ U JMP NEXT
INCR U JMP NEXT
DRAC, ACTDR U JMP NEXT (by post indirect)
DRAC, ACTDR Z JMP ZERO
WRITE U JMP FETCH
ZERO: WRITE, INCR U JMP FETCH

B3A: NOP I CALL indirect
PCTDR, ARTPC U JMP NEXT
WRITE, INCR U JMP FETCH

From Table 7-1:
F3 = 101 (5)  PC ← PC + 1
F3 = 110 (6)  PC ← AR

FROM:
F3 output 6
F3 output 5

clock

7-20
A field of 5 bits can specify $2^5 - 1 = 31$ microoperations
A field of 4 bits can specify $2^4 - 1 = 15$ microoperations
9 bits

7-21
See Fig. 8-2(b) for control word example.

(2) 16 registers need 4 bits; ALU need 5 bits, and
the shifter need 3 bits, to encode all operations.

4 4 4 5 3 = 20 bits total

(c) 0101 0110 0100 00100 0000
R4 ← R5 + R6
7-22

<table>
<thead>
<tr>
<th>$I_2$</th>
<th>$I_1$</th>
<th>$I_0$</th>
<th>$T$</th>
<th>$S_1$</th>
<th>$S_0$</th>
<th>$L$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

7-23

(2) See Fig. 4-8 (chapter 4)

(b)

7-24

$P$ is used to determine the polarity of the selected status bit.

When $P=0$, $T=G$ because $G \oplus 0 = G$.

When $P=1$, $T=G'$ because $G \oplus 1 = G'$.

When $G$ is the value of the selected bit in MUX2.
(a) 32 multiplexers, each of size 16 x 1.
(b) 4 inputs each, to select one of 16 registers.
(c) 4-to-16-line decoder
(d) $32 + 32 + 1 = 65$ data input lines
   $32 + 1 = 33$ data output lines.
   \[
   \begin{array}{cccc}
   i_4 & i_4 & i_4 & i_6 \\
   SELA & SELB & SELD & OPR \\
   \end{array}
   \]
   \[= 18 \text{ bits} \]

8-2

$30 + 30 + 10 = 120$ nsec.

(The decoder signals propagate at the same as the muxs.)

8-3

(a) $R_1 \leftarrow R_2 + R_3$
(b) $R_4 \leftarrow R_4$
(c) $R_5 \leftarrow R_5 - 1$
(d) $R_6 \leftarrow \text{shl} R_1$
(e) $R_7 \leftarrow \text{Input}$

\[
\begin{array}{cccccc}
\text{SELA} & \text{SELB} & \text{SELD} & \text{opr} & \text{Control word} \\
R_2 & R_3 & R_1 & \text{ADD} & 010 \ 011 \ 001 \ 00010 \\
R_4 & \_ & \_ & \text{C0HA} & 100 \ \_ \_ \_ \ \_ \ 0110 \\
R_5 & R_5 & \_ & \text{DECA} & 101 \ \_ \_ \_ \ \_ \ 0010 \\
R_1 & R_6 & \_ & \text{S1LA} & 001 \ \_ \_ \_ \ \_ \ \_ \_ \_ \_ \ 110000 \\
\text{Input} & \_ & \_ & \text{TSFA} & 000 \ \_ \_ \_ \ \_ \ \_ \_ \_ \_ \ \_ \ 000000 \\
\end{array}
\]

8-4

<table>
<thead>
<tr>
<th>Control word</th>
<th>SELA SELB SELD OPR</th>
<th>Microoperation</th>
</tr>
</thead>
<tbody>
<tr>
<td>001 010 011 00101</td>
<td>R1 R2 R3 SUB</td>
<td>R3 \leftarrow R1 - R2</td>
</tr>
<tr>
<td>000 000 000 00000</td>
<td>Input Input None TSFA</td>
<td>Input \text{Input}</td>
</tr>
<tr>
<td>010 010 010 01100</td>
<td>R2 R2 R2 XOR</td>
<td>R2 \leftarrow R2 \oplus R2</td>
</tr>
<tr>
<td>000 001 000 00010</td>
<td>Input R1 None ADD</td>
<td>Output \text{Output} + R1</td>
</tr>
<tr>
<td>000 000 000 00000</td>
<td>Input R1 None ADD</td>
<td>Output \text{Output} + R1</td>
</tr>
<tr>
<td>000 000 000 00000</td>
<td>Input R1 None ADD</td>
<td>Output \text{Output} + R1</td>
</tr>
<tr>
<td>000 000 000 00000</td>
<td>Input R1 None ADD</td>
<td>Output \text{Output} + R1</td>
</tr>
<tr>
<td>000 000 000 00000</td>
<td>Input R1 None ADD</td>
<td>Output \text{Output} + R1</td>
</tr>
<tr>
<td>000 000 000 00000</td>
<td>Input R1 None ADD</td>
<td>Output \text{Output} + R1</td>
</tr>
<tr>
<td>000 000 000 00000</td>
<td>Input R1 None ADD</td>
<td>Output \text{Output} + R1</td>
</tr>
<tr>
<td>000 000 000 00000</td>
<td>Input R1 None ADD</td>
<td>Output \text{Output} + R1</td>
</tr>
<tr>
<td>000 000 000 00000</td>
<td>Input R1 None ADD</td>
<td>Output \text{Output} + R1</td>
</tr>
<tr>
<td>000 000 000 00000</td>
<td>Input R1 None ADD</td>
<td>Output \text{Output} + R1</td>
</tr>
<tr>
<td>000 000 000 00000</td>
<td>Input R1 None ADD</td>
<td>Output \text{Output} + R1</td>
</tr>
<tr>
<td>000 000 000 00000</td>
<td>Input R1 None ADD</td>
<td>Output \text{Output} + R1</td>
</tr>
<tr>
<td>000 000 000 00000</td>
<td>Input R1 None ADD</td>
<td>Output \text{Output} + R1</td>
</tr>
<tr>
<td>000 000 000 00000</td>
<td>Input R1 None ADD</td>
<td>Output \text{Output} + R1</td>
</tr>
<tr>
<td>000 000 000 00000</td>
<td>Input R1 None ADD</td>
<td>Output \text{Output} + R1</td>
</tr>
<tr>
<td>000 000 000 00000</td>
<td>Input R1 None ADD</td>
<td>Output \text{Output} + R1</td>
</tr>
<tr>
<td>000 000 000 00000</td>
<td>Input R1 None ADD</td>
<td>Output \text{Output} + R1</td>
</tr>
<tr>
<td>000 000 000 00000</td>
<td>Input R1 None ADD</td>
<td>Output \text{Output} + R1</td>
</tr>
<tr>
<td>000 000 000 00000</td>
<td>Input R1 None ADD</td>
<td>Output \text{Output} + R1</td>
</tr>
<tr>
<td>000 000 000 00000</td>
<td>Input R1 None ADD</td>
<td>Output \text{Output} + R1</td>
</tr>
<tr>
<td>000 000 000 00000</td>
<td>Input R1 None ADD</td>
<td>Output \text{Output} + R1</td>
</tr>
<tr>
<td>000 000 000 00000</td>
<td>Input R1 None ADD</td>
<td>Output \text{Output} + R1</td>
</tr>
<tr>
<td>000 000 000 00000</td>
<td>Input R1 None ADD</td>
<td>Output \text{Output} + R1</td>
</tr>
</tbody>
</table>

8-5

(a) Stack full with 64 items.
(b) Stack empty.
8-6

**PUSH:** \[ M[sp] \leftarrow DR \]
\[ sp \leftarrow sp - 1 \]

**Pop:** \[ sp \leftarrow sp + 1 \]
\[ DR \leftarrow M[sp] \]

8-7

(a) \[ AB \times CD \times EF \times ++ \]
(b) \[ AB \times ABD \times CE \times ++ + \]
(c) \[ FG + E \times CD \times + B \times A + \]
(d) \[ ABCDE \times + + + \times FGH \times + * / \]

8-8

(a) \[ \frac{A}{B-(D+E) \times C} \]
(b) \[ A + B - \frac{C}{D \times E} \]

(c) \[ \frac{A}{B \times C} - D + \frac{E}{F} \]
(d) \[ (((F+G) \times E + D) \times C + B) \times A \]

8-9

\[(3+4) \times [10(2+6)+8] = 616 \]

**RPN:** \[ 3 \ 4 \ + \ 2 \ 6 \ + \ 10 \times \ 8 \ + \ * \]

8-10

**WRITE (if not full):**
\[ M[wc] \leftarrow DR \]
\[ wc \leftarrow wc + 1 \]
\[ asc \leftarrow asc + 1 \]

**READ (if not empty):**
\[ DR \leftarrow M[rc] \]
\[ rc \leftarrow rc + 1 \]
\[ asc \leftarrow asc - 1 \]

Memory may wrap-around

---

FIFO:

\[ 10 \]
\[ 1 \]
\[ 2 \]
\[ 3 \]
\[ 4 \]
\[ 5 \]
\[ 6 \]
\[ 7 \]
8-11

\[
\text{ opcode } | \text{ Address 1} | \text{ Address 2} \quad \text{Two address instructions}
\]

\[2^8 = 256 \text{ combinations.}\]
\[2^{56} - 2^{50} = 6 \text{ combinations can be used for one address}\]

\[
\begin{array}{c|c}
\text{ opcode } & \text{ Address} \\
\hline
6 & 2^{13}
\end{array}
\]

