COMPUTER ARCHITECTURE
CMPT 328

FINAL EXAM

STUDENT: AUDIT COPY

▸ **QUESTION 1**

Consider the two possible methods for performing longhand conversion from base k to base b.

Method 1: Work from the left repeatedly dividing by the largest power of b less than or equal to the number to convert.

Method 2: Work from the right repeatedly dividing by b.

Using the method indicated, perform longhand conversion from decimal to hexadecimal on the decimal numbers below:

Method 1: 8922

Method 2: 5652

Show all your work.

Estimated time to complete this question is 5 minutes.

SOLUTION

Method 1: 8922

$$\begin{aligned} \frac{8922}{4096} &= 2\ r\ 730 \\ \frac{730}{256} &= 2\ r\ 218 \\ \frac{218}{16} &= 13\ r\ 10 \\ \frac{10}{1} &= 10\ r\ 0 \end{aligned} \tag{1}$$

0x22DA

Method 2: 5652

$$\begin{aligned} \frac{5652}{16} &= 353\ r\ 4 \\ \frac{353}{16} &= 22\ r\ 1 \\ \frac{22}{16} &= 1\ r\ 6 \\ \frac{1}{16} &= 0\ r\ 1 \end{aligned} \tag{2}$$

0x1614

▸ **QUESTION 2**

Consider the two possible methods for performing longhand conversion from base k to base b.

Method 1: Work from the left repeatedly dividing by the largest power of b less than or equal to the number to convert.

Method 2: Work from the right repeatedly dividing by b.

Using the method indicated, perform longhand conversion from decimal to hexadecimal on the decimal numbers below:

Method 1: 7783

Method 2: 5616

Show all your work.

Estimated time to complete this question is 5 minutes.

SOLUTION

Method 1: 7783

$$\begin{aligned} \frac{7783}{4096} &= 1 \; r \; 3687 \\ \frac{3687}{256} &= 14 \; r \; 103 \\ \frac{103}{16} &= 6 \; r \; 7 \\ \frac{7}{1} &= 7 \; r \; 0 \end{aligned} \tag{3}$$

0x1E67

Method 2: 5616

$$\begin{aligned} \frac{5616}{16} &= 351 \; r \; 0 \\ \frac{351}{16} &= 21 \; r \; 15 \\ \frac{21}{16} &= 1 \; r \; 5 \\ \frac{1}{16} &= 0 \; r \; 1 \end{aligned} \tag{4}$$

0x15F0

▸ QUESTION 3

Consider the two possible methods for performing longhand conversion from base k to base b.

Method 1: Work from the left repeatedly dividing by the largest power of b less than or equal to the number to convert.

Method 2: Work from the right repeatedly dividing by b.

Using the method indicated, perform longhand conversion from decimal to hexadecimal on the decimal numbers below:

Method 1: 5498

Method 2: 6834

Show all your work.

Estimated time to complete this question is 5 minutes.

SOLUTION

Method 1: 5498

$$\begin{aligned}
\frac{5498}{4096} &= 1\ r\ 1402 \\
\frac{1402}{256} &= 5\ r\ 122 \\
\frac{122}{16} &= 7\ r\ 10 \\
\frac{10}{1} &= 10\ r\ 0
\end{aligned} \tag{5}$$

0x157A

Method 2: 6834

$$\begin{aligned}
\frac{6834}{16} &= 427\ r\ 2 \\
\frac{427}{16} &= 26\ r\ 11 \\
\frac{26}{16} &= 1\ r\ 10 \\
\frac{1}{16} &= 0\ r\ 1
\end{aligned} \tag{6}$$

0x1AB2

▸ **QUESTION 4**

Consider the two possible methods for performing longhand conversion from base k to base b.

Method 1: Work from the left repeatedly dividing by the largest power of b less than or equal to the number to convert.

Method 2: Work from the right repeatedly dividing by b.

Using the method indicated, perform longhand conversion from decimal to hexadecimal on the decimal numbers below:

Method 1: 6566

Method 2: 8852

Show all your work.

Estimated time to complete this question is 5 minutes.

SOLUTION

Method 1: 6566

$$
\begin{aligned}
\frac{6566}{4096} &= 1\ r\ 2470 \\
\frac{2470}{256} &= 9\ r\ 166 \\
\frac{166}{16} &= 10\ r\ 6 \\
\frac{6}{1} &= 6\ r\ 0
\end{aligned}
\tag{7}
$$

0x19A6

Method 2: 8852

$$
\begin{aligned}
\frac{8852}{16} &= 553\ r\ 4 \\
\frac{553}{16} &= 34\ r\ 9 \\
\frac{34}{16} &= 2\ r\ 2 \\
\frac{2}{16} &= 0\ r\ 2
\end{aligned}
\tag{8}
$$

0x2294

▸ QUESTION 5

Consider the following logic circuit:

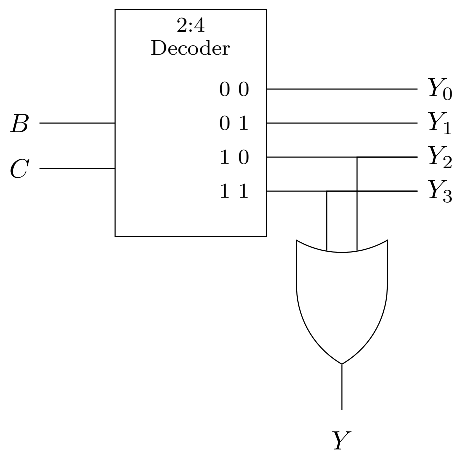

Figure 1: Logic circuit implemented using decoder.

(a) Draw a complete truth table with minterms and minterm names.

(b) Draw a complete truth table with maxterms and maxterm names.

(c) Write a Boolean equation in sum-of-products canonical form. Simplify the Boolean equation.

(d) Write a Boolean equation in product-of-sums canonical form. Simplify the Boolean equation.

(e) Which theorem of Boolean algebra does this circuit illustrate?

Estimated time to complete this question is 15 minutes.

SOLUTION

(a) Truth table for circuit in Figure 1 with minterms and minterm names.

B	C	Y	minterm	names
0	0	0	$\overline{B}\,\overline{C}$	m_0
0	1	0	$\overline{B}\,C$	m_1
1	0	1	$B\,\overline{C}$	m_2
1	1	1	$B\,C$	m_3

Figure 2: Truth table for decoder logic circuit in Figure 1—sum-of-products.

(b) Truth table for circuit in Figure 1 with maxterms and maxterm names.

