CHAPTER 1

Table 1-1:

<table>
<thead>
<tr>
<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' \cdot 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>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<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>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>0</td>
<td>0</td>
<td>1</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>0</td>
<td>1</td>
<td>0</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>0</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>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 1-2:

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

1-3
(a) \( A + A \cdot B = A (1 + B) = A \)
(b) \( A B + AB' = A (B + B') = A \)
(c) \( A' \cdot B \cdot C + AC = C (A' \cdot B + A) = C (A' + A) (B + A) = (A + B) C \)
(d) \( A' \cdot B + A B' + A B C = A' \cdot B + A B (C' + C) = A' \cdot B + A B = B (A' + A) = B \)

1-4
(a) \( A B + A (C \cdot D + C \cdot D') = A B + A C (D + D') = A (B + C) \)
(b) \( (B C' + A' D) (A B' + C D') = 
\begin{align*}
&= \frac{A B B C' + A' A B' D + B C C' D' + A' C D' D}{0}
\end{align*}
= 0

www.samy007.blogfa.com
1-5
(a) \((A+B')(A'+B')' = (A'B')(A'B) = 0\)
(b) \(A + A'B + A'B' = A + A'(B + B') = A + A' = 1\)

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

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

(d) Same as (e)
\[
\begin{align*}
F &= x'y' + x'z \\
F &= y + x'z \\
F &= xy + x'z + y'z \\
F &= c' + a'b
\end{align*}
\]

\[
\begin{align*}
F &= BCD + A'B'D' \\
F &= cD + ABC + ABD
\end{align*}
\]

\[
\begin{align*}
F &= A'C' + A'B'D' + ABD + \left( A'B'D' \right) \\
F &= BD + B'D' + \left( A'B' \right) \\
F &= AD'
\end{align*}
\]
(a) \( F = x'y + z' \)
\( F' = x'z + y'z \)
(2) \( F = (x+z')(y+z') \)

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

\[ F = B'D' + A3' + AC \]

(truth tables and logic diagrams are shown)

\( F' = AC' + B'C' + AB'D' \)
\( F = (A'+C)(B+C)(A'+B+D) \)
(a) \( F = x'z' + w'z \)
(b) \( = (x' + z')(w' + z') \)

\[ S = x'y'z + x'y'z' + xy'z' + xyz \]
\[ = x'(y'z + y'z') + x(y'z' + yz) \rightarrow \text{See Fig 1-2} \]
\[ = x'(y \oplus z) + x(y \oplus z)' \left( \text{Exclusive-NOR} \right) \]
\[ = x \oplus y \oplus z \]

\[ F = xy + xz + yz \]

\[ A = xy + xz + yz \]
\[ F = x \oplus y \oplus z \]

\[ C = z' \]
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

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

(a)

(b)

<table>
<thead>
<tr>
<th>Present state</th>
<th>Inputs</th>
<th>Next state</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>( AB )</td>
<td>( xyd )</td>
<td>( AB )</td>
<td>( z )</td>
</tr>
<tr>
<td>00</td>
<td>00</td>
<td>00</td>
<td>0</td>
</tr>
<tr>
<td>00</td>
<td>01</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>00</td>
<td>10</td>
<td>00</td>
<td>0</td>
</tr>
<tr>
<td>00</td>
<td>11</td>
<td>00</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>00</td>
<td>01</td>
<td>1</td>
</tr>
<tr>
<td>01</td>
<td>01</td>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>01</td>
<td>10</td>
<td>00</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>11</td>
<td>00</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>00</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>01</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>00</td>
<td>01</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>01</td>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>10</td>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>11</td>
<td>1</td>
</tr>
</tbody>
</table>
1-20

\[ J_A = K_A = x \]
\[ J_B = K_B = x' \]

1-21

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

<table>
<thead>
<tr>
<th>Present State</th>
<th>Inputs</th>
<th>Next State ( x )</th>
<th>Flip-flop inputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>E</td>
<td>( x )</td>
</tr>
<tr>
<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>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</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>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

\( J_A = (B \cdot x + B' \cdot x') \cdot E \)

\( K_A = (B \cdot x + B' \cdot x') \cdot E \)

\( K_B = E \)
2-1
(a) Inverters - 2 pins each  \( \frac{12}{2} = 6 \text{ gates} \) 7404
(b) 2-input XOR - 3 pins each  \( \frac{12}{3} = 4 \text{ gates} \) 7446
(c) 3-input OR - 4 pins each  \( \frac{12}{4} = 3 \text{ gates} \)
(d) 4-input AND - 5 pins each  \( \frac{12}{5} = 2 \text{ gates} \) 7421
(e) 5-input NOR - 6 pins each  \( \frac{12}{6} = 2 \text{ gates} \) 74260
(f) 8-input NAND - 9 pins
(g) JK flip-flop - 6 pins each  \( \frac{12}{6} = 2 \text{ FFs} \) 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

[Diagram]

www.samy007.blogfa.com
2-4

\[ D_0 = (A_1 + A_0 + E') = A_1'A_0'E \]
\[ D_1 = (A_1'A_0 + E') = A_1'A_0'E \]
\[ D_2 = (A_1 + A_0 + E') = A_1'A_0'E \]
\[ D_3 = (A_1 + A_0 + E') = A_1A_0E \]

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 \( A_2A_1A_0 = 000 \)

Needs one more output

to recognize the all zeros input condition.

2-7
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.

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

One stage of the Register Fig. 2-7

- **Load**
  - 0: Load
  - 1: Clear

- **Clear**
  - 0: Load
  - 1: Clear

**Function Table**

<table>
<thead>
<tr>
<th>Load</th>
<th>Clear</th>
<th>D</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Get</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Clear to 0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>X</td>
<td>Load 0</td>
</tr>
</tbody>
</table>

Load Clear D Operation

www.samy007.blogfa.com
2-12
\[
\begin{array}{c|c}
\text{Input bits} & 4\text{-bit register} \\
\hline
1 & 1101 \leftarrow \text{Initial value} \\
0 & 1110 \\
1 & 0111 \\
1 & 1011 \\
0 & 1101 \\
1 & 0110 \\
\end{array}
\]

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
\[
1000 \rightarrow 0100 \rightarrow 0010 \rightarrow 0001
\]

2-15
![Diagram of a digital circuit with labels for S1, S0, Ser in-right, Ser in-left, clock, power, clear, Fig. 2-9, and Fig. 2-9 with A0, A1, A2, A3, inputs 1-12, outputs 1-12.]

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

2-17
![Diagram of a digital circuit with labels for count, load, clear, clock, Fig. 2-11 stage 1, output carry, load, clear, clock, Fig. 2-11 stage 2, output carry, to next stage.]

www.samy007.blogfa.com
2-18
After the count reaches \( N-1 = 1001 \), the register loads 0000 from inputs.

2-19

\[
\begin{align*}
(a) & 2K \times 16 = 2^{11} \times 16 & \text{Address lines} & 11 & \text{Data lines} & 16 \\
(b) & 64K \times 8 = 2^{16} \times 16 & 16 & 8 \\
(c) & 16M \times 32 = 2^{24} \times 32 & 24 & 32 \\
(d) & 4G \times 64 = 2^{32} \times 64 & 32 & 64
\end{align*}
\]

2-20

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

2-21

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

2-22

5 inputs common to all chips

2-23

12 data inputs + 2 enable inputs + 8 data outputs + 2 for power = 24 pins
3-1
\[(101110)_2 = 32 + 8 + 4 + 2 = 46 \]
\[(1110101)_2 = 64 + 32 + 16 + 4 + 1 = 117 \]
\[(110110100)_2 = 256 + 128 + 32 + 16 + 4 = 436 \]

3-2
\[(12121)_3 = 3^4 + 2 \times 3^3 + 2^2 + 2 \times 3 + 1 = 81 + 54 + 9 + 6 + 1 = 151 \]
\[(4310)_5 = 4 \times 5^3 + 3 \times 5^2 + 4 = 500 + 75 + 5 = 580 \]
\[(50)_7 = 5 \times 7 = 35 \]
\[(110012)_12 = 12^2 + 2 \times 12 + 1 = 144 + 24 + 1 = 269 \]

3-3
\[(1231)_{10} = 10^4 + 128 + 64 + 15 = 2^9 + 2^7 + 2^6 + 2^2 + 2 + 1 = (1001100111)_2 \]
\[(673)_{10} = 512 + 128 + 32 + 1 = 2^9 + 2^7 + 2^5 + 2 + 1 = (1010100001)_2 \]
\[(1998)_{10} = 1024 + 512 + 256 + 128 + 64 + 8 + 4 + 2 = 2^10 + 2^9 + 2^8 + 2^7 + 2^6 + 2^5 + 2^3 + 2^2 + 2 = (1111001110)_2 \]

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

3-5
\[(F3A7C2)_{16} = (11110011101001111000010)_2 \]
\[= (74723702)_8 \]

3-6
\[(x^2 - 10x + 31)_Y = [(x-5)(x-8)]_{10} \]
\[= x^2 - (5+8)_{10}x + (40)_{10} \]
Therefore: \[(10)_Y = (13)_{10} \quad Y = 13 \]
Also \[(31)_Y = 3 \times 13 + 1 = (40)_{10} \]
\[(Y = 13) \]
3-7 \((215)_{10} = 128 + 64 + 16 + 7 = (11010111)_2\)

(a) \(000011010111\) \ Binary
(b) \(000011110111\) \ Binary coded octal
(c) \(000011011011\) \ Binary coded hexadecimal
(d) \(001100010101\) \ Binary coded decimal

3-8 \((295)_{10} = 256 + 32 + 7 = (100100111)_2\)

(a) \(00000000000000100100111\)
(b) \(0000000000000010100101\)
(c) \(10110010\ 00111001\ 00110101\)

3-10 JOHN DOE

3-11 \(87650123; 9019999; 09990048; 9999999\)

3-12 \(876100; 909343; 900000; 000000\)

3-13 \(01010001; 01111110; 01111111; 11111110; 11111111\)
\(01010010; 01111111; 10000000; 11111111; 00000000\)

3-14

(a) \(5250\)
\(+8679\)
\(\downarrow 3929\)

(b) \(1753\)
\(+1360\)
\(\downarrow 3113\)

(c) \(020\)
\(+900\)
\(\uparrow 920\)

(d) \(1200\)
\(+9750\)
\(\downarrow 0950\)

\(-6887\)
\(-080\)

3-15

(a) \(11010\)
\(+10000\)
\(\downarrow 010110\)

(b) \(11010\)
\(+10011\)
\(\downarrow 010110\)

(c) \(000100\)
\(+010000\)
\(\downarrow 010110\)

(d) \(1010100\)
\(+0101100\)
\(\downarrow 10000000\)

\((26-16=10)\)
\((26-13=13)\)
\((-101100)\)
\((4-48=-44)\)

\((84-84=0)\)
3-16
\[
\begin{align*}
+42 &= 0101010 \\
-42 &= 1010110 \\
(42) &= 0101010 \\
(-42) &= 1010110 \\
(29) &= 0011101 \\
(-29) &= 1100011
\end{align*}
\]

3-17
\[
\begin{align*}
01 &\leftarrow \text{last two carries} \rightarrow 0 \\
+70 &= 01000110 \\
+80 &= 01010000 \\
+150 &= 10010110 \\
\text{greater} &\quad\text{negative}
\end{align*}
\]

3-18
\[
\begin{align*}
(2) (-638) &= 9362 \\
\text{(a):} +785 &= 0.1101 \\
\text{(b):} (-185) &= 9815 \\
\text{(a):} +9147 &= 0.147 \\
\text{(b):} -823 &= 9177
\end{align*}
\]

3-19
\[
\begin{align*}
\text{Mantissa:} & \quad +0.1111\ldots,1 \\
\text{Exponent:} & \quad +11111111 \\
\text{Largest:} & \quad 1 - 2^{-26} \\
\text{Smallest:} & \quad -1111111 \\
\text{Normalized:} & \quad 2^{-1}
\end{align*}
\]

3-20
\[46.5 = 32 + 8 + 4 + 2 + 0.5 = (101110.1)_2\]

Sign: 0
\[
\begin{align*}
\text{24-bit mantissa} &= 1011010000000000 \\
\text{8-bit exponent (+6)} &= 00000110
\end{align*}
\]
### 3-21 (a)

<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>11111</td>
</tr>
<tr>
<td>21</td>
<td>11110</td>
</tr>
<tr>
<td>22</td>
<td>11110</td>
</tr>
<tr>
<td>23</td>
<td>11101</td>
</tr>
<tr>
<td>24</td>
<td>11000</td>
</tr>
<tr>
<td>25</td>
<td>10100</td>
</tr>
<tr>
<td>26</td>
<td>10111</td>
</tr>
<tr>
<td>27</td>
<td>10110</td>
</tr>
<tr>
<td>28</td>
<td>10010</td>
</tr>
<tr>
<td>29</td>
<td>10011</td>
</tr>
<tr>
<td>30</td>
<td>10001</td>
</tr>
<tr>
<td>31</td>
<td>10000</td>
</tr>
</tbody>
</table>