Maximum number of one address instruction:
\[-6 \times 2^{12} = 24,576\]

8-12

(d) RPN: \(X \ A \ B - C + D \ E \times F - F \times G \ H \ K + + / =\)

8-13

\[256 \times 2^{10} = 2^{18}\]

\[
\begin{array}{c|c|c|c}
\text{ opcode } & \text{ Mode } & \text{ Register } & \text{ Address} \\
\hline
5 & 3 & 6 & 18 = 32
\end{array}
\]

8-14

\(Z = \text{Effective address}\)

(a) Direct: \(Z = Y\)
(b) Indirect: \(Z = M[y]\)
(c) Relative: \(Z = Y + W + 2\)
(d) Indexed: \(Z = Y + X\)

8-15

(a) Relative address: \(500 - 751 = -251\)
(b) \(251 = 0000\,1111\,1011; \ -251 = 1111\,0000\,0101\)
(c) \(PC = 751 = 0010\,1110\,0111; \ 500 = 0001\,1111\,0100\)
\[
\begin{align*}
\text{PC} &= 751 = 0010\,1110\,0111 \\
\text{RA} &= -251 = \cancel{1111}\,0000\,0101 \\
\text{EA} &= 500 = 0001\,1111\,0100
\end{align*}
\]

53
8-16 Assuming one word per instruction or operand.

<table>
<thead>
<tr>
<th>Computational type</th>
<th>Branch type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fetch instruction</td>
<td>Fetch instruction</td>
</tr>
<tr>
<td>Fetch effective address</td>
<td>Fetch effective address and transfer to PC</td>
</tr>
<tr>
<td>Fetch operand</td>
<td></td>
</tr>
<tr>
<td>3 memory references</td>
<td>2 memory references</td>
</tr>
</tbody>
</table>

8-17 The address part of the indexed mode instruction must be set to zero.

8-18 Effective address

- (d) Direct: 400
- (e) Immediate: 301
- (c) Relative: 302 + 400 = 702
- (d) Reg. Indirect: 200
- (e) Indexed: 200 + 400 = 600

8-19

<table>
<thead>
<tr>
<th>i</th>
<th>= 1</th>
<th>c</th>
<th>= 0</th>
<th>c</th>
<th>= 1</th>
<th>c</th>
<th>= 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>6E</td>
<td>C3</td>
<td>56</td>
<td>7A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>55</td>
<td>6B</td>
<td>8F</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>82</td>
<td>18</td>
<td>C2</td>
<td>09</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Add with carry

8-20

<table>
<thead>
<tr>
<th>10011100 AND 10101010</th>
<th>01101010 OR 10101010</th>
<th>00110110 XOR 10011100</th>
</tr>
</thead>
<tbody>
<tr>
<td>10001000</td>
<td>11111110</td>
<td>00111010</td>
</tr>
</tbody>
</table>

8-21

- (a) AND with: 0000000011111111
- (b) OR with: 0000000011111111
- (c) XOR with: 0000111111110000

54
8-22
\[ \text{Initial: } 0111011 \quad c = 1 \]
\[ \text{SHR: } 00111101 \]
\[ \text{SHL: } 11110110 \]
\[ \text{SHRA: } 00111101 \]
\[ \text{SHLA: } 11110110 \quad (\text{overflow}) \]
\[ \text{ROR: } 10111101 \]
\[ \text{ROL: } 11110110 \]
\[ \text{ROFC: } 10111101 \]
\[ \text{POCF: } 11110111 \]

8-23
\[ 183 = 01010011 \]
\[ -83 = 10101101 \]
\[ +68 = 01000100 \quad -68 = 10111100 \]
\[ (a) \quad -83 + 10101101 \]
\[ = 01000100 \quad \text{10 carries} \]
\[ +68 \quad 01000100 \]
\[ = 11100000 \quad \text{(two's complement)} \]
\[ -151 \quad 01101001 \]
\[ -128 \quad \text{(overflow)} \]

8-24
\[ Z = F_0F_1F_2F_3F_4F_5F_6F_7 = (F_0 + F_1 + F_2 + F_3 + F_4 + F_5 + F_6 + F_7) \]

8-25
\[ 72 \quad 01110010 \]
\[ 11000110 \]
\[ 138 \quad 00111100 \]
\[ c = 1 \quad s = 0 \quad z = 0 \quad v = 0 \]
\[ 11 \quad \text{(a)} \]
\[ 72 \quad 01110010 \]
\[ 1100011100 \]
\[ 90 \quad 00111100 \]
\[ c = 0 \quad s = 1 \quad z = 0 \quad v = 1 \]
\[ \text{(b)} \]
\[ 9A = 100111010 \quad 110001110 \quad \text{E} \]
\[ 00 \quad 01110010 \quad 110001110 \quad 00 \quad 00000000 \]
\[ c = 0 \quad s = 0 \quad z = 1 \quad v = 0 \]
\[ \text{(c)} \]
\[ 72 \quad 01110010 \quad 01 \quad \text{(d)} \]
\[ 08 \quad 11010010 \quad 00 \quad 00000000 \quad \text{C} \]
\[ c = 0 \quad s = 0 \quad z = 1 \quad v = 0 \]
\[ \text{(b) of row = 1} \]

55
8-26

$c = 1$ if $A < B$, therefore $c = 0$ if $A \geq B$

$z = 1$ if $A = B$, therefore $Z = 1$ if $A \neq B$

For $A > B$ we must have $A \geq B$ provided $A \neq B$

or $c = 0$ and $z = 0$ $(c'z') = 1$

For $A \leq B$ we must have $A < B$ or $A = B$

or $c = 1$ or $z = 1$ $(c+z) = 1$

8-27

$A > B$ implies that $A - B \geq 0$ (positive or zero)

sign $s = 0$ if no overflow (positive)

or $s = 1$ if overflow (sign reversal)

Boolean expression: $s'v' + sv = 1$ or $(s \oplus v) = 0$

$A < B$ is the complement of $A \geq B$ ($A - B$ negative)

then $s = 1$ if $v = 0$

or $s = 0$ if $v = 1$ $(s \oplus v) = 1$

$A > B$ implies $A \geq B$ but not $A = B$

$(s \oplus v) = 0$ and $z = 0$

$A \leq B$ implies $A < B$ or $A = B$

$s \oplus v = 1$ or $z = 1$

8-28

A Boolean circuit diagram is shown with logic gates and equations indicating relationships between $A$, $B$, $c$, $z$, $s$, and $v$. The circuit includes the following logic expressions:

- $c'z' = 1$ if $A > B$
- $c = 0$ if $A \geq B$
- $c = 1$ if $A < B$
- $c+z = 1$ if $A \leq B$
- $z = 1$ if $A = B$
- $z = 0$ if $A \neq B$
- $(s \oplus v) + z = 1$ if $A > B$
- $s \oplus v = 0$ if $A \geq B$
- $s \oplus v = 1$ if $A < B$
- $[(s \oplus v) + z] = 1$ if $A \leq B$
8-29

\[
\begin{align*}
A &= 01000001 \\
B &= 10000100 \\
A + B &= 11000101 \\
\text{Signed:} & \\
& +65 \\
& -124 \\
\text{Unsigned:} & \\
& 65 \\
& 132 \\
& 197 \\
& -59
\end{align*}
\]

(c) \( C = 0 \quad Z = 0 \quad S = 1 \quad V = 0 \)

(d) BNE  BNZ  BM  BNV

8-30

(a) \( A = 01000001 = 65 \)

\[
\begin{align*}
B &= 10000100 = 132 \\
A - B &= 10111101 = -67 \quad (2\text{'s \text{ comp. \ of } 01000011)
\end{align*}
\]

(b) \( C \text{ (borrow)} = 1 \; Z = 0 \; 65 < 132 \; A < B \)

(c) BL, BLE, BNE

8-31

(a) \( A = 01000001 = +65 \)

\[
\begin{align*}
B &= 10000100 = -124 \\
A - B &= 10111101 = +189 = \frac{01011101}{9 \text{ bits}}
\end{align*}
\]

(b) \( S = 1 \text{ (sign reversal)} \quad +189 > 127 \quad \text{Z = 0} \quad V = 1 \text{ (overflow)} \quad 65 > -124 \quad A > B \)

(c) BGT, BGE, BNE

8-32

<table>
<thead>
<tr>
<th>Initial</th>
<th>PC</th>
<th>SP</th>
<th>Top of Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1120</td>
<td>3560</td>
<td>5320</td>
</tr>
<tr>
<td>After CALL</td>
<td>6720</td>
<td>3559</td>
<td>1122</td>
</tr>
<tr>
<td>After RETURN</td>
<td>1122</td>
<td>3560</td>
<td>5320</td>
</tr>
</tbody>
</table>
Branch instruction - Branch without being able to return.

Subroutine call - Branch to subroutine and then return to calling program.

Program interrupt - Hardware initiated branch with possibility to return.

See Sec. 8-7 under "Types of Interrupts."

