CS2208a

Lab #1

September 23, 2013

General Information

- ***Graduate Teaching Assistants (TA's)***
  - Abdulwahab Kabini, *akabani5@uwo.ca*
  - Mike Molnar, *mmolnar2@uwo.ca*
  - Sakif Pritom, *spritom@uwo.ca*
- Email any questions to all three TA's.
- Check the book and lecture notes before emailing us.
- Email assignment marking issues to the TA that marked it.

2

Lab #1 Outline

- Number Systems
- Conversion Methods
- Binary Addition
- Binary Subtraction
- Binary Multiplication
- Fraction Conversions

3

Number Systems

- We will only consider four numbering systems:
  - Binary
  - Octal
  - Decimal
  - Hexadecimal

Base 2	Base 8	Base 10	Base 16
0	0	0	0
1	1	1	1
	2	2	2
	3	3	3
	4	4	4
	5	5	5
	6	6	6
	7	7	7
		8	8
		9	9
			A
			B
			C
			D
			E
			F

4

Conversion Methods

- Division Method
  - Divide the number by the new base.
  - Keep the whole number of the result and the remainder.
  - Use the whole number for the next division.
  - Repeat until the whole number is 0.
  - Converts from decimal to binary, octal, or hexadecimal.

5

Division Method

- Convert 14_{10} to binary:
  - 14/2 = 7 Remainder: 0
  - 7/2 = 3 Remainder: 1
  - 3/2 = 1 Remainder: 1
  - 1/2 = 0 Remainder: 1
- The remainders from the bottom to the top is the result.
  - $14_{10} = 1110_2$

6

Division Method

10 = A
11 = B
12 = C
13 = D
14 = E
15 = F

- Convert 2477_{10} to base 16:

  2477/16 = 154 Remainder: 13 = D

  154/16 = 9 Remainder: 10 = A

  9/16 = 0 Remainder: 9 = 9

- The remainders from the bottom to the top is the result.

  $2477_{10} = 9AD_{16}$

7

Conversion Methods

- Positional Representation
  - Find the value that each digit represents.
  - The position of the digit is important.
  - Add the values to find the results.
  - Converts to decimal from binary, octal, or hexadecimal.

8

Positional Representation

- Convert $2E8_{16}$ to decimal:

$$\begin{aligned} 2E8_{16} &= 2\times16^2 + E\times16^1 + 8\times16^0 \\ &= 2\times256 + 14\times16 + 8\times1 \\ &= 512 + 224 + 8 \\ &= 744_{10} \end{aligned}$$

- Convert 361_8 to decimal:

$$\begin{aligned} 361_8 &= 3\times8^2 + 6\times8^1 + 1\times8^0 \\ &= 3\times64 + 6\times8 + 1\times1 \\ &= 192 + 48 + 1 \\ &= 241_{10} \end{aligned}$$

9

Conversion Methods

- <u>Grouping Method</u>
  - Convert between binary and octal or hexadecimal.
  - Base 8 = 2^3 for binary, so group bits in three's.
  - Base 16 = 2^4 for binary, so group bits in four's.
  - Pad the left most group with 0's if needed.
  - Converts between binary and octal or hexadecimal.

10

Grouping Method

- Convert 1100 1101_2 to octal:

Base 2	011	001	101
Base 8	3	1	5

- Convert 743_8 to binary:

Base 8	7	4	3
Base 2	111	100	011

0 = 000
1 = 001
2 = 010
3 = 011
4 = 100
5 = 101
6 = 110
7 = 111

11

Grouping Method

- Convert 0111 1101 0101_2 to hexadecimal:

Base 2	0111	1101	0101
Base 16	7	D	5

- Convert $FA9_{16}$ to binary:

Base 16	F	A	9
Base 2	1111	1010	1001

0 = 0000
1 = 0001
2 = 0010
3 = 0011
4 = 0100
5 = 0101
6 = 0110
7 = 0111
8 = 1000
9 = 1001
A = 1010
B = 1011
C = 1100
D = 1101
E = 1110
F = 1111