### 3-21 (b)

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Excess-3 Gray</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>0010 1010</td>
</tr>
<tr>
<td>10</td>
<td>0110 1010</td>
</tr>
<tr>
<td>11</td>
<td>0110 1111</td>
</tr>
<tr>
<td>12</td>
<td>0110 1111</td>
</tr>
<tr>
<td>13</td>
<td>0110 1101</td>
</tr>
<tr>
<td>14</td>
<td>0110 1100</td>
</tr>
<tr>
<td>15</td>
<td>0110 0100</td>
</tr>
<tr>
<td>16</td>
<td>0110 0101</td>
</tr>
<tr>
<td>17</td>
<td>0110 0111</td>
</tr>
<tr>
<td>18</td>
<td>0110 0111</td>
</tr>
<tr>
<td>19</td>
<td>0110 0010</td>
</tr>
<tr>
<td>20</td>
<td>0110 0010</td>
</tr>
</tbody>
</table>

### 3-22 8620

(a) BCD 1000 0110 0010 0000  
(b) XS-3 1011 1001 0101 0011  
(c) 2421 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>00000</td>
<td>10000</td>
</tr>
<tr>
<td>1</td>
<td>10001</td>
<td>00000.1</td>
</tr>
<tr>
<td>2</td>
<td>10010</td>
<td>00010</td>
</tr>
<tr>
<td>3</td>
<td>00011</td>
<td>10011</td>
</tr>
<tr>
<td>4</td>
<td>10100</td>
<td>00100</td>
</tr>
<tr>
<td>5</td>
<td>00101</td>
<td>10101</td>
</tr>
<tr>
<td>6</td>
<td>00110</td>
<td>10110</td>
</tr>
<tr>
<td>7</td>
<td>10111</td>
<td>00111</td>
</tr>
<tr>
<td>8</td>
<td>11000</td>
<td>01000</td>
</tr>
<tr>
<td>9</td>
<td>01001</td>
<td>11001</td>
</tr>
</tbody>
</table>
3-24
3984 = 0011 1111 1110 0100
1100 0000 0001 1011 = 6015

3-25

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>y = A ⊕ 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 ⊕ 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>

<table>
<thead>
<tr>
<th>y</th>
<th>z</th>
<th>x = y ⊕ z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
| 0 | 1 |   1 \( \rightarrow \) \begin{align*}
  & A \neq B = 00 \text{ or } 11 \\
  & C \neq D = 01 \text{ or } 10
\end{align*} |
| 1 | 0 |   1 \( \rightarrow \) \begin{align*}
  & A \neq B = 01 \text{ or } 10 \\
  & C \neq D = 00 \text{ or } 11
\end{align*} |
| 1 | 1 |   0     |

**Abbreviations**
- A, B, C, D

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: R1 ← R2
P'Q: R1 ← 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 S1S0 = 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
4-9

4-10

4-11

4-12

<table>
<thead>
<tr>
<th>M</th>
<th>A</th>
<th>B</th>
<th>Sum</th>
<th>C4</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 $A - 1 = A + 2's$ complement of $1 = A + 1111$

4-14

4-15

\[
\begin{array}{c|cccc}
S & C_{in} & X & Y \\
\hline
0 & 0 & A & B & (A+B) \\
0 & 1 & A & 0 & (A+1) \\
1 & 0 & A & 1 & (A-1) \\
1 & 1 & A & B & (A-B) \\
\end{array}
\]
4-16

Binary value of \( F_i \):

\[
\begin{array}{c|c|c|c|c|c|c}
0 & 1 & 2 & 3 \\
\hline
S_1 & S_0 & 4 \times 1 \\
\hline
0 & 1 & 2 & 3 \\
\hline
\end{array}
\]

\[ i = 0, 1, 2, \ldots, 15 \]

4-17

4-18

(a) \[ A = \overline{11011001} \oplus \overline{10110100} \]
\[ A = \overline{11011001} \oplus \overline{01101101} \]

(b) \[ A = \overline{11100010} \]

(c) \[ A = \overline{11110001} \]

4-19

4-18

(a) \[ A = \overline{11100010} \]

(b) \[ A = \overline{11110001} \]

(c) \[ A = \overline{10101000} \]

www.samy007.blogfa.com
4-20

\[ R = 10011100 \]

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

4-21

\[ R = 11011101 \]

logical shift left : 10111010
circular shift right : 01011101
logical shift right : 00101110
circular shift left : 01011100

4-22

\[ S=1 \text{ Shift left} \]

\[ \begin{array}{c}
A_0 \ A_1 \ A_2 \ A_3 \ \text{IL} \\
1 \ 0 \ 0 \ 1 \ \text{shift left}
\end{array} \]

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

\[ \frac{25}{2} = 32 - 25 = 7 \text{ bits for opcode.} \]

(b) \[ \begin{array}{c|c|c|c}
I & \text{Opcode} & \text{Register} & \text{Address} \\
\hline
1 & 7 & 6 & 18 \\
\end{array} \]

= 32 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

<table>
<thead>
<tr>
<th>(1) ( S_2S_1S_0 )</th>
<th>(2) Load (4D)</th>
<th>(3) Memory</th>
<th>(4) Adder</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a) AR \leftarrow PC</td>
<td>010 (PC)</td>
<td>AR</td>
<td>-</td>
</tr>
<tr>
<td>(b) IR \leftarrow M[AR]</td>
<td>111 (M)</td>
<td>IR</td>
<td>Read</td>
</tr>
<tr>
<td>(c) M[AR] \leftarrow TR</td>
<td>110 (TR)</td>
<td>-</td>
<td>Write</td>
</tr>
<tr>
<td>(d) DR \leftarrow AC</td>
<td>100 (AC)</td>
<td>DR and AC</td>
<td>-</td>
</tr>
</tbody>
</table>
(a) \( IR \leftarrow M[PC] \) PC cannot provide address to memory, address must be transferred to AR first
\[ AR \leftarrow PC \]
\[ IR \leftarrow M[AR] \]

(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 5-46b)
\[ AC \leftarrow AC + DR \]
\[ AC \leftarrow DR, DR \leftarrow AC \]

5-6

(2) \[
\begin{array}{c}
0001 0000 0110 0100 \\
ADD \quad (024)_{16}
\end{array}
\]
\[
\begin{array}{c}
0001 0000 0110 0100 \\
ADD \quad \text{content of } M[024] \text{ to } AC \quad \text{ADD 024}
\end{array}
\]

(b) \[
\begin{array}{c}
1011 0001 0010 0100 \\
I \text{ STA} \quad (124)_{6}
\end{array}
\]
\[
\begin{array}{c}
\text{Store } AC \text{ in } M[M[124]] \quad \text{STA I 124}
\end{array}
\]

(c) \[
\begin{array}{c}
0111 0000 0010 0000 \\
\text{Register} \quad \text{Increment } AC
\end{array}
\]
\[
\begin{array}{c}
0111 0000 0010 0000 \\
\text{INC}
\end{array}
\]

5-7

CLE Clear E
CME Complement E
5-8

\[ \text{clock} \quad T_0 \quad T_1 \quad T_2 \quad T_3 \quad T_0 \]

\[ T_0 \]

\[ T_1 \]

\[ T_2 \]

\[ T_3 \]

\[ C_7 \]

\[ C_7T_3 \]

\[ \uparrow \text{sc goes to 0} \]

\[ \text{causing } T_0 = 1 \]

5-9

<table>
<thead>
<tr>
<th></th>
<th>E</th>
<th>AC</th>
<th>PC</th>
<th>AR</th>
<th>IR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial</td>
<td>1</td>
<td>A937</td>
<td>022</td>
<td>100</td>
<td>7100</td>
</tr>
<tr>
<td>CLA</td>
<td>1</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>
<td>7400</td>
</tr>
<tr>
<td>CMA</td>
<td>1</td>
<td>5688</td>
<td>022</td>
<td>800</td>
<td>7200</td>
</tr>
<tr>
<td>CME</td>
<td>0</td>
<td>A937</td>
<td>022</td>
<td>100</td>
<td>7100</td>
</tr>
<tr>
<td>CIR</td>
<td>1</td>
<td>D498</td>
<td>022</td>
<td>080</td>
<td>7080</td>
</tr>
<tr>
<td>CIL</td>
<td>1</td>
<td>526F</td>
<td>022</td>
<td>040</td>
<td>7040</td>
</tr>
<tr>
<td>INC</td>
<td>1</td>
<td>A938</td>
<td>022</td>
<td>020</td>
<td>7020</td>
</tr>
<tr>
<td>SDR</td>
<td>1</td>
<td>A937</td>
<td>022</td>
<td>010</td>
<td>7010</td>
</tr>
<tr>
<td>SNA</td>
<td>1</td>
<td>A937</td>
<td>023</td>
<td>008</td>
<td>7008</td>
</tr>
<tr>
<td>SZA</td>
<td>1</td>
<td>A937</td>
<td>022</td>
<td>004</td>
<td>7004</td>
</tr>
<tr>
<td>SZE</td>
<td>1</td>
<td>A937</td>
<td>022</td>
<td>002</td>
<td>7002</td>
</tr>
<tr>
<td>HLT</td>
<td>1</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>
<th>SC</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Initial</strong></td>
<td>021</td>
<td>-</td>
<td>-</td>
<td>A937</td>
<td>-</td>
</tr>
<tr>
<td><strong>AND</strong></td>
<td>022</td>
<td>083</td>
<td>B8F2</td>
<td>A332</td>
<td>0083</td>
</tr>
<tr>
<td><strong>ADD</strong></td>
<td>022</td>
<td>083</td>
<td>B8F2</td>
<td>6229</td>
<td>1083</td>
</tr>
<tr>
<td><strong>LDA</strong></td>
<td>022</td>
<td>083</td>
<td>B8F2</td>
<td>B8F2</td>
<td>2083</td>
</tr>
<tr>
<td><strong>STA</strong></td>
<td>022</td>
<td>083</td>
<td>-</td>
<td>A937</td>
<td>3083</td>
</tr>
<tr>
<td><strong>BUN</strong></td>
<td>083</td>
<td>083</td>
<td>-</td>
<td>A937</td>
<td>4083</td>
</tr>
<tr>
<td><strong>BSA</strong></td>
<td>084</td>
<td>084</td>
<td>-</td>
<td>A937</td>
<td>5083</td>
</tr>
<tr>
<td><strong>ISZ</strong></td>
<td>022</td>
<td>083</td>
<td>B8F3</td>
<td>A937</td>
<td>6083</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><strong>Initial</strong></td>
<td>7FF</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td><strong>T0</strong></td>
<td>7FF</td>
<td>7FF</td>
<td>-</td>
<td>EA9F</td>
</tr>
<tr>
<td><strong>T1</strong></td>
<td>800</td>
<td>7FF</td>
<td>-</td>
<td>EA9F</td>
</tr>
<tr>
<td><strong>T2</strong></td>
<td>800</td>
<td>A9F</td>
<td>-</td>
<td>EA9F</td>
</tr>
<tr>
<td><strong>T3</strong></td>
<td>800</td>
<td>C35</td>
<td>-</td>
<td>EA9F</td>
</tr>
<tr>
<td><strong>T4</strong></td>
<td>800</td>
<td>C35</td>
<td>FFFF</td>
<td>EA9F</td>
</tr>
<tr>
<td><strong>T5</strong></td>
<td>800</td>
<td>C35</td>
<td>0000</td>
<td>EA9F</td>
</tr>
</tbody>
</table>

5-12

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

\[
i = 1 \quad \text{ADD} \quad \text{ADD} \quad 32E
\]

(b) \( AC = 7EC3 \) (ADD)

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

(c) \( PC = 3AF + 1 = 3BO \)

\[
\text{IR} = 932E \quad E = 1 \quad I = 1 \quad SC = 0000
\]

Memory

\[
\begin{array}{c|c}
3AF & 932E \\
32E & 09AC \\
9AC & 8B9F \\
\end{array}
\]