(a) \[ \text{SP} \leftarrow \text{SP-1} \]
    \[ \text{M}[\text{SP}] \leftarrow \text{PSW} \]
    \[ \text{SP} \leftarrow \text{SP-1} \]
    \[ \text{M}[\text{SP}] \leftarrow \text{PC} \]
    \[ \text{TR} \leftarrow \text{IAD} \quad \text{(TR is a temporary register)} \]
    \[ \text{PSW} \leftarrow \text{M}[\text{TR}] \]
    \[ \text{TR} \leftarrow \text{TR+1} \]
    \[ \text{PC} \leftarrow \text{M}[\text{TR}] \]
    "Go to fetch phase."

(b) \[ \text{PC} \leftarrow \text{M}[\text{SP}] \]
    \[ \text{SP} \leftarrow \text{SP+1} \]
    \[ \text{PSW} \leftarrow \text{M}[\text{SP}] \]
    \[ \text{SP} \leftarrow \text{SP+1} \]

Window size = \( L + 2C + G \)

Computer 1: \( 10 + 12 + 10 = 32 \)

Computer 2: \( 8 + 16 + 8 = 32 \)

Computer 3: \( 16 + 32 + 16 = 64 \)

Register file = \( (L+C)W + G \)

Computer 1: \( (10+6)8 + 10 = 16 \times 8 + 10 = 138 \)

Computer 2: \( (8+8)4 + 8 = 16 \times 4 + 8 = 72 \)

Computer 3: \( (16+16)16 + 16 = 32 \times 16 + 16 = 528 \)
8-38

(a) SUB R22, #1, R22  \( R22 \leftarrow R22 - 1 \)  (Subtract 1)
(b) XOR R22, #1, R22  \( R22 \leftarrow R22 \oplus \text{all 1's} \)  (XOR with all 1's)
(c) SUB R0, R22, R22  \( R22 \leftarrow 0 - R22 \)
(d) ADD R0, R0, R22  \( R22 \leftarrow O + O \)
(e) SRA R22, #2, R22  (Arithmetic shift right twice)
(f) OR R1, R1, R1  \( R1 \leftarrow R1 \lor R1 \)
   or ADD R1, R0, R1  \( R1 \leftarrow R1 + O \)
   or SLL R1, #0, R1  (shift left 0 times)

8-39

(a) JMP Z, #3200, (RO)  \( PC \leftarrow 0 + 3200 \)
(b) JMPIR Z, -200  \( PC \leftarrow 3400 + (-200) \)
CHAPTER 9

9-1

\[ A_i \rightarrow R1 \rightarrow \text{ADDER} \rightarrow R5 \text{ Multiplier } \rightarrow R7 \]

\[ B_i \rightarrow R2 \rightarrow \text{ADDER} \rightarrow R6 \]

\[ C_i \rightarrow R3 \]

\[ D_i \rightarrow R4 \]

9-2

<table>
<thead>
<tr>
<th>Segment</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>T_i</td>
<td>T_2</td>
<td>T_3</td>
<td>T_4</td>
<td>T_5</td>
<td>T_6</td>
<td>T_7</td>
<td>T_8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>T_1</td>
<td>T_2</td>
<td>T_3</td>
<td>T_4</td>
<td>T_4</td>
<td>T_5</td>
<td>T_6</td>
<td>T_7</td>
<td>T_8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>T_1</td>
<td>T_2</td>
<td>T_3</td>
<td>T_4</td>
<td>T_3</td>
<td>T_4</td>
<td>T_5</td>
<td>T_6</td>
<td>T_7</td>
<td>T_8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>T_1</td>
<td>T_2</td>
<td>T_3</td>
<td>T_4</td>
<td>T_3</td>
<td>T_4</td>
<td>T_5</td>
<td>T_6</td>
<td>T_7</td>
<td>T_8</td>
<td>T_8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>T_1</td>
<td>T_2</td>
<td>T_3</td>
<td>T_4</td>
<td>T_3</td>
<td>T_4</td>
<td>T_5</td>
<td>T_6</td>
<td>T_7</td>
<td>T_8</td>
<td>T_8</td>
<td>T_8</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>T_1</td>
<td>T_2</td>
<td>T_3</td>
<td>T_4</td>
<td>T_3</td>
<td>T_4</td>
<td>T_5</td>
<td>T_6</td>
<td>T_7</td>
<td>T_8</td>
<td>T_8</td>
<td>T_8</td>
<td>T_8</td>
</tr>
</tbody>
</table>

\[(k - n - 1)t_p = 6 + 8 - 1 = 13 \text{ cycles}\]

9-3

\[ k = 6 \text{ segment} \]
\[ n = 200 \text{ tasks} \]
\[ (k + n - 1) = 6 + 200 - 1 = 205 \text{ cycles} \]

9-4

\[ t_n = 50 \text{ ns} \]
\[ k = 6 \]
\[ t_p = 10 \text{ ns} \]
\[ n = 100 \]

\[ S = \frac{n t_n}{(k + n - 1)t_p} = \frac{100 \times 50}{(6 - 99) \times 10} = 4.76 \]

\[ S_{\text{max}} = \frac{t_n}{t_p} = \frac{50}{10} = 5 \]

60
9-5

(a) \( t_p = 45 + 5 = 50 \) ns \( k = 3 \)

(b) \( t_n = 40 + 45 + 15 = 100 \) ns

(c) \( S = \frac{n \cdot t_n}{(k+n-1) \cdot t_p} = \frac{10 \cdot 100}{(3+9) \cdot 50} = 1.67 \quad \text{for} \ n = 10 \)

\[ \frac{100 \cdot 100}{(3+99) \cdot 50} = 1.96 \quad \text{for} \ n = 100 \]

(d) \( S_{\text{max}} = \frac{t_n}{t_p} = \frac{100}{50} = 2 \)

9-6

(a) See discussion in Sec. 10-3 on array multipliers. There are \( 8 \times 8 = 64 \) AND gates in each segment and an 8-bit binary adder (in each segment).

(b) There are 7 segments in the pipeline.

(c) Average time = \( \frac{k+n-1}{n} \cdot t_p = (n+6) \cdot 30 \)

For \( n = 10 \) \( t_{AV} = 48 \) ns

For \( n = 100 \) \( t_{AV} = 31.8 \) ns

For \( n \to \infty \) \( t_{AV} = 30 \) ns

To increase the speed of multiplication, a carry-save (Wallace tree) adder is used to reduce the propagation time of the carries.

9-7

(a) Clock cycle = \( 95 + 5 = 100 \) ns (time for segment 3)

For \( n = 100 \), \( k = 4 \), \( t_p = 100 \) ns.

Time to add 100 numbers = \( (k+n-1) \cdot t_p = (4+99) \cdot 100 \)
\[ = 10,300 \text{ns} = 10.3 \mu \text{s} \]

(b) Divide segment 3 into two segments of 50 + 5 = 55 and 45 + 5 = 50 ns. This makes \( t_p = 55 \) ns; \( k = 5 \)

\( (k+n-1) \cdot t_p = (5+99) \cdot 55 = 5,720 \text{ns} = 5.72 \mu \text{s} \)
Connect output of adder to input Bx2 in a feedback path and use input A*x for the data X1 through X100. Then use a scheme similar to the one described in conjunction with the adder pipeline in Fig. 9-12.

One possibility is to use the six operations listed in the beginning of Sec. 9-4.

See Sec. 9-4: (a) prefetch target instruction; (b) use a branch target buffer; (c) use a loop buffer; (d) use branch prediction. (Delayed branch is a software procedure.)

1. Load R1 ← M[312]
2. Add R2 ← R2 + M[313]
3. Increment R3
4. Store M[314] ← R3

Segment EX: Transfer memory word to R1.
Segment FO: Read M[313].
Segment DA: Decode (increment) instruction.
Segment FI: Fetch (the store) instruction from memory.

Load: R1 ← Memory
Increment: R1 ← R1 + 1

R1 is loaded in E
It's too early to increment it in A

Insert a No-op instruction between the two instructions in the example of Problem 9-12 (above).
9-14
101 Add R2 to R3
102 Branch to 104
103 Increment R1
104 Store R1

1 2 3 4 5 6 7
IAE
IAE
--↓
IAE

9-15 Use example of Problem 9-14.
101 Branch to 105
102 Add R2 to R3
103 No-operation
104 Increment R1
105 Store R1

IAE
IAE
IAE
IAE

9-16
(a) There are 40 product terms in each inner product.
40^2 = 1,600 inner products must be evaluated, one
for each element of the product matrix.
(b) 40^3 = 64,000

9-17
8 + 60 + 4 = 72 clock cycles for each inner product.
There are 60^2 = 3600 inner products. Product matrix
takes 3600 x 72 = 259,200 clock cycles to evaluate.

9-18
memory array 1 use addresses: 0, 4, 8, 12, ..., 1020
Array 2: 1, 5, 9, 13, ..., 1021
Array 3: 2, 6, 10, ..., 1022
Array 4: 3, 7, 11, ..., 1023

9-19
\[
\frac{250 \times 10^9}{10^6} = 2,500 \text{ sec} = 41.67 \text{ minutes}
\]

9-20
Divide the 400 operations into each of the four
processors. Processing time is: \( \frac{400 \times 40}{4} = 4,000 \text{ nsec} \)

Using a single pipeline, processing time is = \( 400 \times 10 = 4,000 \text{ nsec} \)
26 - 1 = 63, Overflow if sum greater than 63.