B	C	Y	minterm	names
0	0	0	$B+C$	M_0
0	1	0	$B+\overline{C}$	M_1
1	0	1	$\overline{B}+C$	M_2
1	1	1	$\overline{B}+\overline{C}$	M_3

Figure 3: Truth table for multiplexer logic circuit in Figure 1—product-of-sums.

(c) Sum-of-products Boolean equation for truth table Figure 2.

Boolean equation in sum-of-products canonical form and simplification. The Boolean equation in sum-of-products canonical form is the sum of the minterms for which Y is 1.

$$\begin{aligned} Y &= m_2 + m_3 \\ &= B\,\overline{C} + B\,C \\ &= B \bullet (\overline{C} + C) \\ &= B \bullet 1 \\ Y &= B \end{aligned} \tag{9}$$

(d) Product-of-sums Boolean equation for truth table Figure 3.

Boolean equation in product-of-sums canonical form and simplification. The Boolean equation in product-of-sums canonical form is the product of the maxterms for which Y is 0.

$$\begin{aligned}
Y &= M_0 \bullet M_1 \\
&= (B + C) \bullet (B + \overline{C}) \\
&= B\,B + B\,\overline{C} + B\,C + C\,\overline{C} \\
&= B + B \bullet (\overline{C} + C) + 0 \\
&= B + B \bullet (\overline{C} + C) \\
&= B + B \bullet 1 \\
&= B + B \\
Y &= B
\end{aligned} \tag{10}$$

(e) The Boolean function illustrates theorem of Boolean algebra T10 combining.

▸ **QUESTION 6**
Consider the following logic circuit:

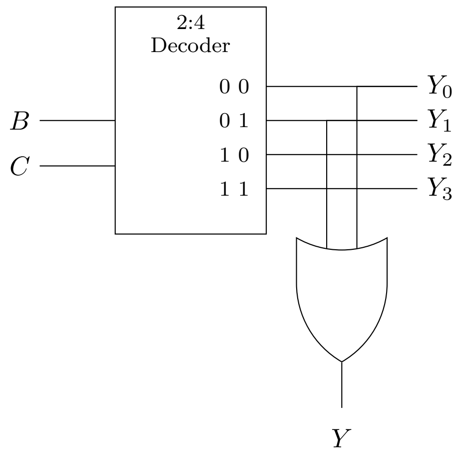

Figure 4: Logic circuit implemented using decoder.

(a) Draw a complete truth table with minterms and minterm names.

(b) Draw a complete truth table with maxterms and maxterm names.

(c) Write a Boolean equation in sum-of-products canonical form. Simplify the Boolean equation.

(d) Write a Boolean equation in product-of-sums canonical form. Simplify the Boolean equation.

(e) Which theorem of Boolean algebra does this circuit illustrate?

Estimated time to complete this question is 15 minutes.

SOLUTION

(a) Truth table for circuit in Figure 4 with minterms and minterm names.

B	C	Y	minterm	names
0	0	1	$\overline{B}\,\overline{C}$	m_0
0	1	1	$\overline{B}\,C$	m_1
1	0	0	$B\,\overline{C}$	m_2
1	1	0	$B\,C$	m_3

Figure 5: Truth table for decoder logic circuit in Figure 4—sum-of-products.

(b) Truth table for circuit in Figure 4 with maxterms and maxterm names.

B	C	Y	minterm	names
0	0	1	$B+C$	M_0
0	1	1	$B+\overline{C}$	M_1
1	0	0	$\overline{B}+C$	M_2
1	1	0	$\overline{B}+\overline{C}$	M_3

Figure 6: Truth table for multiplexer logic circuit in Figure 4—product-of-sums.

(c) Sum-of-products Boolean equation for truth table Figure 5.

Boolean equation in sum-of-products canonical form and simplification. The Boolean equation in sum-of-products canonical form is the sum of the minterms for which Y is 1.

$$\begin{aligned} Y &= m_0 + m_1 \\ &= \overline{B}\,\overline{C} + \overline{B}\,C \\ &= \overline{B} \bullet (\overline{C} + C) \\ &= \overline{B} \bullet 1 \\ Y &= \overline{B} \end{aligned} \tag{11}$$

(d) Product-of-sums Boolean equation for truth table Figure 6.

Boolean equation in product-of-sums canonical form and simplification. The Boolean equation in product-of-sums canonical form is the product of the maxterms for which Y is 0.

$$\begin{aligned} Y &= M_2 \bullet M_3 \\ &= (\overline{B} + C) \bullet (\overline{B} + \overline{C}) \\ &= \overline{B}\,\overline{B} + \overline{B}\,\overline{C} + B\,\overline{C} + \overline{C}\,C \\ &= \overline{B} + \overline{B} \bullet (\overline{C} + C) + 0 \\ &= \overline{B} + \overline{B} \bullet 1 \\ &= \overline{B} + \overline{B} \\ Y &= \overline{B} \end{aligned} \tag{12}$$

(e) The Boolean function illustrates theorem of Boolean algebra T10 combining.

▸ QUESTION 7

Consider the following implementation of a full binary adder:

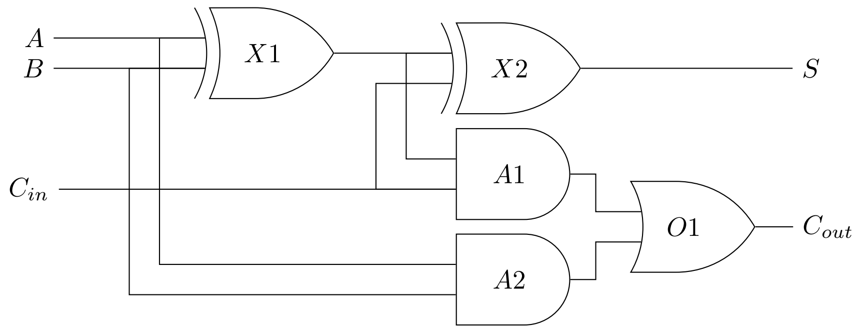

Figure 7: Binary full adder, 1–bit.

(a) Is the full adder circuit shown in Figure 7 an example of combinational logic or sequential logic? Why?

(b) Explain how the full adder works. Consider the discrete cases where:

I A and B do not generate a carry and C_{in} is 0.

II A and B do not generate a carry and C_{in} is 1.

III A and B do generate a carry and C_{in} is 0.

IV A and B do generate a carry and C_{in} is 1.

(c) Draw a complete truth table. In your truth table show the values of each literal: A, B, C_{in}, S, and C_{out}.

Estimated time to complete this question is 15 minutes.