\( AC = 7EC3 \)
<table>
<thead>
<tr>
<th>Instruction</th>
<th>Action 1</th>
<th>Action 2</th>
<th>Action 3</th>
<th>Action 4</th>
<th>Action 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>S-13</td>
<td>D_{0} T_{4} : \text{DR} \leftarrow M[\text{AR}]</td>
<td>D_{5} T_{5} : \text{AC} \leftarrow \text{AC} \oplus \text{DR}, \text{SC} \leftarrow 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADM</td>
<td>D_{1} T_{4} : \text{DR} \leftarrow M[\text{AR}]</td>
<td>D_{1} T_{5} : \text{DR} \leftarrow \text{AC}, \text{AC} \leftarrow \text{AC} + \text{DR}</td>
<td>D_{1} T_{6} : M[\text{AR}] \leftarrow \text{AC}, \text{AC} \leftarrow \text{DR}, \text{SC} \leftarrow 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SUB</td>
<td>D_{2} T_{4} : \text{DR} \leftarrow M[\text{AR}]</td>
<td>D_{2} T_{5} : \text{DR} \leftarrow \text{AC}, \text{AC} \leftarrow \text{DR}</td>
<td>D_{2} T_{6} : \text{AC} \leftarrow \overline{\text{AC}}</td>
<td>D_{2} T_{7} : \text{AC} \leftarrow \text{AC} + 1</td>
<td>D_{2} T_{8} : \text{AC} \leftarrow \text{AC} + \text{DR}, \text{SC} \leftarrow 1</td>
</tr>
<tr>
<td>XCH</td>
<td>D_{3} T_{4} : \text{DR} \leftarrow M[\text{AR}]</td>
<td>D_{3} T_{5} : M[\text{AR}] \leftarrow \text{AC}, \text{AC} \leftarrow \text{DR}, \text{SC} \leftarrow 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SEQ</td>
<td>D_{4} T_{4} : \text{DR} \leftarrow M[\text{AR}]</td>
<td>D_{4} T_{5} : \text{TR} \leftarrow \text{AC}, \text{AC} \leftarrow \text{AC} \oplus \text{DR}</td>
<td>D_{4} T_{6} : \text{If}(\text{AC} = 0) \text{ then } (\text{PC} \leftarrow \text{PC} + 1), \text{AC} \leftarrow \text{TR}, \text{SC} \leftarrow 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BPA</td>
<td>D_{5} T_{4} : \text{If}(\text{AC} = 0 \land \text{AC}(15) = 0)</td>
<td></td>
<td>then (\text{PC} \leftarrow \text{AR}), \text{SC} \leftarrow 0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

If $D_7 = 0$

Indirect = 1

$T_3$: $AR(12-15) \leftarrow 0$

$AR \leftarrow PC$, $PC \leftarrow PC + 1$

$T_3$: $AR \leftarrow M[AR]$

Execute memory-reference instruction

---

$5-16$

(a)

15  
PC 0

15  
AR 0

15  
TR 0

...  
IR 0

(b)

Memory

8-bits

8-bits

8-bits

8-bits

(c)

$T_0$: IR $\leftarrow M[PC]$, $PC \leftarrow PC + 1$

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

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

$T_3$: DR $\leftarrow M[AR]$
5-11
\[ IR = \begin{array}{cccc}
6 \text{ bits} & 4 \text{ bits} & 6 \text{ bits} & 14 \text{ bits} \\
\text{opcode 1} & \text{address 1} & \text{opcode 2} & \text{address 2}
\end{array} \]

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) \textit{Bun 2300}
(b) \textit{JON Bun O I} (Branch indirect with address O)

5-19
[Diagram of a computer memory system with inputs, outputs, and logic gates]
\[ J_F = xT_3 + 3T_2 + wT_{5G} \quad K_F = yT_1 + 3T_2 + wT_{5G} ' \]

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

\[
\text{INR}(pc) = R'T_1 + R'T_2 + D_6T_6 \quad Z_{DR} + PB_9(FGI) + PB_8(FGO) + rB_4(A_{C15})' + rB_3(A_{C15}) + rB_2Z_{AC} + rB_1E'
\]

\[
LD(pc) = D_4T_4 + D_5T_5
\]

\[
\text{CLR}(pc) = RT_1
\]

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

5-22 Write: \( D_3T_4 + D_5T_4 + D_6T_6 + RT_1 \) (M[AR] \( \leftarrow xx \))

5-23 \( (T_6 + T_1 + T_2)'(IEN)(FGI+FGO) \); \( R \leftarrow 1 \)

\[
RT_2 : R \leftarrow 0
\]
5-24

\[ x_2 \text{ places } PC \text{ onto the bus, From Table 5-6:} \]

\[ R'T_0 : AR \leq PC \]
\[ R'T_0 : TR \leq PC \]
\[ D_5T_4 : M[AP] \leq PC \]

\[ x_2 = R'T_0 + R'T_0 + D_5T_4 = (R' + R)T_0 + D_5T_4 = T_0 + D_5T_4 \]

5-25

\[ \text{From Table 5-6:} \]

\[ CLR(Sc) = R'T_2 + D_7T_3(I' + I) + (D_6 + D_1 + D_2 + D_5)T_5 \]
\[ + (D_3 + D_4)T_4 + D_6T_6 \]
6-1

\[
\begin{array}{cccc}
010 & CLA & 0000 & AC \\
011 & ADD 016 & C1A5 & PC \\
012 & BUN 014 & C1A5 & 012 \\
013 & HLT & 8184 & 013 \\
014 & AND 017 & 8184 & 014 \\
015 & BUN 013 & 8184 & 015 \\
016 & C1A5 & & 016 \\
017 & 73C6 & & 017
\end{array}
\]

\[
(C1A5)_{16} = \begin{array}{cccc}
1100 & 0001 & 1010 & 0101 \\
& & & \text{AND}
\end{array}
\]

\[
(93C6)_{16} = \begin{array}{cccc}
1001 & 0011 & 1100 & 0110 \\
& & & \text{AND}
\end{array}
\]

\[
1000 & 0011 & 1000 & 0100
\]

6-2

\[
\begin{array}{cccc}
100 & 5103 & \text{BSA 103} & \text{AC} \\
101 & 7200 & \text{CHA} & \text{AC} \\
102 & 7001 & \text{HLT} & \text{AC} \\
103 & 0000 & \text{5101} & \text{Answer} \\
104 & 7800 & \text{CLA} & \text{Answer} \\
105 & 7020 & \text{INC} & \text{Answer} \\
106 & C103 & \text{BUN 103} & \text{Answer}
\end{array}
\]

6-3

\[
\begin{array}{l}
\text{CLA} \\
\text{STA SUM} \{ \text{SUM} = 0 \} \\
\text{LDA SUM} \\
\text{ADD A} \{ \text{SUM} = \text{SUM} + A + B \} \\
\text{ADD B} \\
\text{STA SUM} \\
\text{LDA C} \\
\text{CHA} \\
\text{INC} \{ \text{DIF} = \text{DIF} - C \} \\
\text{ADD DIF} \\
\text{STA DIF} \\
\text{LDA SUM} \\
\text{ADD DIF} \{ \text{SUM} = \text{SUM} + \text{DIF} \} \\
\text{STA SUM}
\end{array}
\]

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

\[
\begin{array}{l}
\text{LDA A} \\
\text{ADD B} \\
\text{STA SUM} \\
\text{LDA C} \\
\text{CHA} \\
\text{INC} \\
\text{ADD DIF} \\
\text{STA DIF} \\
\text{ADD SUM} \\
\text{STA SUM}
\end{array}
\]
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 characters Hex binary

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>E</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>0100 0100 0100 0101</td>
</tr>
<tr>
<td>2</td>
<td>C</td>
<td>space</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>0100 0011 0010 0000</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>3</td>
<td>2</td>
<td>D</td>
<td>0010 1101 0011 0011</td>
</tr>
<tr>
<td>4</td>
<td>S</td>
<td>5 CR</td>
<td>3</td>
<td>5</td>
<td>0</td>
<td>0111 0101 0000 1101</td>
</tr>
</tbody>
</table>

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

\(-35 \rightarrow \begin{array}{cccccccc}
1 & 1 & 1 & 1 & 1 & 1 & 0 & 1
\end{array} \begin{array}{c}
0110 0101 0011 0011
\end{array} = (FF D D)_{16} \)
6-7 (c) LOP 105  
ADS 10B  
PTR 1DC  
NBR 10D  
CTR 10E  
SUM 10F  
\[(100)_{16} = (0000 0000 0110 0110)_{2}\]  
\[-(100)_{16} = (1111 1111 1001 1100)_{2}(FF9C)_{16}\]  
\[(75)_{10} = (0000 0000 0100 1011)_{16}(0048)_{16}\]  
\[(23)_{10} = (0000 0000 0001 0111)_{2}(0017)_{17}\]  

(b) Loc Hex ORG 100 Loc Hex  
100 2:10B LDA ADS 10B 0150 ADS, HEX:150  
101 3:10C STA PTR 10C 0000 PTR, HEX:0  
102 2:10D LDA NBR 10D FF9C NBR, DEC:-100  
103 3:10E STA CTR 10E 0000 CTR, HEX:0  
104 7:800 CLA 10F 0000 SUM, HEX:0  
105 9:10C LOP, ADD PTR I  
106 6:10C ISZ PTR 150 004B DEC:150  
107 6:10E ISZ CTR  
108 4:105 BUN LOP  
109 3:10F STA SUM 183 0017 DEC:23  
10A 7:001 HLT  

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

6-9

www.samy007.blogfa.com
6-10 (a) MRI Table  

<table>
<thead>
<tr>
<th>memory</th>
<th>word</th>
<th>symbol</th>
<th>HEX</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>AND</td>
<td>A</td>
<td>414D</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>D space</td>
<td>4420</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>value</td>
<td>0000</td>
</tr>
<tr>
<td>4</td>
<td>ADD</td>
<td>A D</td>
<td>4144</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>D space</td>
<td>4420</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>value</td>
<td>1000</td>
</tr>
</tbody>
</table>

(b) non-MRI Table  

<table>
<thead>
<tr>
<th>memory</th>
<th>word</th>
<th>symbol</th>
<th>HEX</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CLA</td>
<td>C</td>
<td>434C</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>A space</td>
<td>4120</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>value</td>
<td>7800</td>
</tr>
<tr>
<td>4</td>
<td>CLE</td>
<td>C L</td>
<td>434C</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>E space</td>
<td>4520</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>value</td>
<td>7400</td>
</tr>
</tbody>
</table>

etc.

6-11

LDA B  
CMA  
INC  
ADD A  / Form A-B  
SPA  / 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>memory</th>
<th>word</th>
<th>symbol</th>
<th>HEX</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>ORG</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>101</td>
<td></td>
<td>CLE</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td></td>
<td>CLA</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td></td>
<td>STA CTR /Initialize counter to zero</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td></td>
<td>LDA WRD</td>
<td></td>
</tr>
<tr>
<td>105</td>
<td></td>
<td>SZA</td>
<td></td>
</tr>
<tr>
<td>106</td>
<td></td>
<td>BUN ROT</td>
<td></td>
</tr>
<tr>
<td>107</td>
<td></td>
<td>BUN STP / Word is zero; stop with CTR=0</td>
<td></td>
</tr>
<tr>
<td>108</td>
<td></td>
<td>ROT, CIL / Bring bit to E</td>
<td></td>
</tr>
<tr>
<td>109</td>
<td></td>
<td>SZE</td>
<td></td>
</tr>
<tr>
<td>10A</td>
<td></td>
<td>BUN AGN / bit =1, go to count it</td>
<td></td>
</tr>
<tr>
<td>10B</td>
<td></td>
<td>BUN ROT / bit =0, repeat</td>
<td></td>
</tr>
<tr>
<td>10C</td>
<td></td>
<td>7400 AGN, CLE</td>
<td></td>
</tr>
<tr>
<td>110</td>
<td></td>
<td>6110 ISZ CTR / Increment counter</td>
<td></td>
</tr>
</tbody>
</table>
6-12 (b) Continued

100 7004  SZA  /check if remaining bits = 0
10E 4107  BUN  ROT  /No; rotate again
10E 7001  STP  HLT  /Yes; stop
110 0000  CTR,  HEX  0
111 62C1  WRD,  HEX  62C1

END

6-13  (100)_{16} = (256)_{10}  500 to SFF \rightarrow (256)_{10} locations

ORG 100
LDA  ADS
STA  PTN  /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  /Start with AC=0
ISZ  CTR
BUN  LOP  /Add multiplicand
HLT

A, DEC  -  /Repeat loop A times
B, DEC  -  /multiplier
CTR, HEX  0  /multiplicand
END

www.samy007.blogfa.com
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

```
LDP, CLE
LDA Y
CIR
STA Y
SZE
BUN ONE
BUN ZRO
```

same as beginning of Program in Table 6-14

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

Double-precision add

\( P \leftarrow X + P \)

Same as program in Table 6-15.