(a) (+45) + (+31) = 76
(b) (-31) + (-45) = -76
(c) (+45) - (+31) = 14
(d) (+45) - (-45) = 0
(e) (-31) - (+45) = -76
\[ \begin{align*}
(a) & \quad +35 \quad 0 \quad 100011 \quad (b) & \quad -35 \quad 1 \quad 011101 \\
+40 \quad 0 \quad 101000 \quad & \quad -40 \quad 1 \quad 011000 \\
+75 \quad 1 \quad 001011 \quad & \quad -70 \quad 0 \quad 110101 \\
F = 0 \quad E = 1 & \quad \text{carries} & \quad F = 1 \quad E = 0 \\
F \oplus E = 1 \; \text{overflow} & \quad F \oplus E = 1 \; \text{overflow}
\end{align*} \]

\[ \begin{array}{|c|c|c|}
\hline
\text{case} & \text{operation in sign-magnitude} & \text{operation in sign-2's complement} \\
\hline
1. & (+ X) + (+ Y) & (0+X)+(0+Y) \\
2. & (+ X) + (- Y) & (0+X)+2^k(2^k-Y) \\
3. & (- X) + (+ Y) & 2^k(2^k-X)+(0+Y) \\
4. & (- X) + (- Y) & (2^k+2^k-X)+(2^k+2^k-Y) \\
\hline
\end{array} \]

It is necessary to show that the operations in column (b) produce the results listed in column (c).

**Case 1.** column (b) = column (c)

**Case 2.** If \( X \geq Y \) then \((X-Y) > 0 \) and consists of \( k \) bits,

operation in column (b) gives: \( 2^k+(X-Y) \). Discard carry \( 2^k \) to get \( 0+(X-Y) \) as in column (c).

If \( X < Y \) then \((Y-X) > 0 \). Operation gives \( 2^k+2^k-(Y-X) \) as in column (c).

**Case 3.** is the same as case 2 with \( X \) and \( Y \) reversed.

**Case 4.** Operation in column (b) gives: \( 2^k+2^k+2^k-(X-Y) \).

Discard carry \( 2^k \) to obtain result of (c): \( 2^k +(2^k-X-Y) \).
Transfer Augend sign into $T_s$. Then add: $AC = AC + BR$

As will have sign of sum.

Truth Table for combin. circuit

<table>
<thead>
<tr>
<th>$T_s$</th>
<th>$B_s$</th>
<th>$A_s$</th>
<th>$V$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Boolean function for circuit:

$V = T_s B_s A_s + T_s B_s A_s'$

10-6 (a)

-9 1 0110
-6 1 1001
-15 0 1111

$F=1$, $E=0$ ← Carries

$E \oplus F=1$ but there should be no overflow since result is -15

(b) The procedure $V = E \oplus F$ is valid for 1's complement numbers provided we check the result $0 \ldots 11$ when $V=1$. Algorithm $AC = AC + BR$ $V = E \oplus F$

10-7 Add algorithm flowchart is shown above (Prob.10-6b)
Maximum value of numbers is \( r^n \). It is necessary to show that maximum product is less than or equal to \( r^{2n-1} \). Maximum product is:

\[
(\underbrace{r \ldots r}_{n-1}) (\underbrace{r \ldots r}_{n-1}) = r^{2n} - 2r^n + 1 \leq r^{2n-1}
\]

which gives: \( 2 \leq 2r^n \) or \( 1 \leq r^n \)

This is always true since \( r \geq 2 \) and \( n \geq 1 \)

Multiplier: \( B = 11111 = (31)_{10} \)

\[ 31 \times 21 = 651 \]

Multiplicand: \( B = 11111 = (31)_{10} \)

\[ 31 \times 21 = 651 \]

\[ \frac{10100011}{1011} = 1110 + \frac{1001}{1011} \quad \frac{163}{11} = 14 + \frac{9}{11} \]

\[ B = 1011 \quad \overline{B+1} = 0101 \quad DVF = 0 \]

Dividend in \( AQ \):

\[ \begin{array}{cccc}
\text{Dividend in } AQ & - & \text{E} & \text{A} & \text{Q} & \text{SC} \\
\text{shl } \text{EAQ} & - & - & - & 0 & 1010 & 0011 & 100 \\
\text{add } \overline{B+1}, \text{suppress carry} & - & - & - & 0 & 1000 & 0110 & 100 \\
\text{E=1, set } Q_n \text{ to 1} & - & - & - & 1 & 1001 & 0111 & 011 \\
\text{shl } \text{EAQ} & - & - & - & - & 0010 & 1110 & - \\
\text{add } \overline{B+1}, \text{suppress carry} & - & - & - & 0 & 1010 & 1111 & 010 \\
\text{E=1, set } Q_n \text{ to 1} & - & - & - & 1 & 0111 & 1110 & 010 \\
\text{shl } \text{EAQ} & - & - & - & - & 0111 & 1110 & 010 \\
\text{add } \overline{B+1}, \text{carry to } E & - & - & - & 0 & 1010 & 1111 & 001 \\
\text{E=0, leave } Q_n = 0 & - & - & - & 0 & 1000 & 1110 & 000 \\
\text{add } B & - & - & - & - & 1010 & 1110 & 000 \\
\text{restore remainder} & - & - & - & 1 & 1001 & 1110 & 000 \\
\text{remainder} & - & - & - & & & 1010 & 000 \\
\text{quotient} & - & - & - & & & & \\
\end{array} \]
$10\cdot 10(b) \quad \begin{array}{c} \hline \text{Dividend in Q, } A = 0 \quad E \quad A \quad Q \quad \text{SC} \\
\hline \text{shl} \ AESQ \quad 0101 \\
\hline \text{add B+1} \quad 0011 \\
\hline \end{array}$

$E = 0, \text{ leave } Q_n = 0 \quad 0011 \\
\text{add B} \quad 0011 \\
\text{restore partial remainder} \quad 0011 \\
\text{shl} \ AESQ \quad 0011 \\
\text{add B+1} \quad 0011 \\
\text{E=1, set Q_n to } 1 \quad 0011 \\
\text{shl} \ AESQ \quad 0011 \\
\text{add B+1} \quad 0011 \\
\text{E=0, leave } Q_n = 0 \quad 1110 \\
\text{add B} \quad 0011 \\
\text{restore partial remainder} \quad 0011 \\
\text{shl} \ AESQ \quad 0011 \\
\text{add B+1} \quad 0011 \\
\text{E=1, set Q_n to } 1 \quad 1010 \\
\text{remainder} \quad 0000 \\
\text{quotient} \quad 0101 \\
\text{000}$

$10\cdot 11$

$A + B + 1$ performs: $A + 2^B = 2^A + A - B$

adding $B$: $(2^A + A - B) + B = 2^A + A$

remove end-carry $2^A$ to obtain $A$.

$10\cdot 12$

To correspond with correct result, In general:

$$\frac{A}{B} = Q + \frac{R}{B}$$

Where $A$ is dividend, $Q$ the quotient and $R$ the remainder.

Four possible signs for $A$ and $B$:

$$\frac{+52}{+5} = +10 + \frac{+2}{+5} = +10.4$$

$$\frac{-52}{-5} = -10 + \frac{-2}{-5} = -10.4$$

The sign of the remainder ($2$) must be same as sign of dividend ($52$).

$10\cdot 13$

Add one more stage to Fig. 10-10 with 4 AND gates and a 4-bit adder.
(a) \((-15) \times (+13) = +195 = (0.0110000011)_2\)

\[
\text{BR} = 0 \hspace{1cm} \text{BR}+1 = 10001 \hspace{1cm} \overline{GR} = 01101 \hspace{1cm} (+15)
\]

<table>
<thead>
<tr>
<th>Qn</th>
<th>Qn+1</th>
<th>(AC)</th>
<th>QR</th>
<th>Qn+1</th>
<th>SC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>000000</td>
<td>01101</td>
<td>C</td>
<td>101</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Subtract BR</td>
<td>10001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>10000</td>
<td>10110</td>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ASHR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>01111</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>ASHR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Subtract BR</td>
<td>10001</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>ASHR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>11010</td>
<td>01101</td>
<td>1</td>
<td>010</td>
</tr>
<tr>
<td>11</td>
<td></td>
<td>ASHR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>11101</td>
<td>00110</td>
<td>1</td>
<td>001</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ASHR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>01</td>
<td></td>
<td>01111</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>ASHR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>01100</td>
<td>00011</td>
<td>0</td>
<td>000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ASHR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>+ 195</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(b) \((+15) \times (-13) = -195 = (1100111101)_2\)

\[
\text{BR} = 0 \hspace{1cm} \text{BR}+1 = 10001 \hspace{1cm} \overline{GR} = 10011 \hspace{1cm} (-13)
\]

<table>
<thead>
<tr>
<th>Qn</th>
<th>Qn+1</th>
<th>(AC)</th>
<th>QR</th>
<th>Qn+1</th>
<th>SC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>000000</td>
<td>10011</td>
<td>O</td>
<td>101</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Subtract BR</td>
<td>10001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td>ASHR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>11101</td>
<td>01100</td>
<td>1</td>
<td>011</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ASHR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>01111</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>ASHR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>00101</td>
<td>10110</td>
<td>0</td>
<td>010</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ASHR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>00010</td>
<td>11011</td>
<td>0</td>
<td>001</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>ASHR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Subtract BR</td>
<td>10011</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>ASHR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>11001</td>
<td>11101</td>
<td></td>
<td>000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ASHR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>-195</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Dividend in AQ
Divisor in B

Qs ← As + Bs
SC ← n - 1

EA ← A + B + 1

E = 1
O = 0

EA ← A + B
DVF ← 1

END
(Divide overflow)

shl EAQ
Qn ← E

= 0
Qn = 1

EA ← A + B
EA ← A + B + 1

SC ← SC - 1

SC ≠ 0
SC = 0

shl Q, Qn ← E

= 0
= 1

A ← A + B

END
Quotient is in Q
Remainder is in A
The algorithm for square-root is similar to division with the radicand being equivalent to the dividend and a "test value" being equivalent to the divisor.