SOLUTION

(a) The full adder implements combinational logic. The outputs depend only on the current inputs.

(b) Operation

Case 1: A and B generate a carry only when both are TRUE. $A2$ therefore outputs FALSE. Since C_{in} is FALSE, $A1$ outputs FALSE. Because $A1$ and $A2$ output FALSE, $O1$ also outputs FALSE on C_{out}. $X1$ will output TRUE if either A or B are TRUE and FALSE if both are FALSE. $X2$ has one FALSE input from C_{in} and a TRUE or FALSE input from $X1$ output depending on A or B. $X2$ outputs TRUE on S if one of A or B is TRUE—recall the stipulation of this case that A and B do not generate a carry, they are not both TRUE. $X2$ thus outputs the sum of A and B.

Case 2: A and B generate a carry only when both are TRUE. $A2$ therefore outputs FALSE. Since C_{in} is TRUE, $A1$ has one TRUE input. $X1$ outputs TRUE when A or B are TRUE. In this case, $X2$ and $A1$ have two TRUE inputs. $X2$ outputs FALSE on S and $A1$ outputs TRUE. So $A1$ outputs TRUE and $A2$ outputs FALSE, $O1$ outputs TRUE on C_{out}.

If A and B are both FALSE, $X1$ outputs FALSE. $X2$ receives one FALSE input and one TRUE input so it's output is TRUE on S. $A1$ receives one FALSE input so it's output is FALSE. $A2$ receives two FALSE inputs so it's output is FALSE. $O1$ receives two false inputs and outputs FALSE on C_{out}.

Case 3: A and B are both TRUE. $X1$ outputs FALSE. $A2$ receives two TRUE inputs and outputs TRUE. $A1$ receives two FALSE inputs and outputs FALSE. $O1$ receives two FALSE inputs and outputs FALSE on C_{out}. $X2$ receives two FALSE inputs and outputs FALSE on S.

Case 4: A and B are both TRUE. $X1$ outputs FALSE. $A1$ receives one TRUE input and one FALSE input and so outputs FALSE. $A2$ receives two TRUE inputs and outputs TRUE. $X2$ receives one TRUE input and one FALSE input so it outputs TRUE on S. $O1$ receives one TRUE input and one FALSE input so it outputs TRUE on output C_{out}.

(c) Truth table for full adder implementation shown in Figure 7.

A	B	C_{in}	S	C_{out}
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

Figure 8: Truth table for full adder—Figure 7

▸ **QUESTION 8**

Identify the high-level language construct implemented by the following MIPS assembly language:

```
    bne $s3, $s4, label1
    add $s0, $s1, $s2
    j label2
label1:
    sub $s0, $s1, $s3
label2:
```

Estimated time to complete this question is 5 minutes.

SOLUTION

The MIPS assembly code implements an if/else statement.

▸ **QUESTION 9**

Identify the high-level language construct implemented by the following MIPS assembly language:

```
    addi $t0, $0, 10
label1:
    beq $t0, $0, label2
    addi $t0, $t0, -1
    # some code block
    j label1
label2:
```

Estimated time to complete this question is 5 minutes.

SOLUTION

The MIPS assembly code implements a while loop.

▸ **QUESTION 10**

Identify the high-level language construct implemented by the following MIPS assembly language:

```
    addi $s0, $0, 0
    addi $t0, $0, 10
label1:
    beq $s0, $t0, label2
    # some code block
    addi $s0, $s0, 1
    j label1
label2:
```

Estimated time to complete this question is 5 minutes.

SOLUTION

The MIPS assembly code implements a for loop.

▸ **QUESTION 11**

Suppose MIPS register `$s0` holds the memory location of the following little-endian 32–bit word:

A2	8F	7F	F2

(a) What is the value in MIPS register `$s1` after `lbu $s1, 2($s0)` is executed?

(b) What is the value in MIPS register `$s1` after `lb $s1, 2($s0)` is executed?

(c) What is the value in MIPS register `$s1` after `lb $s1, 1($s0)` is executed?

Estimated time to complete this question is 5 minutes.

SOLUTION

(a) The byte at offset 2 is zero-extended to 32 bits and loaded into register `$s1`. After `lbu $s1, 2($s0)`, `$s1` contains:

00	00	00	8F

(b) The byte at offset 2 is sign-extended to 32 bits and loaded into register `$s1`. Since this byte is a negative two's complement number, the sign bit is 1. After `lb $s1, 2($s0)`, `$s1` contains:

FF	FF	FF	8F

(c) The byte at offset 1 is sign-extended to 32 bits and loaded into register `$s1`. Since this byte is a positive two's complement number, the sign bit is 0. After `lb $s1, 2($s0)`, `$s1` contains:

00	00	00	7F

▸ **QUESTION 12**

Suppose MIPS register `$s0` holds the memory location of the following little-endian 32–bit word:

19	A1	3F	27

(a) What is the value in MIPS register `$s1` after `lbu $s1, 2($s0)` is executed?

(b) What is the value in MIPS register `$s1` after `lb $s1, 2($s0)` is executed?

(c) What is the value in MIPS register `$s1` after `lb $s1, 1($s0)` is executed?

Estimated time to complete this question is 5 minutes.

SOLUTION

(a) The byte at offset 2 is zero-extended to 32 bits and loaded into register `$s1`. After `lbu $s1, 2($s0)`, `$s1` contains:

00	00	00	A1

(b) The byte at offset 2 is sign-extended to 32 bits and loaded into register `$s1`. Since this byte is a negative two's complement number, the sign bit is 1. After `lb $s1, 2($s0)`, `$s1` contains:

FF	FF	FF	A1

(c) The byte at offset 1 is sign-extended to 32 bits and loaded into register `$s1`. Since this byte is a positive two's complement number, the sign bit is 0. After `lb $s1, 2($s0)`, `$s1` contains:

00	00	00	3F

▸ **QUESTION 13**

Suppose MIPS register `$s0` holds the memory location of the following little-endian 32–bit word:

E1	C9	2D	82

(a) What is the value in MIPS register `$s1` after `lbu $s1, 2($s0)` is executed?

(b) What is the value in MIPS register `$s1` after `lb $s1, 2($s0)` is executed?

(c) What is the value in MIPS register `$s1` after `lb $s1, 1($s0)` is executed?

Estimated time to complete this question is 5 minutes.