Continued next page.
6-16 continued

ZRO, LDA XL Double-precision
CIL STA XL left-shift XH+XL
LDA XH
CIL STA XH
ISZ CTR Repeat 16 times
BUN LOP
HLT

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

check sign of multiplier

plus

Y

minus

Y ← Y + 1

2's complement multiplier

X ← X + 1

2's complement multiplicand

Proceed as in Table 6-14
6-18 \[ C \leftarrow A-B \]
CLE
LDA BL
CMA
INC
ADD AL
STA CL

save carry
CLA
CIL
STA TMP
LDA BH
CMA
ADD AH

add carry → ADD TMP
STA CH
HLT

TMP, HEX 0

6-19 \[ z = x \oplus y = xy' + x'y = [(xy')', (x'y')']' \]
LDA Y
CMA AND X
CMA STA TMP
LDA X
CMA AND Y
CMA

6-20
LDA X
CLE
CIL
SIZE BUN ONE
SPA
BUN OVF
BUN EXT
ONE, SNA
BUN OVF
EXT, HLT

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.

\[ z = x \oplus y = xy' + x'y = [(xy')', (x'y')']' \]
LDA Y
CMA AND X
CMA STA Z
HLT

/ zero to low order bit; sign bit in E
6-21  Calling Program

BSA  SUB
HEX 1234 /Subtraction
HEX 4321 /minuend
HEX 0 /difference.

6-22

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 INST /subroutine Table 6-20
STA PTR I
ISZ PTR
ISZ CTR

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

CMA
INC
STA CTR
LOP
LDA PTR I
CMA
STA PTR I
ISZ PTR
ISZ CTR
BUN LOP
ISZ CMP
BUN CMP I

E AC
1 0000 0111 1001 1100 079C
1 1001 0000 0111 1001 9079

www.samy007.blogfa.com
LDA WRD
AND MS1
STA CH1
LDA WRD
AND MS2
CLE
BSA SR8 / subroutine to shift right eight times

6-25

STA CH2
HLT

6-26

Start

Initialize memory buffer

Load next double character from buffer into AC

Load double character again

AND AC with HEX FFOO

AND AC with HEX 0OFF

Compare AC with HEX 000D

Equal?

Equal?

Pack HEX 0D to LNE

END
<table>
<thead>
<tr>
<th>Location</th>
<th>Hex code</th>
<th>Instruction(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>32 13</td>
<td>SRV, STA SAC</td>
</tr>
<tr>
<td>201</td>
<td>70 80</td>
<td>CIR</td>
</tr>
<tr>
<td>202</td>
<td>32 14</td>
<td>STA SE</td>
</tr>
<tr>
<td>203</td>
<td>F2 00</td>
<td>SKI</td>
</tr>
<tr>
<td>204</td>
<td>42 09</td>
<td>BUN NXT</td>
</tr>
<tr>
<td>205</td>
<td>F8 00</td>
<td>INP</td>
</tr>
<tr>
<td>206</td>
<td>F4 00</td>
<td>OUT</td>
</tr>
<tr>
<td>207</td>
<td>82 15</td>
<td>STA PT1 I</td>
</tr>
<tr>
<td>208</td>
<td>62 15</td>
<td>ISZ PT1</td>
</tr>
<tr>
<td>209</td>
<td>F1 00</td>
<td>NXT, SKO</td>
</tr>
<tr>
<td>20A</td>
<td>42 08</td>
<td>BUN EXT</td>
</tr>
<tr>
<td>20B</td>
<td>A2 16</td>
<td>LDA PTR1 I</td>
</tr>
<tr>
<td>20C</td>
<td>F4 00</td>
<td>OUT</td>
</tr>
<tr>
<td>20D</td>
<td>62 16</td>
<td>ISZ PT2</td>
</tr>
<tr>
<td>20E</td>
<td>22 14</td>
<td>EXT, LDA SE</td>
</tr>
<tr>
<td>20F</td>
<td>70 40</td>
<td>CIL</td>
</tr>
<tr>
<td>210</td>
<td>22 13</td>
<td>LDA SAC</td>
</tr>
<tr>
<td>211</td>
<td>F0 80</td>
<td>ION</td>
</tr>
<tr>
<td>212</td>
<td>C0 00</td>
<td>BUN ZRO I</td>
</tr>
<tr>
<td>213</td>
<td>00 00</td>
<td>SAC,</td>
</tr>
<tr>
<td>214</td>
<td>00 00</td>
<td>SE,</td>
</tr>
<tr>
<td>215</td>
<td>00 00</td>
<td>PT1,</td>
</tr>
<tr>
<td>216</td>
<td>00 00</td>
<td>PT2,</td>
</tr>
</tbody>
</table>

---

6-28

SRV, STA SAC
CIR
STA SE
LDA MOD /check MOD
CMA
SZA
BUN NXT /MOD≠ all 1’s
ski
BUN NXT service input
INP
OUT Device
STA PT1 I
ISZ PT1
BUN EXT /MOD≠ 0

NXT, LDA MOD
SZA
BUN EXT
output
BUN EXT device
IN
LDA PTR2 I
OUT ISZ PTR2
EXT, continue as in Table 6-23
A microprocessor is a small size CPU (computer on a chip). A 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 microprogram.

![Diagram](https://www.samy007.blogfa.com)

- Frequency of each clock: \( \frac{1}{100 \times 10^{-9}} = \frac{1000}{100 \times 10^6} = 10 \text{ MHz} \)
- 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 = \(2^{10} \times 32\)

(a) 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.

(a) 0001000 = 8
(b) 0101100 = 44
(c) 0111100 = 60

Op code = 6 bits
Control memory 00 \(\begin{array}{c}
\times
\times
\times
\times
\times
\end{array}\)
Address = 11 bits

\(n\) inputs \(\rightarrow\) \(2^n \times m\) ROM \(\rightarrow\) \(m\) outputs

The ROM can be programmed to provide any desired address for 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 INCA
(b) 000 100 101 NOP
(c) 100 101 000 INCPC

7-12
(a) READ
    DR = M[AR] F2 = 100 000 100 101
    DRTAC
    AC = DR F3 = 101

(b) ACTDR
    DR = AC F2 = 101 000 100 101
    DRTAC
    AC = DR F1 = 100

(c) ARTPC
    PC = AR F3 = 110 Impossible.
    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
(a) 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 10000000
    41: 000 000 000 11 00 10000000
    42: 000 000 000 01 01 10000111
    43: 000 000 110 00 00 10000000
(d) 60: CLRAC, COM  U JMP INDIRECT
61: WRITE READ I CALL FETCH
62: ADD, SUB  S RET 63 (NEXT)
63: DRTAC, INCOR 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
ADM:  NOP  I CALL INDIRECT
READ  U JMP NEXT
DRTAC, INCOR U JMP NEXT
ADD  U JMP EXCHANGE+2
(Table 7-2)
<table>
<thead>
<tr>
<th>ORG 28</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>BTCL</td>
<td>NOP</td>
<td>I</td>
<td>CALL</td>
<td>INDRCT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>READ</td>
<td>U</td>
<td>JMP</td>
<td>NEXT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DRTAC, ACTDR</td>
<td>U</td>
<td>JMP</td>
<td>NEXT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>COM</td>
<td>U</td>
<td>JMP</td>
<td>ANDOP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ORG 32</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BZ</td>
<td>NOP</td>
<td>Z</td>
<td>JMP</td>
<td>ZERO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NOP</td>
<td>U</td>
<td>JMP</td>
<td>FETCH</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ZERO</td>
<td>NOP</td>
<td>I</td>
<td>CALL</td>
<td>INDRCT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ARTPC</td>
<td>U</td>
<td>JMP</td>
<td>FETCH</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ORG 36</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SEQ</td>
<td>NOP</td>
<td>I</td>
<td>CALL</td>
<td>INDRCT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>READ</td>
<td>U</td>
<td>JMP</td>
<td>NEXT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DRTAC, ACTDR</td>
<td>U</td>
<td>JMP</td>
<td>NEXT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>XOR (H SUB)</td>
<td>U</td>
<td>JMP</td>
<td>BEQ1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ORG 69</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BEQ1</td>
<td>DRTAC, ACTDR</td>
<td>Z</td>
<td>JMP</td>
<td>EQUAL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NOP</td>
<td>U</td>
<td>JMP</td>
<td>FETCH</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EQUAL</td>
<td>INC PC</td>
<td>U</td>
<td>JMP</td>
<td>FETCH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ORG 40</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BPNZ</td>
<td>NOP</td>
<td>S</td>
<td>JMP</td>
<td>FETCH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NOP</td>
<td>Z</td>
<td>JMP</td>
<td>FETCH</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NOP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ARTPC</td>
<td>U</td>
<td>JMP</td>
<td>FETCH</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
ISZ: NOP I CALL INDRCT
READ U JMP NEXT
INCDIR U JMP NEXT
DRTAC, ACTDR U JMP NEXT (by fast INDIRCT)
DRTAC, ACTDR Z JMP ZERO
WRITE U JMP FETCH
ZERO: WRITE, INCP C U JMP FETCH

7-18

BSA: NOP I CALL INDRCT
PCTDR, ARTCP U JMP NEXT
WRITE, INCP C U JMP FETCH

7-19

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

FROM:
F3 output 6
F3 output 5

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

4 bits

7-20

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

(a) 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

(b) 001 000 010 010 000 000

7-21

4 4 5 3 = 20 bits total

(c) 0101 0110 0100 00100 000

R4 ← R5 + R6
7-22

<table>
<thead>
<tr>
<th>I₀</th>
<th>I₁</th>
<th>I₂</th>
<th>T</th>
<th>S₀</th>
<th>S₁</th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>0 0</td>
<td>0 1</td>
<td>0 1</td>
<td>A₀</td>
<td>A₀</td>
<td>0 1</td>
</tr>
<tr>
<td>0 0</td>
<td>0 1</td>
<td>0 0</td>
<td>0 0</td>
<td>INC(0)</td>
<td>INC(0)</td>
<td>0 0</td>
</tr>
<tr>
<td>0 0</td>
<td>1 0</td>
<td>0 1</td>
<td>0 1</td>
<td>A₁</td>
<td>A₁</td>
<td>0 1</td>
</tr>
<tr>
<td>0 0</td>
<td>1 1</td>
<td>0 0</td>
<td>0 0</td>
<td>A₀</td>
<td>A₀</td>
<td>0 0</td>
</tr>
<tr>
<td>0 1</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>INC(0)</td>
<td>INC(0)</td>
<td>0 0</td>
</tr>
<tr>
<td>0 1</td>
<td>0 1</td>
<td>0 1</td>
<td>0 1</td>
<td>A₂</td>
<td>A₂</td>
<td>0 1</td>
</tr>
<tr>
<td>0 1</td>
<td>1 0</td>
<td>0 0</td>
<td>0 0</td>
<td>A₀</td>
<td>A₀</td>
<td>0 0</td>
</tr>
<tr>
<td>0 1</td>
<td>1 1</td>
<td>0 0</td>
<td>0 0</td>
<td>A₀</td>
<td>A₀</td>
<td>0 0</td>
</tr>
<tr>
<td>1 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>INC(0)</td>
<td>INC(0)</td>
<td>0 0</td>
</tr>
<tr>
<td>1 0</td>
<td>0 1</td>
<td>0 1</td>
<td>0 1</td>
<td>A₀</td>
<td>A₀</td>
<td>0 0</td>
</tr>
<tr>
<td>1 0</td>
<td>1 0</td>
<td>0 0</td>
<td>0 0</td>
<td>A₀</td>
<td>A₀</td>
<td>0 0</td>
</tr>
<tr>
<td>1 0</td>
<td>1 1</td>
<td>0 0</td>
<td>0 0</td>
<td>A₀</td>
<td>A₀</td>
<td>0 0</td>
</tr>
<tr>
<td>1 1</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>INC(0)</td>
<td>INC(0)</td>
<td>0 0</td>
</tr>
<tr>
<td>1 1</td>
<td>1 0</td>
<td>0 0</td>
<td>0 0</td>
<td>A₀</td>
<td>A₀</td>
<td>0 0</td>
</tr>
<tr>
<td>1 1</td>
<td>1 1</td>
<td>0 0</td>
<td>0 0</td>
<td>A₀</td>
<td>A₀</td>
<td>0 0</td>
</tr>
</tbody>
</table>

7-23

(a) 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 + 0 = G.

When P = 1, T = G' because G + 1 = G'.

When G is the value of the selected bit in MUX2.
8-1
(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.
(e) \[ \begin{array}{cccc}
\text{SELA} & \text{SELB} & \text{SELD} & \text{OPR} \\
4 & 4 & 6 & = 18 \text{ bits} \\
\end{array} \]

8-2
$30 + 80 + 10 = 120$ nSec.
(The decoder signals propagate at the same as the muxs.)