Let \( A \) be the radicand, \( Q \) the square-root, and \( R \) the remainder such that \( Q^2 + R = A \) or:

\[
\sqrt{A} = Q \quad \text{and a remainder}
\]

**General comments:**

1. For \( k \) bits in \( A \) (\( k \) even), \( Q \) will have \( k/2 \) bits:

\[
Q = q_1q_2q_3 \cdots q_{k/2}
\]

2. The first test value is 01

   The second test value is 0001

   The third test value is 000001

   The fourth test value is 00000001

   etc.

3. Mark the bits of \( A \) in groups of two starting from left.

4. The procedure is similar to the division restoring method as shown in the following example:

\[
\begin{array}{cccccc}
9_1 & 9_2 & 9_3 & 9_4 \\
1 & 1 & 0 & 1
\end{array}
= Q = 13
\]

\[
\begin{array}{cccccc}
\sqrt{10} & 10 & 10 & 01
\end{array}
= A = 169
\]

\[
\begin{array}{cccccc}
01 & 0110 & 0101 & 0001 & 000110 & 000011001
\end{array}
\]

subtract first test value 01

Answer positive; let \( q_1 = 1 \)

bring down next pair

subtract second test value 0001

Answer positive; let \( q_2 = 1 \)

bring down next pair

subtract third test value 000001

Answer negative; let \( q_3 = 0 \)

restore partial remainder

bring down next pair

subtract fourth test value 00000001

Answer positive (zero); let \( q_4 = 1 \)

Remainder = 000001
10-17 (a) $e = \text{exponent}$, $e+64 = \text{biased exponent}$

<table>
<thead>
<tr>
<th>$e$</th>
<th>$e+64$</th>
<th>biased exponent</th>
</tr>
</thead>
<tbody>
<tr>
<td>-64</td>
<td>-64 + 64 = 0</td>
<td>0 000 000</td>
</tr>
<tr>
<td>-63</td>
<td>-63 + 64 = 1</td>
<td>0 000 001</td>
</tr>
<tr>
<td>-62</td>
<td>-62 + 64 = 2</td>
<td>0 000 010</td>
</tr>
<tr>
<td>-1</td>
<td>-1 + 64 = 63</td>
<td>0 111 111</td>
</tr>
<tr>
<td>0</td>
<td>0 + 64 = 64</td>
<td>1 000 000</td>
</tr>
<tr>
<td>+1</td>
<td>1 + 64 = 65</td>
<td>1 000 001</td>
</tr>
<tr>
<td>+62</td>
<td>62 + 64 = 126</td>
<td>1 111 110</td>
</tr>
<tr>
<td>+63</td>
<td>63 + 64 = 127</td>
<td>1 111 111</td>
</tr>
</tbody>
</table>

(6) The biased exponent follows the same algorithm as a magnitude comparator. See Sec. 7-2.

(c) $(e_1 + 64) + (e_2 + 64) = (e_1 + e_2 + 64) + 64$

subtract 64 to obtain biased exponent sum.

(d) $(e_1 + 64) - (e_2 - 64) = e_1 + e_2$

add 64 to obtain biased exponent difference.

10-18

(a) $AC = A_5 A_4 A_3 A_2 A_1 A_0 ... A_n$

$BS = B_5 B_4 B_3 B_2 B_1 ... B_n$

If signs are unlike—the one with a 0 (plus) is larger.
If signs are alike—both numbers are either positive or negative.

```
start
  =10 A5 B5 =01
  A<B =11 =00 A>B
  both negative both positive

(-A) - (-B) = B - A

(B-A) > 0 if B > A
(B-A) < 0 if B < A

AC <= AC + BR + 1
AC <= AC + BR + 1

=1 A5 =0
≠0 A =0
A>B A<B A=B

=1 A5 =0
≠0 A =0
A>B A<B A=B
```

72
10-18

\[
\begin{align*}
A_{s} & \quad A_{1} \quad A_{2} \ldots \quad A_{n} \\
+2 & \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 10 \\
+1 & \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 01 \\
0 & \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 00 \\
-1 & \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 11 \\
-2 & \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 00 \\
-3 & \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 01
\end{align*}
\]

10-19

\[
A = \frac{A_{c}}{A_{1} \quad A_{2} \quad A_{3} \ldots \quad A_{n}} \\
B = \frac{B_{c}}{B_{1} \quad B_{2} \quad B_{3} \ldots \quad B_{n}}
\]

(a) \(= 10\) \(\vdots\) \(= 01\)

(b) \(= 10\) \(\vdots\) \(= 01\)

10-20

Fig 10-8

mantissa alignment

\[
\begin{align*}
SC & \leftarrow n-1 \\
\text{a} \leq b & \rightarrow a : b \\
\text{shr A} & \leftarrow a - 1 \\
\text{shr B} & \leftarrow b - b + 1 \\
SC & \leftarrow sc - 1 \\
\neq 0 & \rightarrow END \leftarrow sc = sc - 1
\end{align*}
\]

AC \leftarrow BR

add \rightarrow OP

subtract

AC \leftarrow \overline{AC}

END
10-21 Let \( e \) be a flip-flop that holds end-carry after exponent addition.

```
start
  check for zeros
  \( B_s \leftarrow \overline{B_s} \)
  add
  \( e_a \leftarrow a + \overline{e} + 1 \)
  = 0 \quad = \overline{1}
  a < b \quad a > b
  \ne 0 \quad a = 0
  restore \quad \downarrow
  a \leftarrow a + b
  swap \quad \downarrow
  BR \leftarrow AC
  AC \leftarrow \overline{BR}
  a \leftarrow a + b + 1
  a < b
```

a contains \( a \)'s complement of difference
b contains larger exponent

```
shr A
  a \leftarrow a + 1

\ne 0 \quad a = 0
```

\( a < b \quad \downarrow
\)

\( a \leftarrow a \quad \downarrow
\)

add mantissas
and normalize

10-22 when 2 numbers of \( n \) bits each are multiplied, the product is no more than \( 2n \) bits long—see Prob. 9-7.

10-23 \( \frac{\text{dividend}}{\text{divisor}} = \frac{A}{B} = 0.1xxxx \) where \( x = 0,1 \)

(a) If \( A < B \) then after shift we have \( A = 1.xxxx \) and 1st quotient bit is 21.

(b) If \( A \geq B \), dividend alignment results in \( A = 0.01xxxx \) then after the left shift \( A > B \) and first quotient bit = 1.
10-24

\[
\frac{\text{dividend}}{\text{divisor}} = \frac{-1 \times \text{xxxx} + 2^{e_1}}{-1 \times \text{yyyy} \times 2^{e_2}} = 1 \times \text{yyyy} + 2^{e_1-e_2} + \frac{\text{remainder} \times 2^{e_2}}{\text{yyyy} \times 2^{e_2}}
\]

Remainder bits rrrrr have a binary-point (n-1) bits to the left.

Fig. 10-10 after dividend alignment

\[
\begin{align*}
q &= 3 \\
2 &= a+b+1 \\
a &= a + b + c \\
q &= a \\
a &= a - 1
\end{align*}
\]

10-25

(a) When the exponents are added or incremented
(b) When the exponents are subtracted or decremented
(c) Check end-carry after addition and carry after increment or decrement.

10-26

Assume integer mantissa of n-1=5 bits (excluding sign)

(a) Product: \[A \times Q = \text{xxxx} \times \text{xxxx} \times 2^3\]

Product in AC: \[\text{xxxx} \times 2^{3+5}\]

(b) Single precision normalized dividend: \[\text{xxxx} \times 2^3\]

Dividend in AQ: \[A = \text{xxxx} \times 00000, \times 2^{3-5}\]

10-27

Neglect Be and Ae from Fig. 10-14. Apply carry directly to E.