SOLUTION

(a) The byte at offset 2 is zero-extended to 32 bits and loaded into register `$s1`. After `lbu $s1, 2($s0)`, `$s1` contains:

00	00	00	C9

(b) The byte at offset 2 is sign-extended to 32 bits and loaded into register `$s1`. Since this byte is a negative two's complement number, the sign bit is 1. After `lb $s1, 2($s0)`, `$s1` contains:

FF	FF	FF	C9

(c) The byte at offset 1 is sign-extended to 32 bits and loaded into register `$s1`. Since this byte is a positive two's complement number, the sign bit is 0. After `lb $s1, 2($s0)`, `$s1` contains:

00	00	00	2D

▸ QUESTION 14

Suppose MIPS register `$s0` holds the following 32–bit word:

1111	0110	0000	0000	0010	1111	0011	1000

(a) What is the value in MIPS register `$t1` after `sll $t1, $s0, 4` is executed?

(b) What is the value in MIPS register `$t1` after `srl $t1, $s0, 4` is executed?

(c) What is the value in MIPS register `$t1` after `sra $t1, $s0, 4` is executed?

Estimated time to complete this question is 10 minutes.

SOLUTION

(*a*) `sll` logically shifts the bits left and fills the new least significant bits with 0.

0110	0000	0000	0010	1111	0011	1000	0000

(*b*) `srl` logically shifts the bits right and fills the new most significant bits with 0.

0000	1111	0110	0000	0000	0010	1111	0011

(*c*) `srl` arithmetically shifts the bits right and fills the new most significant bits the sign bit.

1111	1111	0110	0000	0000	0010	1111	0011

▸ QUESTION 15

Suppose MIPS register `$s0` holds the following 32–bit word:

1000	1010	0000	0000	0010	1111	0111	1010

(a) What is the value in MIPS register `$t1` after `sll $t1, $s0, 4` is executed?

(b) What is the value in MIPS register `$t1` after `srl $t1, $s0, 4` is executed?

(c) What is the value in MIPS register `$t1` after `sra $t1, $s0, 4` is executed?

Estimated time to complete this question is 10 minutes.

SOLUTION

(*a*) `sll` logically shifts the bits left and fills the new least significant bits with 0.

1010	0000	0000	0010	1111	0111	1010	0000

(b) `srl` logically shifts the bits right and fills the new most significant bits with 0.

0000	1000	1010	0000	0000	0010	1111	0111

(c) `srl` arithmetically shifts the bits right and fills the new most significant bits the sign bit.

1111	1000	1010	0000	0000	0010	1111	0111

▸ **QUESTION 16**

Suppose MIPS register `$s0` holds the following 32–bit word:

1011	0110	0000	0000	0010	1111	1001	0011

(a) What is the value in MIPS register `$t1` after `sll $t1, $s0, 4` is executed?

(b) What is the value in MIPS register `$t1` after `srl $t1, $s0, 4` is executed?

(c) What is the value in MIPS register `$t1` after `sra $t1, $s0, 4` is executed?

Estimated time to complete this question is 10 minutes.

SOLUTION

(a) `sll` logically shifts the bits left and fills the new least significant bits with 0.

0110	0000	0000	0010	1111	1001	0011	0000

(b) `srl` logically shifts the bits right and fills the new most significant bits with 0.

0000	1011	0110	0000	0000	0010	1111	1001

(c) `srl` arithmetically shifts the bits right and fills the new most significant bits the sign bit.

1111	1011	0110	0000	0000	0010	1111	1001

▸ **QUESTION 17**

Translate the following MIPS assembly language instructions to machine language.

(a) `add $s0, $t0, $t1`

(b) `ori $s0, $s1, 0x80`

Represent the machine language instructions in both hexadecimal and octal.

Estimated time to complete this question is 10 minutes.

SOLUTION

(a) `add $s0, $t0, $t1`
From Table 6.1 MIPS register set in *Digital Design and Computer Architecture*, registers `$s0`, `$t0`, and `$t1` are numbered 16, 8, and 9 respectively. From *Appendix B funct* is 100000. Since this is an R-type instruction, *opcode* is 000000.
The instruction syntax is

`add rd, rs, rt`

opcode	rs	rt	rd	shamt	funct
000000	01000	01001	10000	00000	100000

We can now generically represent the machine language for conversion to hexadecimal and octal:

0000.0001.0000.1001.1000.0000.0010.0000

00.000.001.000.010.011.000.000.000.100.000

By inspection, the result is 0x01098020 and 00102300040.

(*b*) `ori $s0, $s1, 0x80`
From Table 6.1 MIPS register set in *Digital Design and Computer Architecture*, registers `$s0`, and `$s1` are numbered 16, and 17. From *Appendix B*, *opcode* is 001101.
The instruction syntax is

`ori rt, rs, imm`

opcode	rs	rt	imm
001101	10001	10000	0000.0000.1000.0000

We can now generically represent the machine language for conversion to hexadecimal and octal:

0011.0110.0011.0000.0000.0000.1000.0000

00.110.110.001.100.001.000.000.010.000.000

By inspection, the result is 0x36300080 and 06614100200.

▸ **QUESTION 18**
Translate the following MIPS assembly language instructions to machine language.

(a) `sub $s0, $t0, $t1`

(b) `addi $s0, $s1, 0x80`

Represent the machine language instructions in both hexadecimal and octal.

Estimated time to complete this question is 10 minutes.

SOLUTION

(*a*) `sub $s0, $t0, $t1`
From Table 6.1 MIPS register set in *Digital Design and Computer Architecture*, registers `$s0`, `$t0`, and `$t1` are numbered 16, 8, and 9 respectively. From *Appendix B funct* is 100010. Since this is an R-type instruction, *opcode* is 000000.
The instruction syntax is

`sub rd, rs, rt`

opcode	rs	rt	rd	shamt	funct
000000	01000	01001	10000	00000	100010

We can now generically represent the machine language for conversion to hexadecimal and octal:

0000.0001.0000.1001.1000.0000.0010.0010

00.000.001.000.010.011.000.000.000.100.010

By inspection, the result is 0x01098022 and 00102300042.

(*b*) `addi $s0, $s1, 0x80`
From Table 6.1 MIPS register set in *Digital Design and Computer Architecture*, registers `$s0`, and `$s1` are numbered 16, and 17. From *Appendix B*, *opcode* is 001000.
The instruction syntax is

`addi rt, rs, imm`

opcode	rs	rt	imm
001000	10001	10000	0000.0000.1000.0000

We can now generically represent the machine language for conversion to hexadecimal and octal:

0010.0010.0011.0000.0000.0000.1000.0000

00.100.010.001.100.000.000.000.010.000.000

By inspection, the result is 0x22300080 and 04214000200.

QUESTION 19
Translate the following MIPS assembly language instructions to machine language.