8-3
\[
\begin{array}{|c|c|c|c|c|}
\hline
\text{Control word} & \text{SELA} & \text{SELB} & \text{SELD} & \text{OPR} \\
\hline
(\text{a}) & 001 & 010 & 011 & 00101 \\
(\text{b}) & 000 & 100 & 000 & 00000 \\
(\text{c}) & 010 & 010 & 010 & 01100 \\
(\text{d}) & 000 & 001 & 000 & 00010 \\
(\text{e}) & 111 & 100 & 011 & 100000 \\
\hline
\end{array}
\]

8-4
\[
\begin{array}{|c|c|c|c|c|}
\hline
\text{Control word} & \text{SELA} & \text{SELB} & \text{SELD} & \text{OPR} \\
\hline
(\text{a}) & 001 & 010 & 011 & 00101 \\
(\text{b}) & 000 & 100 & 000 & 00000 \\
(\text{c}) & 010 & 010 & 010 & 01100 \\
(\text{d}) & 000 & 001 & 000 & 00010 \\
(\text{e}) & 111 & 100 & 011 & 100000 \\
\hline
\end{array}
\]

8-5
(a) Stack full with 64 items.
(b) Stack empty.
8-6

\[ \text{PUSH: } M[SP] \leftarrow DR \]
\[ SP \leftarrow SP-1 \]
\[ \text{POP: } SP \leftarrow SP+1 \]
\[ DR \leftarrow M[SP] \]

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) \[ ABCDE + * * FGH + * \]

8-8

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

8-9

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

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

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 from 7 to 0

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

WRITE: (if not full):
\[ M[WC] \leftarrow DR \]
\[ WC \leftarrow WC + 1 \]
\[ ASC \leftarrow ASC + 1 \]

READ: (if not empty)
\[ DR \leftarrow M[RC] \]
\[ RC \leftarrow RC + 1 \]
\[ ASC \leftarrow ASC - 1 \]

Empty \[ \text{J} = \text{all } 0\text{'s} \]
Full \[ \text{J} = \text{all } 1\text{'s} \]

Memory may wrap-around from 7 to 0
8-11

\[ 8 \quad 12 \quad 12 = 32 \text{ bits} \]

Two address instructions

\[ 2^8 = 256 \text{ combinations.} \]

\[ 256 - 250 = 6 \text{ combinations can be used for one address} \]

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ 6 \times 2^12 ]</td>
<td></td>
</tr>
</tbody>
</table>

Maximum number of one address instruction:

\[ 6 \times 2^{12} = 24576 \]

8-12

(d) RPN:

\[ X \ A \ B - C + D E * F - G H K * + / = \]

8-13

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

address = 18 bits

Mode = 3

Register = 6

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Mode</th>
<th>Register</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>3</td>
<td>6</td>
<td>18 = 32</td>
</tr>
</tbody>
</table>

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 = 00001110101; -251 = 11110000101

(c) \[ \text{PC} = 751 = 00101110111; \ 500 = 00011110100 \]

\[ \text{PC} = 751 = 00101110111; \ 500 = 00011110100 \]

\[ \text{RA} = -251 = \underline{1}11100000101 \]

\[ \text{EA} = 500 = 00011110100 \]

www.samy007.blogfa.com
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

(a) Direct: 400

(b) Immediate: 301

(c) Relative: 302 + 400 = 702

(d) Reg. Indirect: 200

(e) Indexed: 200 + 400 = 600

8-19

1 = c 0 = c 1 = c 0 = Reset initial carry

6E C3 56 7A

13 55 6B 8F

82 18 C2 09 Add with carry

8-20

\[
\begin{align*}
10011100 \text{ AND} & \quad 10101010 \text{ OR} & \quad 10011100 \\
10101010 \text{ OR} & \quad 10011100 \text{ XOR} & \quad 10101010
\end{align*}
\]

8-21

(a) AND with: 0000000011111111

(b) OR with: 0000000011111111

(c) XOR with: 0001111111110000
8-22
\[
\begin{align*}
\text{Initi} & : 0 1 1 1 0 1 1 \quad C = 1 \\
\text{SHR} & : 0 0 1 1 1 1 0 1 \\
\text{SHL} & : 1 1 1 1 0 1 1 0 \\
\text{SHRA} & : 0 0 1 1 1 1 0 1 \\
\text{SHLA} & : 1 1 1 1 0 1 1 0 \quad (\text{overflow}) \\
\text{ROR} & : 1 0 1 1 1 1 0 1 \\
\text{ROL} & : 1 1 1 1 0 1 1 0 \\
\text{ROEC} & : 1 0 1 1 1 1 0 1 \\
\text{POLC} & : 1 1 1 1 0 1 1 1 \\
\end{align*}
\]

8-23
\[
\begin{align*}
83 - 0 & 1 0 1 0 0 1 1 \quad -83 = 1 0 1 0 1 1 0 1 \\
+68 & = 0 1 0 0 0 1 0 0 \quad -68 = 1 0 1 1 1 1 0 0 \\
\text{(a)} -83 & + 1 0 1 0 1 1 0 1 \quad \text{(b)} -68 & + 1 0 1 1 1 1 0 0 \\
+68 & = 0 1 0 0 0 1 0 0 \quad \text{10} & = \text{carries} \\
-15 & = 1 1 1 1 0 0 0 1 \quad -83 & + 1 0 1 0 1 1 0 1 \\
(\text{two's complement}) & \quad -151 & = 0 1 1 0 1 0 0 1 \\
\quad & \quad -128 & = \text{(overflow)} \\
\end{align*}
\]

8-24
\[
\begin{align*}
Z & = F_0 F_1 F_2 F_3 F_4 F_5 F_6 F_7 = (F_0 + F_1 + F_2 + F_3 + F_4 + F_5 + F_6 + F_7)'
\end{align*}
\]

8-25
\[
\begin{align*}
\text{(a)} & \quad 72 \ 0 1 1 1 0 0 1 0 \quad (b) & \quad 72 \ 0 1 1 1 0 0 1 0 \\
& \quad C = 0 \quad & \quad C = 0 \quad S = 1 \quad Z = 0 \quad V = 1 \\
\quad \text{c6} & \quad \text{c6} \ 1 1 0 0 0 1 1 0 \quad \text{c6} \ 1 1 1 1 1 1 0 0 \\
\quad 13 & \quad 0 0 1 1 0 0 0 \quad \text{90} \quad 1 0 0 1 0 0 0 0 \\
\quad c = 1 \quad s = 0 \quad z = 0 \quad v = 0 & \quad \text{c = 1} \quad s = 0 \quad z = 0 \quad v = 0 \\
\end{align*}
\]

\[
\begin{align*}
\text{(c)} & \quad 9A = 1 0 0 1 1 0 1 0 1 \quad \text{(d)} & \quad 72 \ 0 1 1 1 0 0 1 0 \\
& \quad 0 1 0 0 1 0 1 0 \quad 8D \ 1 0 0 0 1 1 0 0 \\
\quad 72 & \quad 0 1 1 1 0 0 1 0 \quad \text{00} \quad 0 0 0 0 0 0 0 0 \\
& \quad D8 \ 1 1 0 1 0 0 0 \quad \text{c = 0} \quad s = 0 \quad z = 1 \quad v = 0 \\
\quad & \quad \text{(b) s = 1} \quad \text{z = 0} \quad \text{v = 1} \quad \text{(e)} & \quad \text{c = 0} \quad s = 0 \quad z = 1 \quad v = 0 \\
\end{align*}
\]

www.samy007.blogfa.com
8-26

\[ \text{if } A < B, \text{ then } c = 0 \ \text{if } A \geq B \]\[ z = 1 \ \text{if } A = B, \text{ therefore } z = 1 \ \text{if } A \neq B \]

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

\( A \geq B \) 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 \( A \geq B \) \((A - B \text{ negative})\)

then \( s = 1 \) if \( v = 0 \) \((s \oplus v) = 1\)

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

\( A > B \) 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

\[ c \]
\[ z \]
\[ s \]
\[ v \]

\[ c'z' = 1 \]
\[ c = 0 \]
\[ c = 1 \]
\[ c + z = 1 \]
\[ z = 1 \]
\[ z = 0 \]
\[ [s \oplus v] + z = 1 \]
\[ s \oplus v = 0 \]
\[ s \oplus v = 1 \]
\[ [(s \oplus v) + z] = 1 \]
8-29

\[
\begin{array}{c|c|c}
8-29 & \text{Unsigned} & \text{Signed} \\
\hline
A = 01000001 & 65 & +65 \\
B = 10000100 & 132 & -124 \\
A + B = 11000101 & 197 & -59 \\
\end{array}
\]

(c) \( C = 0 \), \( Z = 0 \), \( S = 1 \), \( V = 0 \)
(d) \( BNC \), \( BNZ \), \( BM \), \( BNV \)

8-30

(a) \( A = 01000001 = 65 \)
\( B = 10000100 = 132 \)
\( A - B = 10111101 = -67 \) (2's comp. of 01000011)

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

(c) \( BL, BLE, BNE \)

8-31

(a) \( A = 01000001 = +65 \)
\( B = 10000100 = -124 \)
\( A - B = 10111101 + 189 = 01011101 \)

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

(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."

8-35
(a) $SP \leftarrow SP-1$
$M[SP] \leftarrow PSW$
$SP \leftarrow SP-1$
$M[SP] \leftarrow PC$
$TR \leftarrow IAD$ (TR is a temporary register)
$PSW \leftarrow M[TR]$
$TR \leftarrow TR+1$
$PC \leftarrow M[TR]$
Go to fetch phase.

8-36
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

(b) XOR $R22, #1, R22

(c) SUB $R0, R22, R22

(d) ADD $R0, $R0, R22

(e) SRA $R22, #2, R22

(f) OR $R1, $R1, $R1

or ADD $R1, $R0, $R1

or SLL $R1, #0, $R1

$R22 ← $R22 - 1 (Subtract 1)

$R22 ← $R22 ⊕ all 1's ($x ⊕ 1 = x'$)

$R22 ← 1 - 1

$R22 ← 1 + 1

Arithmetic shift right twice

$R1 ← R1 \lor R1

$R1 ← R1 + 0

Shift left 0 times

8-39
(a) JMP $Z, #3200, (R0)

(b) JMP $R Z, -200

PC ← 0 + 3200

PC ← 3400 + (-200)
CHAPTER 9

9.1

\[ \begin{array}{c}
A_t \quad R_1 \\
B_t \quad R_2 \\
C_t \quad R_3 \\
D_t \quad R_4 \\
\text{Adder} \\
R_5 \\
\text{Multiplier} \\
R_7 \\
\end{array} \]

9.2

\[
\begin{array}{cccccccccccc}
\text{Segment} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 \\
1 & T_1 & T_2 & T_3 & T_4 & T_5 & T_6 & T_7 & T_8 \\
2 & T_1 & T_2 & T_3 & T_4 & T_5 & T_6 & T_7 & T_8 \\
3 & T_1 & T_2 & T_3 & T_4 & T_5 & T_6 & T_7 & T_8 \\
4 & T_1 & T_2 & T_3 & T_4 & T_5 & T_6 & T_7 & T_8 \\
5 & T_1 & T_2 & T_3 & T_4 & T_5 & T_6 & T_7 & T_8 \\
6 & T_1 & T_2 & T_3 & T_4 & T_5 & T_6 & T_7 & T_8 \\
\end{array}
\]

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

9.3

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

9.4

\[
t_n = 50 \text{ ns} \\
k = 6 \\
t_p = 10 \text{ ns} \\
\gamma = 100 \\
S = \frac{n \cdot t_n}{(k + \gamma - 1) t_p} = \frac{100 \times 50}{(6 - 99) \times 10} = 4.76 \\
S_{\text{max}} = \frac{t_n}{t_p} = \frac{50}{10} = 5
\]

www.samy007.blogfa.com
(a) \( t_p = 45 + 5 = 50 \) ns \( k = 3 \)

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

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

\[ S = \frac{100 \times 100}{(3+99)50} = 1.96 \] for \( n = 100 \)

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

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 \( = \frac{k+n-1}{n} t_p = \frac{(n+6)30}{n} \)

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 \( = \frac{(k+n-1) t_p}{4} = 100 \)

\( = 10,300 \) ns \( = 10.3 \mu s \)

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

\( (k+n-1) t_p = (5 + 99) 55 = 5,720 \) ns \( = 5.72 \) ms
Connect output of adder to input Ex2b in a feedback path and use input Ax2a 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: (1) 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 | I A E |
| 102 | Branch to 104 | I A E |
| 103 | Increment R1 | I A E |
| 104 | Store R1     | I A E |

9-15 Use example of Problem 9-14.