12

Grouping Method

- To convert from hexadecimal to octal, you do it through binary conversion:

- Convert $FA9_{16}$ to octal
- $FA9_{16}$ → 1111 1010 1001
  → 111 110 101 001
  → 7 6 5 1

- $FA9_{16}$ → 7651_8

0 = 0000	8 = 1000
1 = 0001	9 = 1001
2 = 0010	A = 1010
3 = 0011	B = 1011
4 = 0100	C = 1100
5 = 0101	D = 1101
6 = 0110	E = 1110
7 = 0111	F = 1111

13

Grouping Method

- To convert from octal to hexadecimal you do it through binary conversion:

- Convert 315_8 to hexadecimal
- 315_8 → 011 001 101
  → 11001101
  → 1100 1101
  → C D
- 315_8 → CD_{16}

0 = 0000	8 = 1000
1 = 0001	9 = 1001
2 = 0010	A = 1010
3 = 0011	B = 1011
4 = 0100	C = 1100
5 = 0101	D = 1101
6 = 0110	E = 1110
7 = 0111	F = 1111

14

Binary Addition

- 0 + 0 = 0
- 0 + 1 = 1
- 1 + 1 = 0 Carry 1
- 0 + 0 + 0 = 0
- 0 + 0 + 1 = 1
- 0 + 1 + 1 = 0 Carry 1
- 1 + 1 + 1 = 1 Carry 1

15

Binary Addition – Two's Complement

$$\begin{array}{r} 1\,1\,0\,0\quad 1\,0\,0\,1 \\ +\,1\,1\,1\,1\quad 1\,1\,1\,1 \\ \hline \end{array}$$

16

Binary Addition – Two's Complement

$$\begin{array}{ccccccccc}
 & & & & & & & 1 & \\
 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\
+ & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
\hline
 & & & & & & & & 0
\end{array}$$

17

Binary Addition – Two's Complement

$$\begin{array}{ccccccccc}
 & & & & & & 1 & 1 & \\
 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\
+ & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
\hline
 & & & & & & & 0 & 0
\end{array}$$

18

Binary Addition – Two's Complement

$$\begin{array}{ccccccccc}
 & & & & & 1 & 1 & 1 & \\
 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\
+ & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
\hline
 & & & & & & 0 & 0 & 0
\end{array}$$

19

Binary Addition – Two's Complement

$$\begin{array}{ccccccccc}
 & & & & 1 & 1 & 1 & 1 & \\
 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\
+ & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
\hline
 & & & & & 1 & 0 & 0 & 0
\end{array}$$

20

Binary Addition – Two's Complement

			1	1	1	1	1	
	1	1	0	0	1	0	0	1
+	1	1	1	1	1	1	1	1
				0	1	0	0	0

21

Binary Addition – Two's Complement

		1	1	1	1	1	1	
	1	1	0	0	1	0	0	1
+	1	1	1	1	1	1	1	1
			0	0	1	0	0	0

22

Binary Addition – Two's Complement

	1	1	1	1	1	1	1	
	1	1	0	0	1	0	0	1
+	1	1	1	1	1	1	1	1
		1	0	0	1	0	0	0

23

Binary Addition – Two's Complement

1	1	1	1	1	1	1	1	
	1	1	0	0	1	0	0	1
+	1	1	1	1	1	1	1	1
	1	1	0	0	1	0	0	0

- No overflow because the sign of the sum matches sign of operands.
- The carry out is ignored in two's complement.

24

Binary Addition – Unsigned

```
   1 1 0 0   1 0 0 1
+  1 1 1 1   1 1 1 1
-------------------
```


Binary Addition – Unsigned

```
1  1 1 1 1   1 1 1
   1 1 0 0   1 0 0 1
+  1 1 1 1   1 1 1 1
-------------------
   1 1 0 0   1 0 0 0
```

- Overflow occurred because there was a carry out.