(a) `and $s0, $t0, $t1`

(b) `lw $s0, 0x80($s1)`

Represent the machine language instructions in both hexadecimal and octal.

Estimated time to complete this question is 10 minutes.

SOLUTION

(*a*) `and $s0, $t0, $t1`
From Table 6.1 MIPS register set in *Digital Design and Computer Architecture*, registers `$s0`, `$t0`, and `$t1` are numbered 16, 8, and 9 respectively. From *Appendix B funct* is 100100. Since this is an R-type instruction, *opcode* is 000000.
The instruction syntax is

`and rd, rs, rt`

opcode	rs	rt	rd	shamt	funct
000000	01000	01001	10000	00000	100100

We can now generically represent the machine language for conversion to hexadecimal and octal:

0000.0001.0000.1001.1000.0000.0010.0100

00.000.001.000.010.011.000.000.000.100.100

By inspection, the result is 0x01098024 and 00102300044.

(*b*) `lw $s0, 0x80($s1)`
From Table 6.1 MIPS register set in *Digital Design and Computer Architecture*, registers `$s0`, and `$s1` are numbered 16, and 17. From *Appendix B*, *opcode* is 100011.
The instruction syntax is

`lw rt, imm(rs)`

opcode	rs	rt	imm
100011	10001	10000	0000.0000.1000.0000

We can now generically represent the machine language for conversion to hexadecimal and octal:

1000.1110.0011.0000.0000.0000.1000.0000

10.001.110.001.100.000.000.000.010.000.000

By inspection, the result is 0x8E300080 and 21614000200.

▸ **QUESTION 20**
Translate the following MIPS machine language instructions to assembly language.

(a) 0x2008001F

(b) 0x00895022

Estimated time to complete this question is 10 minutes.

SOLUTION

(*a*) 0x2008001F
Represent the instruction in binary by converting each hexadecimal digit to it's 4–bit binary value.

0010.0000.0000.1000.0000.0000.0001.1111

The first 6 bits correspond to the *opcode.* Since the opcode is non-zero, the instruction is not R-type. Looking up 001000 in Appendix B yields the instruction

`addi rt, rs, imm`

The instruction is I-type so the machine language fields are represented like this:

opcode	rs	rt	imm
001000	00000	01000	0000.0000.0001.1111

Using Table 6.1 MIPS register set we find rs = 00000 = 0 ($0), rt = 01000 = 8 ($t0). The decimal value of the 16–bit imm is 31.
Therefore, the assembly instruction is:

```
addi $t0, $0, 31
```

(b) 0x00895022
Represent the instruction in binary by converting each hexadecimal digit to it's 4–bit binary value.

0000.0000.1000.1001.0101.0000.0010.0010

The first 6 bits correspond to the *opcode*. Since the opcode is zero, the instruction is R-type. The *funct* field is the last 6 bits 100010. Looking up 100010 in Appendix B yields the instruction

```
sub rd, rs, rt
```

The instruction is R-type so the machine language fields are represented like this:

opcode	rs	rt	rd	shamt	funct
000000	00100	01001	01010	00000	100010

Using Table 6.1 MIPS register set we find rs = 00100 = 4 ($a0), rt = 01001 = 9 ($t1), rd = 01010 = 10 ($t2).
Therefore, the assembly instruction is:

```
sub $t2, $a0, $t1
```

▸ **QUESTION 21**
Translate the following MIPS machine language instructions to assembly language.

(a) 0x01044806

(b) 0xA0A90000

Estimated time to complete this question is 10 minutes.

SOLUTION

(a) 0x01044806
Represent the instruction in binary by converting each hexadecimal digit to it's 4–bit binary value.

0000.0001.0000.0100.0100.1000.0000.0110

The first 6 bits correspond to the *opcode*. Since the opcode is zero, the instruction is R-type. The *funct* field is the last 6 bits 000110. Looking up 000110 in Appendix B yields the instruction

```
srlv rd, rt, rs
```

The instruction is R-type so the machine language fields are represented like this:

opcode	rs	rt	rd	shamt	funct
000000	01000	00100	01001	00000	000110

Using Table 6.1 MIPS register set we find rs = 01000 = 8 ($t0), rt = 00100 = 4 ($a0), rd = 01001 = 9 ($t1).
Therefore, the assembly instruction is:

```
srlv $t1, $a0, $t0
```

(b) 0xA0A90000
Represent the instruction in binary by converting each hexadecimal digit to it's 4–bit binary value.

1010.0000.1010.1001.0000.0000.0000.0000

The first 6 bits correspond to the *opcode*. Since the opcode is non-zero, the instruction is not R-type. Looking up 101000 in Appendix B yields the instruction

```
sb rt, imm(rs)
```

The instruction is I-type so the machine language fields are represented like this:

opcode	rs	rt	imm
101000	00101	01001	0000.0000.0000.0000

Using Table 6.1 MIPS register set we find rs = 00101 = 5 ($a1), rt = 01001 = 9 ($t1). The decimal value of the 16–bit imm is 0.
Therefore, the assembly instruction is:

```
sb $t1, 0($a1)
```

▸ **QUESTION 22**
Translate the following MIPS machine language instructions to assembly language.

(a) 0x31290001

(b) 0x0009482A

Estimated time to complete this question is 10 minutes.

SOLUTION

(a) 0x31290001
Represent the instruction in binary by converting each hexadecimal digit to it's 4–bit binary value.

0011.0001.0010.1001.0000.0000.0000.0001

The first 6 bits correspond to the *opcode*. Since the opcode is non-zero, the instruction is not R-type. Looking up 001100 in Appendix B yields the instruction

```
andi rt, rs, imm
```

The instruction is I-type so the machine language fields are represented like this:

opcode	rs	rt	imm
001100	01001	01001	0000.0000.0000.0001

Using Table 6.1 MIPS register set we find rs = 01001 = 9 ($t1), rt = 01001 = 9 ($t1). The decimal value of the 16–bit imm is 1.
Therefore, the assembly instruction is:

```
andi $t1, $t1, 1
```

(*b*) 0x0009482A
Represent the instruction in binary by converting each hexadecimal digit to it's 4–bit binary value.