| 101 | Branch to 105 | I A E |
| 102 | Add R2 to R3 | I A E |
| 103 | No-operation | I A E |
| 104 | Increment R1 | I A E |
| 105 | Store R1     | I A E |

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 uses 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

\[
\frac{250 \times 10^9}{100 \times 10^6} = 2,500 \text{ sec} = 41.67 \text{ minutes}
\]

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: \[400 \times 40 = 4,000 \text{ nsec}\]

www.samy007.blogfa.com
$2^{6-1} = 63$, Overflow if sum greater than $63$

(a) $(+45) + (+31) = 76$  \[1\ 3\ 7\]  path, AVF = 1

(b) $(-31) + (-45) = -76$  \[1\ 3\ 7\]  AVF = 1

(c) $+45 - (+31) = 14$  \[2\ 6\ 9\ 10\]  AVF = 0

(d) $+45 - (+45) = 0$  \[2\ 6\ 9\ 11\]  AVF = 0

(e) $-31 - (+45) = -76$  \[2\ 5\ 7\]  AVF = 1
10-3

\[
\begin{align*}
(a) & \quad +35 & 0 & 100011 \\
& +40 & 0 & 101000 \\
& +75 & 1 & 001011 \\
& F=0 & E=1 & \text{Carries} \\
(b) & -35 & 1 & 011101 \\
& -40 & 1 & 011000 \\
& -70 & 0 & 110101 \\
& F=1 & E=0 \\
\end{align*}
\]

\[F \oplus E = 1; \text{ overflow}\]

\[F \oplus E = 1; \text{ overflow}\]

\[10-4\]

<table>
<thead>
<tr>
<th>Case</th>
<th>Operation in sign-magnitude</th>
<th>Operation in sign-2's complement</th>
<th>Required result in sign-2's complement</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>((+X) + (+Y))</td>
<td>((0+X)+(0+Y))</td>
<td>(0+(X+Y))</td>
</tr>
<tr>
<td>2.</td>
<td>((+X) + (-Y))</td>
<td>((0+X)+2^k+(2^k-Y))</td>
<td>(0+(X-Y)) if (X \geq Y) (2^k+2^k-(Y-X)) if (X &lt; Y)</td>
</tr>
<tr>
<td>3.</td>
<td>((-X) + (+Y))</td>
<td>(2^k+(2^k-X)+(0+Y))</td>
<td>(0+(Y-X)) if (Y \geq X) (2^k+2^k-(X-Y)) if (Y &lt; X)</td>
</tr>
<tr>
<td>4.</td>
<td>((-X) + (-Y))</td>
<td>((2^k+2^k-X)+(2^k+2^k-Y))</td>
<td>(2^k+2^k-(X+Y))</td>
</tr>
</tbody>
</table>

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 \((X-Y) \geq 0\) 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: \(2^k+2^k+(X-Y)\).

Discard carry \(2^k\) to obtain result of (c): \(2^k+(2^k-Y-X)\).
Transfer Augend sign into Ts, then add: \( AC \leftarrow AC + BR \)
As will have sign of sum.

Truth Table for combin. circuit

<table>
<thead>
<tr>
<th>Ts</th>
<th>B_s</th>
<th>A_s</th>
<th>V</th>
<th>change of sign</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0, change of sign</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0, quantities subtracted</td>
</tr>
<tr>
<td>0</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>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</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)

\(-9 \quad 1 \quad 0110\)
\(-6 \quad 1 \quad 1001\)
\(-15 \quad \boxed{0} \quad 1111\)

\(F = \boxed{1}\) \(E = 0\) \(\leftarrow\) Carries

\(E + F = 1\) but there should be no overflow since result is -15.

(b) The procedure \( V \leftarrow E + F \) is valid for 1's complement numbers provided we check the result \(0 \quad 1111...11\) when \(V = 1\). Algorithm

Add

\(AC \leftarrow AC + BR\)
\(V \leftarrow E + F\)

10-7 Add algorithm flowchart is shown above (Prob. 10-66)
Maximum value of numbers is $r^n-1$. It is necessary
to show that maximum product is less than or equal
to $r^n-1$. Maximum product is:

$$(r^n-1)(r^n-1) = r^{2n} - 2r^n + 1 \leq r^n-1$$

which gives: $2 \leq 2r^n$ or $1 \leq r^n$

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

**10-9** Multiplicand $B = 11111 = (31)_{10}$

Multiplication:

**Multiplier in Q:**

$A = 00000$

$Q = 10010$

$Q_n = 1$, and $B = 11111$

**$E = 0$**

**shr EAQ:**

$01111$ $11010$ $100$

$Q_n = 0$, shr EAQ: $00111$ $11101$ $011$

$Q_n = 1$, add $B$: $11111$ $100110$

**shr EAQ:**

$01001110110$ $010$

$Q_n = 0$, shr EAQ: $010011110111001$

$Q_n = 1$, add $B$: $11111$ $101000$

**shr EAQ:**

$0100010111000$

$(651)_{10}$

**10-10(a)**

$$\frac{10100011}{1101} = 110 + \frac{1001}{1011}$$

$$16 \frac{3}{11} = 14 + \frac{9}{11}$$

$B = 1011$, $B+1 = 0101$

**DVF = 0**

**Dividend in AQ:**

$E = 0$

**shr EAQ:**

$1010$ $0011$ $Q$

$S = 100$

$E = 1$, set $Q_n$ to $1$: $1001$ $0111$ $011$

**shr EAQ:**

$0010$ $1110$

$E = 1$, set $Q_n$ to $1$: $0111$ $1111$ $010$

**add $B+1$, suppress carry:**

$E = 1$, set $Q_n$ to $1$: $0101$ $1111$

**shr EAQ:**

$0101$

$E = 1$, set $Q_n$ to $1$: $0100$ $1111$ $001$

**add $B+1$, carry to $E$:**

$E = 1$, set $Q_n$ to $1$: $0101$

**shr EAQ:**

$0101$ $1110$

$E = 0$, leave $Q_n = 0$: $0110$ $1110$

**add $B$:**

$0111$

**restore remainder:**

$000$

$\text{remainder}$ $000$

$\text{quotient}$

www.samy007.blogfa.com
\[ \frac{10_{10}}{10_{11}} = 0.101 \quad 3 = 0.011 \quad \overline{3} + 1 = 10.01 \]

<table>
<thead>
<tr>
<th>Dividend in (Q), (A = 0)</th>
<th>(E)</th>
<th>(A)</th>
<th>(Q)</th>
<th>(SC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shl (EAQ)</td>
<td></td>
<td>0000</td>
<td>1111</td>
<td>100</td>
</tr>
<tr>
<td>add (B+1)</td>
<td></td>
<td>1101</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(E=0), leave (Q_n=0)</td>
<td></td>
<td>1110</td>
<td>1110</td>
<td></td>
</tr>
<tr>
<td>add (B)</td>
<td></td>
<td>0011</td>
<td></td>
<td></td>
</tr>
<tr>
<td>restore partial remainder</td>
<td></td>
<td></td>
<td></td>
<td>011</td>
</tr>
<tr>
<td>Shl (EAQ)</td>
<td></td>
<td>0011</td>
<td>1100</td>
<td></td>
</tr>
<tr>
<td>add (B+1)</td>
<td></td>
<td>1101</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(E=1), set (Q_n) to 1</td>
<td></td>
<td>0000</td>
<td>1101</td>
<td>010</td>
</tr>
<tr>
<td>Shl (EAQ)</td>
<td></td>
<td>0011</td>
<td>1010</td>
<td></td>
</tr>
<tr>
<td>add (B+1)</td>
<td></td>
<td>1101</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(E=0), leave (Q_n=0)</td>
<td></td>
<td>1110</td>
<td>1010</td>
<td></td>
</tr>
<tr>
<td>add (B)</td>
<td></td>
<td>0011</td>
<td></td>
<td></td>
</tr>
<tr>
<td>restore partial remainder</td>
<td></td>
<td></td>
<td></td>
<td>001</td>
</tr>
<tr>
<td>Shl (EAQ)</td>
<td></td>
<td>0001</td>
<td>0100</td>
<td></td>
</tr>
<tr>
<td>add (B+1)</td>
<td></td>
<td>0101</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(E=1), set (Q_n) to 1</td>
<td></td>
<td>0000</td>
<td>0101</td>
<td>000</td>
</tr>
</tbody>
</table>

**10-11**

\(A + B+1\) performs: \(A + 2^\text{ lowers} - B = 2^\text{ lowers} + A - B\)

Adding \(B\): \((2^\text{ lowers} + A - B) + B = 2^\text{ lowers} + A\)

Remove end-carry \(2^\text{ lowers}\) to obtain \(A\).

**10-12**

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 \(B\):

\[
\frac{+52}{+5} = +10 + \frac{+2}{+5} = 10.4 \quad \frac{-52}{+5} = -10 + \frac{-2}{+5} = -10.4
\]

\[
\frac{+52}{-5} = -10 + \frac{+2}{-5} = -10.4 \quad \frac{-52}{-5} = +10 + \frac{-2}{-5} = +10.4
\]

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

**10-13**

Add one more stage to Fig. 10-10 with hand gates and a 4-bit adder.
(a) \((+15) \times (+13) = +195\) = \((0110 0000 1011)_2\)

\[
\begin{array}{c|c|c|c|c|c}
Q_n & Q_{n+1} & AC & QR & Q_{n+1} & SC \\
\hline
10 & \text{Initial} & 00000 & 01101 & 0 & 101 \\
1 & \text{Subtract BR} & 10001 & & & \\
\hline
0 & 0 & \text{ashr} & 10001 & 10110 & 1 & 100 \\
1 & \text{Add BR} & 01111 & & & \\
\hline
10 & \text{Subtract BR} & 10001 & & & \\
\hline
& \text{ashr} & 11010 & 01101 & 1 & 010 \\
\hline
11 & \text{ashr} & 11101 & 00110 & 1 & 001 \\
\hline
01 & \text{Add BR} & 11111 & & & \\
\hline
& \text{ashr} & 01100 & 00011 & 0 & 000 \\
\hline
\end{array}
\]

\[+195\]

(b) \((+15) \times (-13) = -195\) = \((1100 111101)_2\)

\[
\begin{array}{c|c|c|c|c|c}
Q_n & Q_{n+1} & AC & QR & Q_{n+1} & SC \\
\hline
10 & \text{Initial} & 00000 & 10011 & 0 & 101 \\
\hline
1 & \text{Subtract BR} & 10001 & & & \\
\hline
\hline
0 & 1 & \text{ashr} & 11000 & 11001 & 1 & 100 \\
\hline
11 & \text{ashr} & 11100 & 01100 & 1 & 011 \\
\hline
01 & \text{Add BR} & 01111 & & & \\
\hline
\hline
& \text{ashr} & 00101 & 10110 & 0 & 010 \\
\hline
\hline
00 & \text{ashr} & 00010 & 11011 & 0 & 001 \\
\hline
10 & \text{Subtract BR} & 10011 & & & \\
\hline
& \text{ashr} & 11001 & 11101 & 1 & 010 \\
\hline
\hline
\end{array}
\]

\[-195\]
Dividend in AQ
Divisor in B

QS ← A + B
SC ← n - 1

EA ← A + B + 1

= 0

EA ← A + B
DVF ← 1

END (Divide overflow)

= 0

shl EAQ
Qn ← E

= 0

shl Q, Qn ← E

= 1

AE ← A + B

= 0

Qn

AE ← A + B + 1

SC ← SC - 1

= 0

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

**General comments:**

1. For \( k \) bits in \( A \) (\( k \) even), \( Q \) will have \( \frac{k}{2} \) bits:
   \[
   Q = q_1q_2q_3\ldots q_{\frac{k}{2}}
   \]

2. The first test value is 01
   The second test value is 001,01
   The third test value is 0001,01
   The fourth test value is 00001,01 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:

\[
\begin{array}{c|c|c|c|c|c}
9_1 & 9_2 & 9_3 & 9_4 \\
\hline
1 & 1 & 0 & 1 & 1 & = Q = 13 \\
\hline
\end{array}
\]
\[
\begin{array}{c}
\sqrt{10101101} = A = 169 \\
\hline
01 & \text{subtract first test value 01} \\
01 & \text{Answer positive; let } q_1 = 1 \\
\hline
0110 & \text{bring down next pair} \\
0101 & \text{subtract second test value 01,01} \\
0001 & \text{Answer positive; let } q_2 = 1 \\
0001 & \text{bring down next pair} \\
001101 & \text{subtract third test value 001,01} \\
001101 & \text{Answer negative; let } q_3 = 0 \\
000110 & \text{restore partial remainder} \\
00011001 & \text{bring down next pair} \\
00011001 & \text{subtract fourth test value 0001,01} \\
\hline
\text{Remainder} = 00000 \text{ answer positive (zero); let } q_4 = 1
\end{array}
\]
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>