<table>
<thead>
<tr>
<th>B.</th>
<th>10^3</th>
<th>10^2</th>
<th>10^1</th>
<th>10^0</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>E</th>
<th>BCD arith. Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>As</td>
<td>10^3</td>
</tr>
<tr>
<td>----</td>
<td>------</td>
</tr>
<tr>
<td>A</td>
<td></td>
</tr>
</tbody>
</table>
\[ \begin{array}{c}
\frac{673}{356} = 1.881 \\text{ carry } 1
\end{array} \]

**10-28**

\[ 10\text{'s comp. of } 356 = 644 + \]

**10-29**

\[ \begin{array}{c}
M = 1
\end{array} \]

**10-30**

<table>
<thead>
<tr>
<th>Inputs $B_8 B_4 B_2 B_1$</th>
<th>Outputs $X_8 X_4 X_2 X_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0</td>
<td>1 0 0 1</td>
</tr>
<tr>
<td>1 0 0 1</td>
<td>1 0 0 0</td>
</tr>
<tr>
<td>2 0 1 0</td>
<td>0 1 1 1</td>
</tr>
<tr>
<td>3 0 1 1</td>
<td>0 1 1 0</td>
</tr>
<tr>
<td>4 1 0 0</td>
<td>0 1 0 1</td>
</tr>
<tr>
<td>5 1 0 1</td>
<td>0 1 0 0</td>
</tr>
<tr>
<td>6 1 1 0</td>
<td>0 0 1 1</td>
</tr>
<tr>
<td>7 1 1 1</td>
<td>0 0 1 0</td>
</tr>
<tr>
<td>8 1 0 0 0</td>
<td>0 0 0 1</td>
</tr>
<tr>
<td>9 1 0 0 1</td>
<td>0 0 0 0</td>
</tr>
</tbody>
</table>

\[ d(B_8 B_4 B_2 B_1) = \Sigma (10, 11, 12, 13, 14, 15) \]

are don't-care conditions

\[ X_8 = B_8 B_4 B_2 B_1 \]

\[ X_4 = B_4 B_2 + B_4 B_2 \]

\[ X_2 = B_2 \]

\[ X_1 = B_1 \]
The excess-3 code is self-complementing code. Therefore, to get 9's complement we need to complement each bit.

\[
M = 0 \quad \text{for} \quad x = B \\
M = 1 \quad \text{for} \quad x = 9's \text{ comp. of } B
\]

\[
M \quad B_i \quad x_i = B_i \oplus M
\]

\[
\begin{array}{c|c|c}
0 & 0 & 0 \\
0 & 1 & 1 \\
1 & 0 & 1 \\
1 & 1 & 0 \\
\end{array}
\]

10-31. The excess-3 code is self-complementing code. Therefore, to get 9's complement we need to complement each bit.

\[
M = 0 \quad \text{for} \quad x = B \\
M = 1 \quad \text{for} \quad x = 9's \text{ comp. of } B
\]

\[
M \quad B_i \quad x_i = B_i \oplus M
\]

\[
\begin{array}{c|c|c}
0 & 0 & 0 \\
0 & 1 & 1 \\
1 & 0 & 1 \\
1 & 1 & 0 \\
\end{array}
\]
Algorithm is similar to flow chart of Fig. 10-2

\[ 10-3^b \ (a) \quad \overline{B} = 410 \]

\[
\begin{align*}
\text{initial} & \quad \overline{Ae} \quad \overline{A} \quad \overline{Q} \quad \overline{SC} \\
Q_L \neq 0 & \quad 0 \quad 470 \quad 151 \quad 3 \\
Q_L \neq 0 & \quad 0 \quad 940 \quad 150 \\
Q_L = 0, dshr & \quad 0 \quad 094 \quad 015 \quad 2 \\
Q_L \neq 0 & \quad 0 \quad 564 \quad 014 \\
Q_L \neq 0 & \quad 1 \quad 034 \quad 013 \\
& \quad 1 \quad 504 \quad 012 \\
& \quad 2 \quad 444 \quad 010
\end{align*}
\]

\[ = \overline{Q_L} \]

\[
\begin{align*}
Q_L = 0, dshr & \quad 0 \quad 244 \quad 401 \quad 1 \\
Q_L \neq 0 & \quad 0 \quad 714 \quad 400 \\
Q_L = 0, dshr & \quad 0 \quad 071 \quad 440 \quad 0
\end{align*}
\]

\[
\begin{align*}
\text{Product} & = 199801
\end{align*}
\]

\[ (b) \]

\[
\begin{align*}
999 \\ \times 199 \\
89910 \\
98901 \\
+99900 \\
198801
\end{align*}
\]

- first partial product \( Ae = 8 \)
- second partial product \( Ae = 9 \)
- final product \( Ae = 1 \)
At the termination of multiplication we shift right the content of \( A \) to get zero in \( Ae \).

At the termination of division, \( B \) is added to the negative difference. The negative difference is in 10's complement so \( Ae = 9 \). Adding \( B_e = 0 \) to \( Ae = 9 \) produces a carry and makes \( Ae = 0 \).

Change the symbols as defined in Table 10-1 and use same algorithms as in Sec. 10-4 but with multiplication and division of mantissas as in Sec. 10-5.
11-1 \[
\begin{align*}
12 &= 0000011000 \\
13 &= 0000011011 \\
14 &= 0000011100 \\
15 &= 0000011111 \\
&\quad \text{To CS, RST, RS0}
\end{align*}
\]

11-2

<table>
<thead>
<tr>
<th>Interface</th>
<th>Port A</th>
<th>Port B</th>
<th>Control Reg</th>
<th>Status Reg</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>1000 0000</td>
<td>1000 0000</td>
<td>1000 0010</td>
<td>1000 0011</td>
</tr>
<tr>
<td>2</td>
<td>0100 0000</td>
<td>0100 0000</td>
<td>0100 0010</td>
<td>0100 0011</td>
</tr>
<tr>
<td>3</td>
<td>0010 0000</td>
<td>0010 0000</td>
<td>0010 0010</td>
<td>0010 0011</td>
</tr>
<tr>
<td>4</td>
<td>0001 0000</td>
<td>0001 0000</td>
<td>0001 0010</td>
<td>0001 0011</td>
</tr>
<tr>
<td>5</td>
<td>0000 1000</td>
<td>0000 1000</td>
<td>0000 1010</td>
<td>0000 1011</td>
</tr>
<tr>
<td>6</td>
<td>0000 0100</td>
<td>0000 0100</td>
<td>0000 0110</td>
<td>0000 0111</td>
</tr>
</tbody>
</table>

11-3

- Character printer
- Line printer
- Laser printer
- Digital plottter
- Graphic display
- Voice output
- Digital to analog converter
- Instrument indicator

11-5

See text discussion in See. 11-2.

11-6

(a) Status command - Checks status of flag bit.
(b) Control command - Moves magnetic head in disk.
(c) Status command - Checks if device power is on.
(d) Control command - Moves paper position.
(e) Data input command - Reads value of a register
11-7

(a) CPU data bus
  "READ" I/O

I/O data bus
  I/B

Interface
  STB

Unit

I/O
  device

(b) I/O data bus
  STB

Valid data

I/B

"READ"

(c) I/O device (source)

Place data on bus and set STB low

Accept data from bus and set
IBF High

"READ" signal from
CPU changes IBF to low

If IBF is High then input data from interface

11-8

20 MHz = 20 x 10^6 Hz

T = \frac{10^{-6}}{20} = 50 \text{ nsec.}

clock

50 nsec

Address

READ strobe

40 nsec

Data from memory

WRITE strobe

DATA for memory

Data for memory
Registers refer to Fig. 11-8. Output flag is a bit in status register.

1. Output flag to indicate when transmitter register is empty.
2. Input flag to indicate when receiver register is full.
3. Enable interrupt if any flag is set.
4. Parity error; (5) Framing error; (6) Overrun error.

11-11
10 bits: Start bit + 7 ASCII + parity + stop bit.
From Table 11-1 ASCII W = \( \overline{101011} \)
with even parity = \( \overline{1101011} \)
with start 2nd stop bits = \( 1110101110 \)
11-12
(a) \( \frac{1200}{8} = 150 \) characters per second (cps)
(b) \( \frac{1200}{11} = 109 \) cps
(c) \( \frac{1200}{10} = 120 \) cps

11-13
(a) \( \frac{k \text{ bytes}}{(m-n) \text{ bytes/sec}} = \frac{k}{m-n} \text{ sec.} \)
(b) \( \frac{k}{n-m} \text{ sec.} \)
(c) No need for FIFO

11-14

Initial
After delete = 1
After delete = 0
After insert = 1
(Insert goes to 0)

\[ F = 0011 \quad \text{Output} \leftrightarrow R4 \]
\[ F = 0100 \quad R4 \leftrightarrow R3 \]
\[ F = 1001 \quad R1 \leftrightarrow \text{Input} \]
\[ F = 0101 \quad R2 \leftrightarrow R1 \]
\[ F = 0011 \quad R3 \leftrightarrow R2 \]

11-15
(a) Empty buffer
(b) Full buffer
(c) Two items

Input ready output ready \( F_1 - F_4 \)
\[ \begin{array}{ccc}
1 & 0 & 0000 \\
0 & 1 & 1111 \\
1 & 1 & 0011 \\
\end{array} \]

11-16

Flag = 0 if data register full (After CPU writes data)
Flag = 1 if data register empty (After the transfer to device)
When flag goes to 0, enable "Data ready" and place data on I/O bus. When "Acknowledge" is enabled, set the flag to 1 and disable "ready" handshake line.
11-17 CPU Program flow chart:

11-18 See text Section 11-4.