0000.0000.0000.1001.0100.1000.0010.1010

The first 6 bits correspond to the *opcode*. Since the opcode is zero, the instruction is R-type. The *funct* field is the last 6 bits 101010. Looking up 101010 in Appendix B yields the instruction

`slt rd, rs, rt`

The instruction is R-type so the machine language fields are represented like this:

opcode	rs	rt	rd	shamt	funct
000000	00000	01001	01001	00000	101010

Using Table 6.1 MIPS register set we find rs = 00000 = 0 ($0), rt = 01001 = 9 ($t1), rd = 01001 = 9 ($t1). Therefore, the assembly instruction is:

`slt $t1, $0, $t1`

QUESTION 23
Consider the following multiplexer logic circuit:

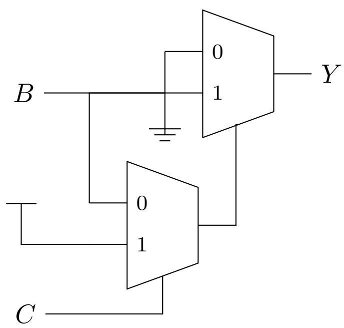

Figure 9: Logic circuit implemented with multiplexers.

,

(a) Draw a complete truth table with minterms and minterm names.

(b) Draw a complete truth table with maxterms and maxterm names.

(c) Write a Boolean equation in sum-of-products canonical form. Simplify the equation.

(d) Write a Boolean equation in product-of-sums canonical form. Simplify the equation.

(e) Which theorem of Boolean algebra does this logic circuit illustrate?

Estimated time to complete this question is 15 minutes.

SOLUTION

(a) Truth table for Figure 9 with minterms and minterm names.

B	C	$Y1$	Y	minterm	names
0	0	0	0	$\overline{B}\,\overline{C}$	m_0
0	1	1	0	$\overline{B}\,C$	m_1
1	0	1	1	$B\,\overline{C}$	m_2
1	1	1	1	$B\,C$	m_3

Figure 10: Truth table for multiplexer logic circuit in Figure 9—sum-of-products.

(b) Truth table for Figure 9 with maxterms and maxterm names.

B	C	$Y1$	Y	minterm	names
0	0	0	0	$B+C$	M_0
0	1	1	0	$B+\overline{C}$	M_1
1	0	1	1	$\overline{B}+C$	M_2
1	1	1	1	$\overline{B}+\overline{C}$	M_3

Figure 11: Truth table for multiplexer logic circuit Figure 9—product-of-sums.

(c) Sum-of-products Boolean equation for truth table Figure 10.

The Boolean equation in sum-of-products canonical form is the sum of the minterms for which Y is 1.

$$\begin{aligned} Y &= m_2 + m_3 \\ &= B\,\overline{C} + B\,C \\ &= B \bullet (\overline{C} + C) \\ &= B \bullet 1 \\ Y &= B \end{aligned} \tag{13}$$

(d) Product-of-sums Boolean equation for truth table Figure 11.

The Boolean equation in product-of-sums canonical form is the product of the maxterms for which Y is 0.

$$\begin{aligned} Y &= M_0 \bullet M_1 \\ &= (B + C) \bullet (B + \overline{C}) \\ &= B\,B + B\,\overline{C} + B\,C + C\,\overline{C} \\ &= B + B \bullet (C + \overline{C}) + 0 \\ &= B + B \bullet 1 \\ &= B + B \\ Y &= B \end{aligned} \tag{14}$$

(e) The Boolean function illustrates the combining theorem T10.

▸ **QUESTION 24**

Consider the following multiplexer logic circuit:

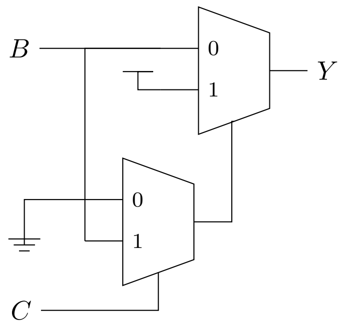

Figure 12: Logic circuit implemented with multiplexers.

,

(a) Draw a complete truth table with minterms and minterm names.

(b) Draw a complete truth table with maxterms and maxterm names.

(c) Write a Boolean equation in sum-of-products canonical form. Simplify the equation.

(d) Write a Boolean equation in product-of-sums canonical form. Simplify the equation.

(e) Which theorem of Boolean algebra does this logic circuit illustrate?

Estimated time to complete this question is 15 minutes.

SOLUTION

(a) Truth table for Figure 12 with minterms and minterm names.

B	C	$Y1$	Y	minterm	names
0	0	0	0	$\overline{B}\,\overline{C}$	m_0
0	1	0	0	$\overline{B}\,C$	m_1
1	0	0	1	$B\,\overline{C}$	m_2
1	1	1	1	$B\,C$	m_3

Figure 13: Truth table for multiplexer logic circuit in Figure 12—sum-of-products.

(b) Truth table for Figure 12 with maxterms and maxterm names.

B	C	$Y1$	Y	minterm	names
0	0	0	0	$B + C$	M_0
0	1	0	0	$B + \overline{C}$	M_1
1	0	0	1	$\overline{B} + C$	M_2
1	1	1	1	$\overline{B} + \overline{C}$	M_3

Figure 14: Truth table for multiplexer logic circuit Figure 12—product-of-sums.

(c) Sum-of-products Boolean equation for truth table Figure 13.

The Boolean equation in sum-of-products canonical form is the sum of the minterms for which Y is 1.

$$\begin{aligned} Y &= m_2 + m_3 \\ &= B\,\overline{C} + B\,C \\ &= B \bullet (\overline{C} + C) \\ &= B \bullet 1 \\ Y &= B \end{aligned} \tag{15}$$

(d) Product-of-sums Boolean equation for truth table Figure 14.

The Boolean equation in product-of-sums canonical form is the product of the maxterms for which Y is 0.

$$\begin{aligned} Y &= M_0 \bullet M_2 \\ &= (B + C) \bullet (B + \overline{C}) \\ &= B\,B + B\,\overline{C} + B\,C + C\,\overline{C} \\ &= B + B \bullet (C + \overline{C}) + 0 \\ &= B + B \bullet 1 \\ &= B + B \\ Y &= B \end{aligned} \tag{16}$$

(e) The Boolean function illustrates the covering combining T10 dual.

▸ **QUESTION 25**

Use perfect induction to prove that the two logic gates in Figure 15 are equivalent.

Figure 15: DeMorgan equivalent logic.

Estimated time to complete this question is 5 minutes.

SOLUTION

Apply perfect induction by making a truth table for each logic gate. If the logic gates perform the same function, the truth tables will have the same inputs and outputs.

B	C	Y
0	0	1
0	1	1
1	0	1
1	1	0

Figure 16: Truth table for NAND gate, Figure 15

B	C	Y
0	0	1
0	1	1
1	0	1
1	1	0

Figure 17: Truth table for OR gate with inverted inputs, Figure 15

▸ **QUESTION 26**

Use perfect induction to prove that the two logic gates in Figure 18 are equivalent.