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

$BS = B_5 B_4 B_3 B_2 B_1 B_0 .... B_n$

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  A_5 B_5 =01

A<B   =11  =00

A>B

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

(b) A > 0 if B > A

(b) A < 0 if B < A

AC = AC + BR + 1

AC = AC + BR + 1

(A - B) > 0 if A > B

(A - B) < 0 if A < B

=1  A_5 = 0

=0 A = 0

A>B

A<B

A=B

A=B

A>B

A=B

A=B
```
10-18 (b)

\[ A_0, A_1, A_2, \ldots, A_n \]
\[ +2: 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0 \]
\[ +1: 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \]
\[ 0: 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \]
\[ -1: 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \]
\[ -2: 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 0 \]
\[ -3: 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 0 \quad 1 \quad 0 \]

10-19

\[ A = A_0, A_1, A_2, A_3, \ldots, A_n \]
\[ B = B_0, B_1, B_2, B_3, \ldots, B_n \]

(a) \[ = 10 \quad A_0 B_0 \]
\[ A < B: 0 \quad = 01 \quad A > B \]
\[ = 11 \quad = 00 \]
\[ EA = A + B + 1 \]
\[ A = 0 \quad E = 0 \]
\[ A < B \quad A = B \quad A > B \]

(b) \[ = 10 \quad A_0 B_0 \]
\[ A < B: 11 \quad = 00 \quad A > B \]
\[ = 01 \quad = 10 \]
\[ EA = A + B + 1 \]
\[ A = 0 \quad E = 0 \]
\[ A < B \quad A = B \quad A > B \]

10-20

**Fig. 10-8**

mantissa alignment

\[ SC \leftarrow n - 1 \]

\[ a < b \quad a > b \]
\[ \text{shr} A \quad a \leftarrow a - 1 \quad \text{shr} B \quad b \leftarrow b + 1 \]
\[ \text{same as Fig. 10-8} \quad b \leftarrow b + 1 \]
\[ SC \leftarrow SC - 1 \quad SC \leftarrow SC - 1 \]
\[ \neq 0 \quad \neq 0 \quad \neq 0 \]
\[ SC = 0 \quad END = 0 \quad SC \neq 0 \]

\[ AC \leftarrow BR \]
\[ \text{add} \quad \text{subtract} \]
\[ A_0 \leftarrow \bar{A}_0 \]

www.samy007.blogfa.com
10-21 Let "e" be a flip-flop that holds end-carry after exponent addition.

\[
\begin{align*}
\text{start} \\
\text{check for zeros} \\
& \text{sub} \quad B_s \leftarrow B_s \\
& \text{op} \quad a \oplus b \\
& \text{add} \quad ea \leftarrow a + b + 1
\end{align*}
\]

\[
\begin{align*}
& \text{if } a < b \\
& \quad \text{restore } a \leftarrow a + b \\
& \quad \text{swap } BR \leftarrow AC \\
& \quad \quad AC \leftarrow BR \\
& \quad \text{if } a < b \quad a \leftarrow a + b + 1
\end{align*}
\]

\[
\begin{align*}
& \text{if } a < b \\
& \quad \text{Shr A} \\
& \quad \quad a \leftarrow a + 1 \\
& \quad \text{if } a = 0 \\
& \quad \quad a \leftarrow b \\
& \quad \text{add mantissas and normalize}
\end{align*}
\]

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 \[
\frac{\text{dividend}}{\text{divisor}} \quad \begin{aligned} A &= 0.1 \text{xxxx} \\ B &= 0.1 \text{xxxx} \end{aligned} \quad \text{where } x = 0, 1
\]

(a) If \( A < B \) then after shift we have \( A = 1.\text{xxxx} \) and 1st quotient bit is 0.

(b) if \( A \geq B \), dividend alignment results in \( A = 0.01\text{xxxx} \) then after the left shift \( A > B \) and 1st quotient bit is 1.

www.samy007.blogfa.com
10-24

\[
\frac{\text{dividend}}{\text{divisor}} = \frac{\text{remainder}}{0.1xxx \times 2^{e_1} + 0.1yyyy \times 2^{e_2}} = \frac{1.0000 \times 2^{e_1}}{0.1yyyy \times 2^{e_2}}
\]

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

Fig. 10-10 after dividend alignment

\[
\begin{align*}
q &= 3 \\
a &= a + b + 1 \\
a &= a + b + 1ac \\
q &= a \\
a &= a - 9
\end{align*}
\]

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)

(2) Product: \(A \times Q = xxxxx \times x0000 \times 2^3\)

Product in AC: \(xxxxx \times 2^{3+5}\)

(5) Single precision normalized dividend: \(xxxxx \times 2^3\)

Dividend in AQ: \(A \times Q = xxxxx \times 00000 \times 2^{3-5}\)

10-27

Neglect Be and Ae from Fig. 10-14. Apply carry directly to E.
10-28 \[ \begin{array}{c}
-673 \\
356 \\
\hline
317
\end{array} \]

10's comp. of 356 = 644

\[ \begin{array}{c}
673 + \\
\hline
317
\end{array} \]

carry → 1

\[ C_4 = 1 \]
\[ C_3 = 1 \]
\[ C_2 = 0 \]
\[ C_1 = 1 \]

\[ M = 1 \]

\[ \text{Fig. 10-19} \]

\[ \text{inputs} \]

<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 0 1 0 0 1 1 9</td>
<td></td>
</tr>
<tr>
<td>1 0 0 0 1 1 0 0 0 8</td>
<td></td>
</tr>
<tr>
<td>2 0 0 1 0 1 1 1 1 7</td>
<td></td>
</tr>
<tr>
<td>3 0 0 1 1 0 1 1 0 6</td>
<td></td>
</tr>
<tr>
<td>4 0 1 0 0 0 1 0 1 5</td>
<td></td>
</tr>
<tr>
<td>5 0 1 0 1 0 1 0 0 4</td>
<td></td>
</tr>
<tr>
<td>6 0 1 1 0 0 0 1 1 3</td>
<td></td>
</tr>
<tr>
<td>7 0 1 1 1 0 0 1 0 2</td>
<td></td>
</tr>
<tr>
<td>8 1 0 0 0 0 0 0 0 1</td>
<td></td>
</tr>
<tr>
<td>9 1 0 0 1 1 0 0 0 0 0</td>
<td></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

\[ X_8 = B_8' B_4' B_2' \]
\[ X_4 = B_4' B_2 + B_4 B_2 \]
\[ X_2 = B_2 \]
\[ X_1 = B_1' \]
The excess-3 code is self-complementing code. Therefore, to get 9's complement we need to complement each bit.

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

\[
\begin{array}{c|c|c}
M & B_i & x'_i = B_i \oplus M \\
\hline
0 & 0 & 0 \\
0 & 1 & 1 \\
1 & 0 & 0 \\
1 & 1 & 1 \\
\end{array}
\]
Algorithm is similar to flow chart of Fig. 10-2

10-34 (a) $B = 410$

<table>
<thead>
<tr>
<th>Initial</th>
<th>$A_e$</th>
<th>$A$</th>
<th>$Q_1$</th>
<th>$S_C$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Q_L \neq 0$</td>
<td>0</td>
<td>000</td>
<td>152</td>
<td>3</td>
</tr>
<tr>
<td>$Q_L = 0$</td>
<td>0</td>
<td>470</td>
<td>151</td>
<td></td>
</tr>
<tr>
<td>$Q_L = 0, dshr$</td>
<td>0</td>
<td>940</td>
<td>150</td>
<td></td>
</tr>
<tr>
<td>$Q_L \neq 0$</td>
<td>1</td>
<td>034</td>
<td>013</td>
<td></td>
</tr>
<tr>
<td>$Q_L = 0$</td>
<td>1</td>
<td>504</td>
<td>012</td>
<td></td>
</tr>
<tr>
<td>$Q_L = 0, dshr$</td>
<td>2</td>
<td>974</td>
<td>011</td>
<td></td>
</tr>
<tr>
<td>$Q_L \neq 0$</td>
<td>0</td>
<td>444</td>
<td>010</td>
<td></td>
</tr>
<tr>
<td>$Q_L = 0, dshr$</td>
<td>0</td>
<td>244</td>
<td>401</td>
<td>1</td>
</tr>
<tr>
<td>$Q_L \neq 0$</td>
<td>0</td>
<td>714</td>
<td>400</td>
<td></td>
</tr>
<tr>
<td>$Q_L = 0, dshr$</td>
<td>0</td>
<td>071</td>
<td>440</td>
<td>0</td>
</tr>
</tbody>
</table>

Product

(b) \[
\begin{align*}
999 \\
\times 199 \\
\hline
89910 \\
+ 99900 \\
\hline
198801
\end{align*}
\]

- First partial product $A_e = 8$
- Second partial product $A_e = 9$
- Final product $A_e = 1$
\[
\frac{10-35}{32} = 52 + \frac{16}{32}
\]
\[B = 03\overline{2}\]
\[\overline{B} + 1 = 968\text{ (10's comp.)}\]

<table>
<thead>
<tr>
<th>Initial</th>
<th>(E)</th>
<th>(A)</th>
<th>(Q)</th>
<th>(SC)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>16</td>
<td>80</td>
<td>2</td>
</tr>
<tr>
<td>(dshl)</td>
<td>1</td>
<td>68</td>
<td>00</td>
<td></td>
</tr>
<tr>
<td>(add\ \overline{B} + 1)</td>
<td>1</td>
<td>36</td>
<td>01</td>
<td></td>
</tr>
<tr>
<td>(E = 1)</td>
<td>1</td>
<td>68</td>
<td>02</td>
<td></td>
</tr>
<tr>
<td>(add\ \overline{B} + 1)</td>
<td>1</td>
<td>40</td>
<td>02</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>68</td>
<td>03</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>08</td>
<td>04</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>68</td>
<td>05</td>
<td></td>
</tr>
<tr>
<td>(E = 0)</td>
<td>1</td>
<td>68</td>
<td>06</td>
<td></td>
</tr>
<tr>
<td>(add\ B)</td>
<td>0</td>
<td>32</td>
<td>07</td>
<td></td>
</tr>
<tr>
<td>restore remainder</td>
<td>1</td>
<td>08</td>
<td>05</td>
<td>1</td>
</tr>
</tbody>
</table>

(continued here)

<table>
<thead>
<tr>
<th>Initial</th>
<th>(E)</th>
<th>(A)</th>
<th>(Q)</th>
<th>(SC)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>80</td>
<td>50</td>
<td>1</td>
</tr>
<tr>
<td>(dshl)</td>
<td>-</td>
<td>9</td>
<td>68</td>
<td></td>
</tr>
<tr>
<td>(add\ \overline{B} + 1)</td>
<td>-</td>
<td>68</td>
<td>51</td>
<td></td>
</tr>
<tr>
<td>(E = 1)</td>
<td>-</td>
<td>48</td>
<td>52</td>
<td></td>
</tr>
<tr>
<td>(add\ B)</td>
<td>-</td>
<td>016</td>
<td>52</td>
<td></td>
</tr>
<tr>
<td>(E = 0)</td>
<td>-</td>
<td>984</td>
<td>52</td>
<td>0</td>
</tr>
</tbody>
</table>

10-36
(a) At the termination of multiplication we shift right the content of \(A\) to get zero in \(Ae\).
(b) 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 \(Ae = 0\).

10-37
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

\[
\begin{align*}
12 &= 00001100 \\
13 &= 00001101 \\
14 &= 00001110 \\
15 &= 00001111 \\
\end{align*}
\]

To 
CS [RSI RSO]

### 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 0001</td>
<td>0100 0010</td>
<td>0100 0011</td>
</tr>
<tr>
<td>3</td>
<td>0010 0000</td>
<td>0010 0001</td>
<td>0010 0010</td>
<td>0010 0011</td>
</tr>
<tr>
<td>4</td>
<td>0001 0000</td>
<td>0001 0001</td>
<td>0001 0010</td>
<td>0001 0011</td>
</tr>
<tr>
<td>5</td>
<td>0000 1000</td>
<td>0000 1001</td>
<td>0000 1010</td>
<td>0000 1011</td>
</tr>
<tr>
<td>6</td>
<td>0000 0100</td>
<td>0000 0101</td>
<td>0000 0110</td>
<td>0000 0111</td>
</tr>
</tbody>
</table>

### 11-3

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

### 11-5

See text discussion in Sec. 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.
(a) CPU data bus  "READ"  I/O
   I/O data bus STB
   Interface IBF
   I/O device

(b) I/O data bus
   Valid data
   STB
   IBF
   "READ"

(c) I/O device (source)
   Interface
   CPU

   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 nsec.

clock

Address

READ STROBE

WRITE STROBE

DATA

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

11-10
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 = 10101111
with even parity = 11010111
with start 2nd stop bits = 1110101110
(a) \( \frac{1200}{8} = 150 \) characters per second (cps)
(b) \( \frac{1200}{11} = 109 \) cps
(c) \( \frac{1200}{10} = 120 \) cps

\[ \frac{k \text{ bytes}}{(m-n) \text{ bytes/sec}} = \frac{R}{m-n} \text{ sec.} \]

(b) \( \frac{k}{n-m} \) sec. (c) No need for FIFO

**Initial**
- F = 0011
- Output ← R4

**After delete = 1**
- F = 0010
- R4 ← R3

**After delete = 0**
- F = 0001
- R1 ← Input

**After insert = 1**
- F = 1001
- R2 ← R1

**Insert goes to 0**
- F = 0101
- R3 ← R2

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

<table>
<thead>
<tr>
<th>Input ready</th>
<th>output ready</th>
<th>F₁-F₄</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0000</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1111</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0011</td>