11-19 If an interrupt is recognized in the middle of an instruction execution, it is necessary to save all the information from control registers in addition to processor registers. The state of the CPU to be saved is more complex.

11-20

1. Initially, device 2 sends an interrupt request:  
   
   \[
   \begin{array}{c}
   \text{Device 1} \\
   PI=0; PD=0; RF=0 \\
   \end{array}
   \quad \begin{array}{c}
   \text{Device 2} \\
   PI=0; PD=0; RF=1 \\
   \end{array}
   \]

2. Before CPU responds with acknowledge, device 1 sends interrupt request:
   
   \[
   \begin{array}{c}
   \text{Device 1} \\
   PI=0; PD=0; RF=1 \\
   \end{array}
   \quad \begin{array}{c}
   \text{Device 2} \\
   PI=0; PD=0; RF=1 \\
   \end{array}
   \]

3. After CPU sends an acknowledge, \( PI=1; PD=0; RF=1 \) device 1 has priority:  
   
   \[
   \begin{array}{c}
   \text{Device 1} \\
   \text{VAD enable} = 1 \\
   \end{array}
   \quad \begin{array}{c}
   \text{Device 2} \\
   \text{VAD enable} = 0 \\
   \end{array}
   \]
Table 11-2

<table>
<thead>
<tr>
<th>X0</th>
<th>I1</th>
<th>I2</th>
<th>I3</th>
<th>XY</th>
<th>IST</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>00</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>01</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>XX</td>
<td>0</td>
</tr>
</tbody>
</table>

Map simplification

\[ X = I_0'I_1' \]
\[ Y = I_0'I_1 + I_0'I_2' \]

Same as Fig. 11-14. Needs 8 AND gates and an 8 x 3 decoder.

11-24

<table>
<thead>
<tr>
<th>X0</th>
<th>I2</th>
<th>I3</th>
<th>I4</th>
<th>I5</th>
<th>I6</th>
<th>I7</th>
<th>XY3</th>
<th>IST (E)</th>
<th>Binary</th>
<th>Hexadecimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>000</td>
<td>1</td>
<td>10100000</td>
<td>AD</td>
</tr>
<tr>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>000</td>
<td>1</td>
<td>10100100</td>
<td>A4</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>010</td>
<td>1</td>
<td>10101000</td>
<td>A8</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>100</td>
<td>1</td>
<td>10101100</td>
<td>A8</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>110</td>
<td>1</td>
<td>10110000</td>
<td>B0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>110</td>
<td>1</td>
<td>10110100</td>
<td>B0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>111</td>
<td>1</td>
<td>10111000</td>
<td>B0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>111</td>
<td>1</td>
<td>10111100</td>
<td>B0</td>
</tr>
</tbody>
</table>

11-25

76 = \((01001100)\)_2
Replace the six 0's by \(010011\), XY

11-26

Set the mask bit belonging to the interrupt source so it can interrupt again.
At the beginning of the service routine, check the value of the return address in the stack. If it is an address within the source service program, then the same source has interrupted again while being serviced.

11-21

The service routine checks the flags in sequence to determine which one is set. The first flag that is checked has the highest priority level. The priority level of the other sources corresponds to the order in which the flags are checked.
When the CPU communicates with the DMA controller, the read and write lines are used as inputs from the CPU to the DMA controller.

When the DMA controller communicates with memory, the read and write lines are used as outputs from the DMA to memory.

(a) CPU initiates DMA by transferring:
   256 to the word count register,
   1230 to the DMA address register.
   Bits to the control register to specify a write operation.

(b) 1. I/O device sends a "DMA request."
   2. DMA sends BR (bus request) to CPU.
   3. CPU responds with a BG (bus grant).
   4. Contents of DMA address register are placed in address bus.
   5. DMA sends "DMA acknowledge" to I/O device and enables the write control line to memory.
   6. Data word is placed on data bus by I/O device.
   7. Increment DMA address register by 1 and decrement DMA word count register by 1.
   8. Repeat steps 4-7 for each data word transferred.

CPU refers to memory on the average once (or more) every 1 μsec. (1/10⁶). Characters arrive one every 1/2400 = 416.6 μsec. Two characters of 8 bits each are packed into a 16-bit word every 2 x 416.6 = 833.3 μsec. The CPU is slowed down by no more than (833.3) x 100 = 0.12 %.

The CPU can wait to fetch instructions and data from memory without any damage occurring except loss of time. DMA usually transfers data from a device that cannot be stopped since information continues to flow so loss of data may occur.
11-31  
**CPU operations**

- Send START I/O instruction to channel
- CPU continues with another program
- Check status word in memory location 64 for correct transfer

**I/O channel operations**

- Access memory location 72 for a channel address word (CAW)
- Use CAW to access command words from memory and conduct I/O transfers
- When I/O transfer is completed, the channel stores a status word in location 64 and interrupts the CPU.

11-32  
There are 26 letters and 10 numerals. 26 x 26 + 26 x 10 = 936 possible addresses.

11-33  
The processor transmits the address of the terminal followed by ENQ (enquiry) code 0000 0101. The terminal responds with either ACK (acknowledge) or NAK (negative acknowledge) or the terminal does not respond during a timeout period. If the processor receives an ACK, it sends a block of text.

11-34  
<table>
<thead>
<tr>
<th>DLE</th>
<th>STX</th>
<th>DLE</th>
<th>DLE</th>
<th>ETX</th>
<th>DLE</th>
<th>DLE</th>
<th>ETX</th>
<th>DLE</th>
<th>ETX</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

↑     ↑     ↑     ↑     ↑
| delete | delete | delete | delete | delete |

32-bit text = 0001 0000 1000 0011 0001 0000 1000 0011

11-35  
32 bits between two flags; 48 bits including the flags.

11-36  
Information to be sent (1023): 011111111111
After zero insertion, information transmitted: 011111011110 delete
Information received after 0's deletion: 0111111111
12-1 (2) \( \frac{2048}{128} = 16 \) chips

(b) \( 2048 = 2^{11} \) 11 lines to address 2048 bytes
\( 128 = 2^{7} \) 7 lines to address each chip

4 lines to decoder for selecting 16 chips

(c) 4x16 decoder

12-2

(2) 8 chips are needed with address lines connected in parallel.

(b) \( 16 \times 8 = 128 \) chips. Use 14 address lines \( (16^k = 2^{14}) \)
10 lines specify the chip address
4 lines are decoded into 16 chip-select inputs.

12-3

10 pins for inputs, 4 for chip-select, 8 for outputs, 2 for power.
Total of 24 pins.

12-4

\[ \frac{4096}{128} = 32 \text{ RAM chips} ; \quad \frac{4096}{512} = 8 \text{ ROM chips}. \]

\[ 4096 = 2^{12} \] There are 12 common address lines + 1 line to select between RAM and ROM.

<table>
<thead>
<tr>
<th>Component</th>
<th>Address</th>
<th>16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM</td>
<td>0000-0FFF</td>
<td>0 0 0 0 [ \xrightarrow{5 \times 32 \text{ decoder}} ] x x x x x x x x</td>
</tr>
<tr>
<td>ROM</td>
<td>1000-1FFF</td>
<td>0 0 0 1 [ \xrightarrow{3 \times 8 \text{ decoder}} ] x x x x x x x x</td>
</tr>
</tbody>
</table>

12-5

RAM \( 2048 \div 256 = 8 \) chips ; \( 2048 = 2^{11} \); \( 256 = 2^{8} \)

ROM \( 4096 \div 1024 = 4 \) chips; \( 4096 = 2^{12} \); \( 1024 = 2^{10} \)

Interface \( 4 \times 4 = 16 \) registers; \( 16 = 2^{4} \)

<table>
<thead>
<tr>
<th>Component</th>
<th>Address</th>
<th>16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM</td>
<td>0000-07FF</td>
<td>0 0 0 0 [ \xrightarrow{3 \times 8 \text{ decoder}} ] x x x x x x x x</td>
</tr>
<tr>
<td>ROM</td>
<td>4000-4FFF</td>
<td>0 1 0 0 [ \xrightarrow{3 \times 8 \text{ decoder}} ] x x x x x x x x</td>
</tr>
<tr>
<td>Interface</td>
<td>8000-800F</td>
<td>1 0 0 0 0 0 0 0 0 0 0 0 0 x x x</td>
</tr>
</tbody>
</table>

12-6

The processor selects the external register with an address 8000 hexadecimal. Each bank of 32K bytes are selected by addresses 0000-7FFF. The processor loads an 8-bit number into the register with a single 1 and 7 0's. Each output of the register selects one of the 8 banks of 32K bytes through a chip-select input. A memory bank can be changed by changing the number in the register.
12-7 Average time \( T_a = T_s + \frac{1}{2R} + \frac{N_s}{N_t} \cdot \frac{1}{R} \) to read a sector.

12-8 An eight-track tape reads 8 bits (one character) at the same time. Transfer rate \( 1600 \times 120 = 192,000 \) characters/s.