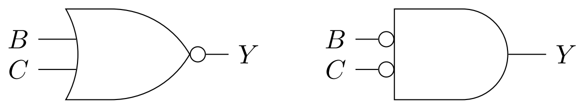

Figure 18: DeMorgan equivalent logic.

Estimated time to complete this question is 5 minutes.

SOLUTION

Apply perfect induction by making a truth table for each logic gate. If the logic gates perform the same function, the truth tables will have the same inputs and outputs.

B	C	Y
0	0	1
0	1	0
1	0	0
1	1	0

Figure 19: Truth table for NOR gate, Figure 18

B	C	Y
0	0	1
0	1	0
1	0	0
1	1	0

Figure 20: Truth table for AND gate with inverted inputs, Figure 18

▸ **QUESTION 27**

Consider the following logic circuit:

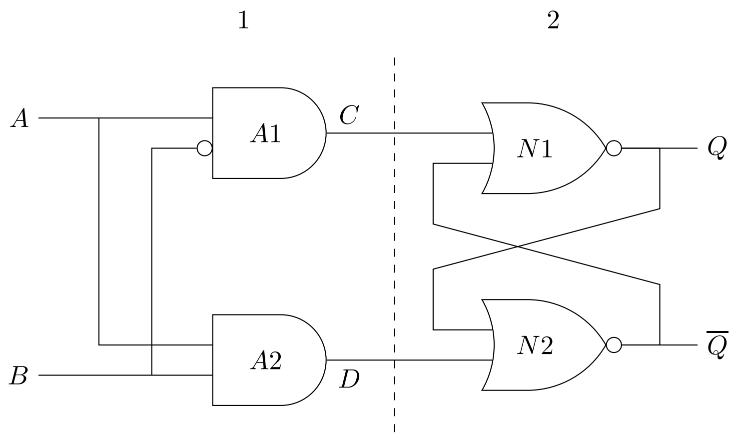

Figure 21: Logic circuit.

(a) Identify the logic circuit in Figure 21. Is it an example of combinational or sequential logic. Why?

(b) Explain in detail the operation of this circuit. In other words, for various inputs, what are the outputs of each gate on each indicated literal A, B, C, D, Q, $\overline{Q}$.

(c) There are two distinct elements in this circuit. They are numbered 1 and 2 and are separated by a dashed line. Explain the purpose and function of the individual elements. Characterize each individual element as combination or sequential.

(d) Draw a complete truth table for the logic circuit in Figure 21. In your truth table show the values of each labeled literal corresponding to inputs A and B.

Estimated time to complete this question is 15 minutes.

SOLUTION

(a) The Logic circuit in Figure 21 is a D latch. Overall it is an example of sequential logic. It's current output is a function of it's current and previous inputs.

(b) The operation of this circuit is explained in detail as a function of it's inputs and outputs below:

Case 1: $A = 1, B = 0$:
$A1$ has two TRUE inputs so it's C output is TRUE. $A2$ has only one TRUE input so it's D output is FALSE. $N1$ sees at least 1 TRUE input C so it produces a FALSE output on Q. $N2$ sees both Q and D FALSE, so it produces a TRUE output on $\overline{Q}$.

Case 2: $A = 1, B = 1$:
$A1$ has only one TRUE input so it's output is FALSE. $A2$ has two TRUE inputs so it's output is TRUE. $N2$ has at least one TRUE input so it outputs FALSE on $\overline{Q}$. $N1$ has inputs FALSE from $A1$ on C and FALSE from $N1$ on $\overline{Q}$ so it's output is TRUE.

Case 3: $A = 0, B = 0, 1$:
$A1$ has at least one FALSE input so it's C output is FALSE. $A2$ has at lease one FALSE input so it's output is FALSE.

$N1$ receives inputs of FALSE and $\overline{Q}$. Because we don't yet know $\overline{Q}$, we can't determine the output Q. $N2$ receives inputs of FALSE and Q. Since we don't know Q or $\overline{Q}$ we can't determine the state of the outputs from the inputs alone. The outputs remain Q_{prev}.

(c) The D Latch combines an SR latch (circuit element 2) with added logic of two AND gates (circuit element 1) to eliminate the ambiguous SR latch behavior when Set (C) and Reset (D) are simultaneously asserted HIGH. Circuit element 1 is combinational. Circuit element 2 (SR latch) is sequential.

(d) Truth table for D latch in Figure 21.

A	B	C	D	Q	$\overline{Q}$
0	0	0	0	Q_{prev}	$\overline{Q}_{prev}$
0	1	0	0	Q_{prev}	$\overline{Q}_{prev}$
1	0	1	0	0	1
1	1	0	1	1	0

Figure 22: Truth table for sequential logic circuit—Figure 21

▸ **QUESTION 28**

Given the decimal number 447...

(a) How many whole binary digits, N, are needed to represent this decimal number? Use the base conversion formula 1.1 on page 8 of the text to compute N.

(b) Convert this number to $N+1$–bit binary where N is the number of bits you computed in the first step. Use a calculator for this step.

(c) Considering this number the *magnitude*, convert to it's negative two's complement binary representation. Do not use your calculator for this step. Show your work.

(d) Sign-extend the $N+1$–bit two's complement binary number to 32 binary digits.

Estimated time to complete this question is 5 minutes.

SOLUTION

In general, the number of possible combinations for a number system of base b and digits N_b is

$$M_b = b^{N_b}$$

For a given number R_k of base k, we can write

$$M_b = R_k$$

Since we are solving for N_b, we substitute and write

$$b^{N_b} = R_k$$

We can solve for N_b by taking log_b of each side of the relationship and solving for N_b.

$$log_b(b^{N_b}) = log_b(R_k)$$

Which may be rewritten as

$$N_b \bullet log_b(b) = log_b(R_k)$$

Since $log_b(b) = 1$, we can further rewrite as

$$N_b = log_b(R_k)$$

Since $b = 2$ in this case, we can write the equation as follows:

$$N_2 = log_2(R_k)$$

This is how we will compute the number of binary digits.

(a) We are given $k = 10$, where k is the base—i.e. decimal—and $R_{10} = 447$, where R_{10} is the number represented in decimal.

The binary range, M_2, represented by the decimal number according to the relationships established above is:

$$\begin{aligned} N_2 &= log_2(R_k) \\ &= log_2(447) \\ N_2 &= 8.8041 \end{aligned} \tag{17}$$

The number of whole binary digits necessary to represent the decimal number 447 is 9.

(b) Convert this number to 9–bit binary using a calculator.