</tr>
</tbody>
</table>

**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.
CPU Program flow chart:

1. Read data from memory into a CPU Register
2. Read status register from interface and check output flag bit
3. Write data into interface data register
4. Continue
5. Transferred all data?

See text Section 11-4.

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:
   - Device 1
     - PI=0; P0=0; RF=0
   - Device 2
     - PI=0; P0=0; RF=1
2. Before CPU responds with acknowledge, device 1 sends interrupt request:
   - Device 1
     - PI=0; P0=0; RF=1
   - Device 2
     - PI=0; P0=0; RF=1
3. After CPU sends an acknowledge, device 1 has priority:
   - Device 1
     - VAD enable = 1
   - Device 2
     - VAD enable = 0
### 11-22 Table 11-2

<table>
<thead>
<tr>
<th>$I_0$</th>
<th>$I_1$</th>
<th>$I_2$</th>
<th>$I_3$</th>
<th>$XY$</th>
<th>$I_{ST}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i$</td>
<td>$X$</td>
<td>$X$</td>
<td>$X$</td>
<td>$00$</td>
<td>$I$</td>
</tr>
<tr>
<td>$0$</td>
<td>$1$</td>
<td>$X$</td>
<td>$X$</td>
<td>$01$</td>
<td>$I$</td>
</tr>
<tr>
<td>$0$</td>
<td>$0$</td>
<td>$1$</td>
<td>$X$</td>
<td>$10$</td>
<td>$I$</td>
</tr>
<tr>
<td>$0$</td>
<td>$0$</td>
<td>$0$</td>
<td>$1$</td>
<td>$11$</td>
<td>$I$</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 8x3 decoder.

### 11-24

<table>
<thead>
<tr>
<th>$I_0$</th>
<th>$I_1$</th>
<th>$I_2$</th>
<th>$I_3$</th>
<th>$I_4$</th>
<th>$I_5$</th>
<th>$I_6$</th>
<th>$I_7$</th>
<th>$XY$</th>
<th>$I_{ST}$</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>$X$</td>
<td>$000$</td>
<td>$I$</td>
</tr>
<tr>
<td>$0$</td>
<td>$1$</td>
<td>$X$</td>
<td>$X$</td>
<td>$X$</td>
<td>$X$</td>
<td>$X$</td>
<td>$X$</td>
<td>$001$</td>
<td>$I$</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>$X$</td>
<td>$010$</td>
<td>$I$</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>$X$</td>
<td>$011$</td>
<td>$I$</td>
</tr>
<tr>
<td>$0$</td>
<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>$I$</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>$X$</td>
<td>$101$</td>
<td>$I$</td>
</tr>
<tr>
<td>$0$</td>
<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>$I$</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>$1$</td>
<td>$111$</td>
<td>$I$</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>$0$</td>
<td>$XXX$</td>
<td>$0$</td>
</tr>
</tbody>
</table>

### Binary

<table>
<thead>
<tr>
<th>(E)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10100000</td>
</tr>
<tr>
<td>10100100</td>
</tr>
<tr>
<td>10101000</td>
</tr>
<tr>
<td>10101100</td>
</tr>
<tr>
<td>10110000</td>
</tr>
<tr>
<td>10110100</td>
</tr>
<tr>
<td>10111000</td>
</tr>
<tr>
<td>10111100</td>
</tr>
</tbody>
</table>

### Hexadecimal

| A0 |
| A4 |
| A8 |
| AC |
| BD |
| BF |
| 8F |
| BC |

### 11-25

$76 = (01001100)_2$

Replace the six 0's by $010011$ in $XY$

### 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, and 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

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 00000101. 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 | DLE STX DLE DLE ETX DLE DLE ETX DLE ETX

32-bit text = 00010000 10000011 00010000 10000011

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

11-36 | Information to be sent (1023):

After zero insertion, information transmitted: 011111011110

Information received after 0's deletion: 011111111111
12-1 (a) \( \frac{2048}{128} = 16 \) chips
(b) \( 2048 = 2^{11} \) lines to address 2048 bytes
\( \frac{128}{2^7} = 2^4 \) lines to address each chip
(c) 4x16 decoder

12-2
(a) 8 chips are needed with address lines connected in parallel.
(b) \( 16 \times 8 = 128 \) chips. Use 14 address lines (\( 16 \times 8 = 2^{14} \))
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>16151413 121109 8765 4321</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM</td>
<td>0000-0FFF</td>
<td>( \frac{5x32}{\text{decoder}} ) xxx xxxxx</td>
</tr>
<tr>
<td>ROM</td>
<td>1000-1FFF</td>
<td>( \frac{3x8}{\text{decoder}} ) xxx xxxxx</td>
</tr>
</tbody>
</table>

12-5
RAM \( \frac{2048}{256} = 8 \) chips ; \( 2048 = 2^{11} \); \( \frac{256}{2^8} \)
ROM \( \frac{4096}{1024} = 4 \) chips ; \( \frac{4096}{2^{12}} \); \( \frac{1024}{2^{10}} \)
Interface \( 4 \times 4 = 16 \) registers ; \( 16 = 2^4 \)

<table>
<thead>
<tr>
<th>Component</th>
<th>Address</th>
<th>16151413 121109 8765 4321</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM</td>
<td>0000-07FF</td>
<td>( \frac{3x8}{\text{decoder}} ) xxx xxxxx</td>
</tr>
<tr>
<td>ROM</td>
<td>4000-4FFF</td>
<td>( \frac{2x8}{\text{decoder}} ) xxx xxxxx</td>
</tr>
<tr>
<td>Interface</td>
<td>8000-80FF</td>
<td>0100 0000 0000 0000 xxx</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 = $E + \text{time for half revolution} +$ time to read a sector.

$$T_a = T_s + \frac{1}{2R} + \frac{N_s}{N_e} \cdot \frac{1}{R}$$

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

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

From other bits in the same word

12-10 A match occurs if $T_i = 1$

match = $M_i T_i$

$M_i \rightarrow T_i \rightarrow \text{match if 1}$

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 \(2^d = m\) (\(m = \text{No. of words in memory}\)). For each count, the \(M_i\) bit is checked and if 1, the corresponding read signal for word \(i\) is activated.

Let \(x_j = A_{ij}F_{i1}' + A_i'F_{i1}\) (argument bit = memory word bit)

Output indicator \(G_i = 1\) if:
- \(A_1F_{i1} = 1\) and \(K_1 = 1\) (First bit in \(A = 1\) while \(F_{i1} = 0\))
- or if \(x_1A_2F_{i2}' = 1\) and \(K_2 = 1\) (First pair of bits are equal and second bit in \(A = 1\) while \(F_{i2} = 0\))

\[G_i = (A_1F_{i1} + K_1')(x_1A_2F_{i2}' + K_2')(x_1x_2A_3F_{i3} + K_3') \cdots (x_1x_2 \cdots x_{n-1}A_nF_{in} + K_n')\]
12-15

Given that the size of the cache is 128K, which is equal to $2^{17}$, the index address has 10 bits to accommodate $\frac{2^{18}}{2} = 1024$ words of cache.

(a) The index table is structured as follows:

<table>
<thead>
<tr>
<th>TAG</th>
<th>INDEX</th>
</tr>
</thead>
</table>

- Block $\rightarrow$ Word
- 8 bits $\rightarrow$ 2 bits

(b) The cache entry is structured as follows:

- Tag 1 $\leftarrow$ Data 1
- Tag 2 $\leftarrow$ Data 2

- Total cache size is $1024 \times 2 (7+32) = 1024 \times 78$

12-16

(a) Cache access

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

(b) Cache + Memory access

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

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

12-17

Sequence: A B C D B E D A C E C E

LRU

Count value = 3 2 1 0

Initial words = A B C D
B is a hit A C D B
E is a miss C D B E
D is a hit C B E D
A is a miss B E D A
C is a miss E D A C
E is a hit D A C E
C is a hit D A E C
E is a hit D A C E
12-18

\[ 64K \times 16 : 16 \text{ bit address, 16-bit data.} \]

(a)

\[
\begin{array}{c|c|c|c}
\text{TAG} & \text{BLOCK} & \text{WRD} \\
\hline
6 & 8 & 2 \\
\end{array}
\]

\[ \text{INDEX = 10 bit cache address.} \]

(b)

\[ \begin{array}{c|c|c}
V & \text{TAG} & \text{DATA} \\
\hline
16 & & \text{bits in each word of cache} \\
\end{array} \]

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

12-19

(a) Address space = 24 bits \[2^{24} = 16 \text{ M words} \]

(b) Memory space = 16 bits \[2^{16} = 64 \text{ K words} \]

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

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

4 2 0 1 2 6 1 4 0 1 0 2 3 5 7

<table>
<thead>
<tr>
<th>Page reference</th>
<th>(a) Pages in main memory</th>
<th>(b) Pages in memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial</td>
<td>0124 4201</td>
<td>0124 4201</td>
</tr>
<tr>
<td>2</td>
<td>0124 4201</td>
<td>0124 4201</td>
</tr>
<tr>
<td>6</td>
<td>0126 2016</td>
<td>0126 2016</td>
</tr>
<tr>
<td>1</td>
<td>0146 0164</td>
<td>0146 0164</td>
</tr>
<tr>
<td>4</td>
<td>0146 0164</td>
<td>0146 0164</td>
</tr>
<tr>
<td>0</td>
<td>0146 0164</td>
<td>0146 0164</td>
</tr>
<tr>
<td>1</td>
<td>0146 0164</td>
<td>0146 0164</td>
</tr>
<tr>
<td>0</td>
<td>0124 4201</td>
<td>0124 4201</td>
</tr>
<tr>
<td>2</td>
<td>0124 4201</td>
<td>0124 4201</td>
</tr>
<tr>
<td>3</td>
<td>2345 6423</td>
<td>0123 1023</td>
</tr>
<tr>
<td>5</td>
<td>2345 6423</td>
<td>0123 1023</td>
</tr>
<tr>
<td>7</td>
<td>2357 2357</td>
<td>2357 2357</td>
</tr>
</tbody>
</table>
12-22

$600AF$ and $F00AF$

12-23

Logical address: 

<table>
<thead>
<tr>
<th>7 bits</th>
<th>5 bits</th>
<th>12 bits</th>
<th>-24 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Segment</td>
<td>Page</td>
<td>Word</td>
<td></td>
</tr>
</tbody>
</table>

Physical address:

<table>
<thead>
<tr>
<th>12 bits</th>
<th>12 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block</td>
<td>Word</td>
</tr>
</tbody>
</table>

12-24

Segment $36 = (0100100)_2$ (7-bit binary)

Page $15 = (01111)_2$ (5-bit binary)

Word $2000 = (011111010000)_2$ (12-bit binary)

Logical address $= C150100 01111 011111010000$

(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 requires 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.

13-3

\[ \pi \times m \] switches

13-4

log \_2 n stages with \( \frac{n}{2} \) switches in each stage.

13-5

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

13-6

[Diagram of a network with switches and inputs/outputs labeled 0 through 3, and outputs 00, 01, 10, 11 shown.]
Arbitration switch:

A connected to output
B connected to output

Distribution switch:

Input connected to A
Input connected to B

(b)

I = Interchange 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

If PI goes to 0, while using the bus, the device must terminate its bus operation and then reset both flip-flops.

Bus busy line (busy if equal to 0)
13-10

Encoder input: 10110
Encoder output: 01 (It has highest priority)
Decoder input: 01
Decoder output: 0100 Arbitrator 2(i) is acknowledged

13-11

As explained in the text, connect output P0 from
arbitrator 4 into input P5 of arbitrator 1. Once the
line is disabled, the arbitrator 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.

11-14

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 coherency problem can be resolved by either updating or invalidating all other cache values of the written information.