12-9 From Sec. 12-4:

\[
M_i = \prod_{j=1}^{n} (A_j \oplus F_{ij}) K_j
\]

\[
M_i' = \sum_{j=1}^{n} (A_j \oplus F_{ij}) K_j
\]

From other bits in the same word

12-10 A match occurs if \( T_i = 1 \)

\[
match = M_i T_i
\]

12-11 \( M_i = \left( \prod_{j=1}^{n} A_j F_{ij} + A_j' F_{ij} + K_j' \right) \cdot (K_1 + K_2 + K_3 + \ldots + K_n) \)

At least one key bit \( K_i \) must be equal to 1.

12-12(c)
A $d$-bit counter drives a $d$-to-$m$ line decoder where $2^d = m$ ($m$ = No. of words in memory). For each count, the $M_i$ bit is checked and if 1, the corresponding read signal for word $i$ is activated.

Let $X_j = A_j F_{i,j} + A'_j F'_{i,j}$ (argument bit = memory word bit)

Output indicator $G_i = 1$ if:

$A_1 F_{i1} = 1$ and $K_1 = 1$ (First bit in $A = 1$ while $F_{i1} = 0$)

or if $X_1 A_2 F_{i2} = 1$ and $K_2 = 1$ (First pair of bits are equal and second bit in $A = 1$ while $F_{i2} = 0$)

$G_i = (A_1 F'_{i1} + K'_1)(X_1 A_2 F_{i2} + K'_2)(X_2 A_3 F_{i3} + K'_3)...(X_{n-1} A_n F_{in} + K'_n)$
12-15

\[ 128K = 2^{17} \]; For a set size of 2, the index address has 10 bits to accommodate \( \frac{2048}{2} = 1024 \) words of cache.

(a) \[
\begin{array}{|c|c|}
\hline
\text{TAG} & \text{INDEX} \\
\hline
\end{array}
\]

\[ \text{Block} \rightarrow \text{Word} \quad \text{8 bits} \rightarrow \text{2 bits} \]

(b) \[
\begin{array}{c}
\text{Tag 1} \quad \text{Data 1} \\
\text{Tag 2} \quad \text{Data 2}
\end{array}
\]

Size of cache memory is \( 1024 \times 2 \times (7+32) \)
\[ = 1024 \times 78 \]

12-16

(a) \[ 0.9 \times 100 + 0.1 \times 11000 = 90 + 110 = 200 \text{ nsec.} \]

\[ \begin{array}{c}
\text{cache access} \\
\text{cache + memory access}
\end{array} \]

(b) \[ 0.2 \times 1000 + 0.8 \times 200 = 200 + 160 = 360 \text{ nsec.} \]

\[ \begin{array}{c}
\text{write access} \\
\text{read access from (a)}
\end{array} \]

(c) Hit ratio = \( 0.8 \times 0.9 = 0.72 \)

12-17

Sequence: \( ABCDEAECAE \)

LRU

Count value = \( 3 \quad 2 \quad 1 \quad 0 \)

Initial Words = \( A \quad B \quad C \quad D \)

\( B \) is a hit \( \rightarrow A \quad C \quad D \quad B \)

\( E \) is a miss \( \rightarrow C \quad D \quad B \quad E \)

\( D \) is a hit \( \rightarrow C \quad B \quad E \quad D \)

\( A \) is a miss \( \rightarrow B \quad E \quad D \quad A \)

\( C \) is a miss \( \rightarrow E \quad D \quad A \quad C \)

\( E \) is a hit \( \rightarrow D \quad A \quad C \quad E \)

\( C \) is a hit \( \rightarrow D \quad A \quad E \quad C \)

\( E \) is a hit \( \rightarrow D \quad A \quad C \quad E \)
12-18

64K x 16: 16 bit address; 16 bit data.

(a) 6 8 2 = 16 bit address

\[
\begin{array}{ccc}
\text{TAG} & \text{BLOCK} & \text{WRD} \\
\end{array}
\]

INDEX = 10 bit cache address.

(b) 1 6 16 = 23

\[
\begin{array}{ccc}
\text{V} & \text{TAG} & \text{DATA} \\
\end{array}
\]

in each 16 bit of each cache

(c) \(2^{6} = 256\) blocks of 4 words each

12-19

(a) memory space = 24 bits \(2^{24} = 16\) M words

(b) memory space = 16 bits \(2^{16} = 64\) K words

(c) \(\frac{16\text{M}}{\text{2K}} = 8\) K pages \(\frac{64\text{K}}{\text{2K}} = 32\) blocks

12-20

The pages that are not in main memory are:

<table>
<thead>
<tr>
<th>Page</th>
<th>Address</th>
<th>Address that will cause fault</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2K</td>
<td>2048 – 3071</td>
</tr>
<tr>
<td>3</td>
<td>3K</td>
<td>3072 – 4095</td>
</tr>
<tr>
<td>5</td>
<td>5K</td>
<td>5120 – 6143</td>
</tr>
<tr>
<td>7</td>
<td>7K</td>
<td>7168 – 8191</td>
</tr>
</tbody>
</table>

12-21

<table>
<thead>
<tr>
<th>Page reference</th>
<th>(a) Pages in main memory</th>
<th>Contents of FIFO</th>
<th>(b) Pages in memory</th>
<th>LRU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial</td>
<td>0124 4201</td>
<td></td>
<td>0124 4201</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0124 4201</td>
<td></td>
<td>0124 4012</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0126 2016</td>
<td></td>
<td>0126 0126</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0146 0164</td>
<td></td>
<td>0146 2614</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0146 0164</td>
<td></td>
<td>0146 6140</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0146 0164</td>
<td></td>
<td>0146 6401</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1246 1642</td>
<td></td>
<td>0123 4102</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2346 6423</td>
<td></td>
<td>0123 1023</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>2345 4235</td>
<td></td>
<td>0235 0235</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>2357 2357</td>
<td></td>
<td>2357 2357</td>
<td></td>
</tr>
</tbody>
</table>

Most recently used
600AF and 500AF

12-23

Logical address: \[\begin{array}{ccc} 7 \text{ bits} & 5 \text{ bits} & 12 \text{ bits} \\ \text{Segment} & \text{Page} & \text{Word} \end{array}\]

Physical address: \[\begin{array}{cc} 12 \text{ bits} & 12 \text{ bits} \\ \text{Block} & \text{Word} \end{array}\]

12-24

Segment 36 = (0100100)_2 (7-bit binary)
Page 15 = (01111)_2 (5-bit binary)
Word 2000 = (01111010000)_2 (12-bit binary)

Logical address = C1 60 100 01111 01111010000

(24-bit binary)
Tightly coupled multiprocessors require that all processors in the system have access to a common global memory. In loosely coupled multiprocessors, the memory is distributed and a mechanism is required to provide message-passing between the processors. Tightly coupled systems are easier to program since no special steps are required to make shared data available to two or more processors. A loosely coupled system required that sharing of data be implemented by the messages.

The address assigned to common memory is never assigned to any of the local memories. The common memory is recognized by its distinct address.

\[ P \times m \text{ switches} \]

\[ \log _2 n \text{ stages with } \frac{n}{2} \text{ switches in each stage.} \]

Inputs 0, 2, 4, and 6 will be disconnected from outputs 2 and 3.
Arbitration switch:

A connected to output
B connected to output

Distribution switch:

Input connected to A
Input connected to B

(b)

I = interchay switch

(c)
Paths from 7 to 9:

- 7 - 15 - 13 - 9
- 7 - 15 - 11 - 9
- 7 - 3 - 11 - 9
- 7 - 3 - 1 - 9
- 7 - 5 - 13 - 9
- 7 - 5 - 1 - 9

13-9

PI

Request for bus transfer
Transfer completed

If PI = 0*:

If PI goes to 0 while using the bus, the device must terminate its bus operation and then reset both flip-flops.
13-10
Encoder input  0 1 1 0
Encoder output  0 1 0 1 0 (It has highest priority)
Decoder input  0 1 0 0
Decoder output  0 1 1 0 0 Arbiter 2 (3) is acknowledged

13-11
As explained in the text, connect output P0 from arbiter 4 into input P3 of arbiter 6. Once the
line is disabled, the arbiter that released the
bus has the lowest priority.

13-12
Memory access needed to send data from one
processor to another must be synchronized
with test-and-set instructions. Most of the
time would be taken up by unsuccessful
test by the receiver. One way to speed
the transfer would be to send an interrupt
request to the receiving processor.

13-13
(a) Mutual exclusion implies that each processor
claims exclusive control of the resources
allocated to it.

(b) Critical section is a program sequence that
must be completely executed without interruptions
by other processors.

(Continued in next page)
(c) **Hardware lock** is a hardware signal to ensure that a memory read is followed by a memory write without interruption from another processor.

(d) **Semaphore** is a variable that indicates the number of processes attempting to use the critical section.

(e) **Test and set instruction** causes a read-modify-write memory operation so that the memory location cannot be accessed and modified by another processor.

Cache coherency is defined as the situation in which all cache copies of shared variables in a multiprocessor system have the same value at all times. A snoopy cache controller is a monitoring action that detects a write operation into any cache. The cache coherence problem can be resolved by either updating or invalidating all other cache values of the written information.