1.1011.1111

(c) Convert the 9–bit binary to it's complement representation. Invert each bit and add 1.

1.0100.0001

(d) Sign-extend to 32 bits.

Copy the sign bit into the most significant bit positions.

1111.1111.1111.1111.1111.1110.0100.0001

▸ **QUESTION 29**

Given the decimal number 384…

(a) How many whole binary digits, N, are needed to represent this decimal number? Use the base conversion formula 1.1 on page 8 of the text to compute N.

(b) Convert this number to $N + 1$–bit binary where N is the number of bits you computed in the first step. Use a calculator for this step.

(c) Considering this number the *magnitude*, convert to it's negative two's complement binary representation. Do not use your calculator for this step. Show your work.

(d) Sign-extend the $N + 1$–bit two's complement binary number to 32 binary digits.

Estimated time to complete this question is 5 minutes.

SOLUTION

In general, the number of possible combinations for a number system of base b and digits N_b is

$$M_b = b^{N_b}$$

For a given number R_k of base k, we can write

$$M_b = R_k$$

Since we are solving for N_b, we substitute and write

$$b^{N_b} = R_k$$

We can solve for N_b by taking log_b of each side of the relationship and solving for N_b.

$$log_b(b^{N_b}) = log_b(R_k)$$

Which may be rewritten as

$$N_b \bullet log_b(b) = log_b(R_k)$$

Since $log_b(b) = 1$, we can further rewrite as

$$N_b = log_b(R_k)$$

Since $b = 2$ in this case, we can write the equation as follows:

$$N_2 = log_2(R_k)$$

This is how we will compute the number of binary digits.

(a) We are given $k = 10$, where k is the base—i.e. decimal—and $R_{10} = 384$,

where R_{10} is the number represented in decimal.

The binary range, M_2, represented by the decimal number according to the relationships established above is:

$$\begin{aligned} N_2 &= log_2(R_k) \\ &= log_2(384) \\ N_2 &= 8.5849 \end{aligned} \tag{18}$$

The number of whole binary digits necessary to represent the decimal number 384 is 9.

(b) Convert this number to 10–bit binary using a calculator.

01.1000.0000

(c) Convert the 10–bit binary to it's complement representation. Invert each bit and add 1.

10.1000.0000

(d) Sign-extend to 32 bits.

Copy the sign bit into the most significant bit positions.

1111.1111.1111.1111.1111.1110.1000.0000

▸ **QUESTION 30**

Given the decimal number 513...

(a) How many whole binary digits, N, are needed to represent this decimal number? Use the base conversion formula 1.1 on page 8 of the text to compute N.

(b) Convert this number to $N+1$–bit binary where N is the number of bits you computed in the first step. Use a calculator for this step.

(c) Considering this number the *magnitude*, convert to it's negative two's complement binary representation. Do not use your calculator for this step. Show your work.

(d) Sign-extend the $N+1$–bit two's complement binary number to 32 binary digits.

Estimated time to complete this question is 5 minutes.

SOLUTION

In general, the number of possible combinations for a number system of base b and digits N_b is

$$M_b = b^{N_b}$$

For a given number R_k of base k, we can write

$$M_b = R_k$$

Since we are solving for N_b, we substitute and write

$$b^{N_b} = R_k$$

We can solve for N_b by taking log_b of each side of the relationship and solving for N_b.

$$log_b(b^{N_b}) = log_b(R_k)$$

Which may be rewritten as

$$N_b \bullet log_b(b) = log_b(R_k)$$

Since $log_b(b) = 1$, we can further rewrite as

$$N_b = log_b(R_k)$$

Since $b = 2$ in this case, we can write the equation as follows:

$$N_2 = log_2(R_k)$$

This is how we will compute the number of binary digits.

(a) We are given $k = 10$, where k is the base—i.e. decimal—and $R_{10} = 513$, where R_{10} is the number represented in decimal.

The binary range, M_2, represented by the decimal number according to the relationships established above is:

$$\begin{aligned} N_2 &= log_2(R_k) \\ &= log_2(513) \\ N_2 &= 9.0028 \end{aligned} \tag{19}$$

The number of whole binary digits necessary to represent the decimal number 513 is 10.

(b) Convert this number to 11–bit binary using a calculator.

$$010.0000.0001$$

(c) Convert the 10–bit binary to it's complement representation. Invert each bit and add 1.

$$101.1111.1111$$

(d) Sign-extend to 32 bits.

Copy the sign bit into the most significant bit positions.

$$1111.1111.1111.1111.1111.1101.1111.1111$$

▸ **QUESTION 31**

Given the decimal number 257...

(a) How many whole binary digits, N, are needed to represent this decimal number? Use the base conversion formula 1.1 on page 8 of the text to compute N.

(b) Convert this number to $N+1$–bit binary where N is the number of bits you computed in the first step. Use a calculator for this step.

(c) Considering this number the *magnitude*, convert to it's negative two's complement binary representation. Do not use your calculator for this step. Show your work.

(d) Sign-extend the $N+1$–bit two's complement binary number to 32 binary digits.

Estimated time to complete this question is 5 minutes.

SOLUTION

In general, the number of possible combinations for a number system of base b and digits N_b is

$$M_b = b^{N_b}$$

For a given number R_k of base k, we can write

$$M_b = R_k$$

Since we are solving for N_b, we substitute and write

$$b^{N_b} = R_k$$

We can solve for N_b by taking log_b of each side of the relationship and solving for N_b.

$$log_b(b^{N_b}) = log_b(R_k)$$

Which may be rewritten as

$$N_b \bullet log_b(b) = log_b(R_k)$$

Since $log_b(b) = 1$, we can further rewrite as

$$N_b = log_b(R_k)$$

Since $b = 2$ in this case, we can write the equation as follows:

$$N_2 = log_2(R_k)$$

This is how we will compute the number of binary digits.

(a) We are given $k = 10$, where k is the base—i.e. decimal—and $R_{10} = 257$,

where R_{10} is the number represented in decimal.

The binary range, M_2, represented by the decimal number according to the relationships established above is:

$$\begin{aligned} N_2 &= log_2(R_k) \\ &= log_2(257) \\ N_2 &= 8.0056 \end{aligned} \tag{20}$$

The number of whole binary digits necessary to represent the decimal number 257 is 9.

(b) Convert this number to 10–bit binary using a calculator.

01.0000.0001

(c) Convert the 10–bit binary to it's complement representation. Invert each bit and add 1.

10.1111.1111

(d) Sign-extend to 32 bits.

Copy the sign bit into the most significant bit positions.

1111.1111.1111.1111.1111.1110.1111.1111