Binary Addition – Sign and Magnitude

```
   1 1 0 0   1 0 0 1
+  1 1 1 1   1 1 1 1
-------------------
```


Binary Addition – Sign and Magnitude

```
1      1 1 1   1 1 1
   1 | 1 0 0   1 0 0 1
+  1 | 1 1 1   1 1 1 1
---------------------
   1 | 1 0 0   1 0 0 0
```

- Overflow since the magnitude of the result was too large to fit in 7 bits.
- *The sign is arbitrated separately.*

Binary Addition – Two's Complement

$$
\begin{array}{ccccccccc}
 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\
+ & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\
\hline
\end{array}
$$

29

Binary Addition – Two's Complement

$$
\begin{array}{ccccccccc}
 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\
+ & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\
\hline
 & & & & & & 0 & 1 & 1
\end{array}
$$

30

Binary Addition – Two's Complement

$$
\begin{array}{ccccccccc}
 & & & & 1 & & & & \\
 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\
+ & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\
\hline
 & & & & & 0 & 0 & 1 & 1
\end{array}
$$

31

Binary Addition – Two's Complement

$$
\begin{array}{ccccccccc}
 & & & 1 & 1 & & & & \\
 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\
+ & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\
\hline
 & & & & 0 & 0 & 0 & 1 & 1
\end{array}
$$

32

Binary Addition – Two's Complement

```
      1 1
  0 1 0 0   1 0 1 0
+ 0 1 0 1   1 0 0 1
-------------------
      1 0   0 0 1 1
```

33

Binary Addition – Two's Complement

```
  1   1 1
  0 1 0 0   1 0 1 0
+ 0 1 0 1   1 0 0 1
-------------------
    0 1 0   0 0 1 1
```

34

Binary Addition – Two's Complement

```
  1   1 1
  0 1 0 0   1 0 1 0
+ 0 1 0 1   1 0 0 1
-------------------
  1 0 1 0   0 0 1 1
```

- Overflow because the sign of the sum differs from the sign of operands.

35

Binary Addition - Unsigned

```
  0 1 0 0   1 0 1 0
+ 0 1 0 1   1 0 0 1
-------------------
```

36

Binary Addition - Unsigned

```
   1     1  1
   0  1  0  0     1  0  1  0
+  0  1  0  1     1  0  0  1
-----------------------------
   1  0  1  0     0  0  1  1
```

•No overflow occurred because there was no carry out.

37

Binary Addition – Sign and Magnitude

```
   0  1  0  0     1  0  1  0
+  0  1  0  1     1  0  0  1
-----------------------------
```

38

Binary Addition – Sign and Magnitude

```
1           1  1
   0 | 1  0  0     1  0  1  0
+  0 | 1  0  1     1  0  0  1
-----------------------------
   0 | 0  1  0     0  0  1  1
```

•Overflow since the magnitude of the result was too large to fit in 7 bits.

•*The sign is arbitrated separately.*

39

Binary Addition – Two's Complement

```
   1  1  0  1     1  1  0  0
+  0  1  0  0     0  0  1  0
-----------------------------
```

40

Binary Addition – Two's Complement

	1	1	0	1	1	1	0	0
+	0	1	0	0	0	0	1	0
			0	1	1	1	1	0

41

Binary Addition – Two's Complement

	1							
	1	1	0	1	1	1	0	0
+	0	1	0	0	0	0	1	0
		0	0	1	1	1	1	0

42

Binary Addition – Two's Complement

1	1							
	1	1	0	1	1	1	0	0
+	0	1	0	0	0	0	1	0
	0	0	0	1	1	1	1	0

- Overflow is **not possible** since the signs of the operands differ.
- The carry out is ignored in two's complement.

43

Binary Addition - Unsigned

	1	1	0	1	1	1	0	0
+	0	1	0	0	0	0	1	0

44

Binary Addition - Unsigned

	1	1						
		1	1	0	1	1	1	0	0
+	0	1	0	0	0	0	1	0
	0	0	0	1	1	1	1	0

•Overflow occurred because there was a carry out.

45

Binary Addition – Sign and Magnitude

$$
\begin{array}{r}
1\ 1\ 0\ 1\quad 1\ 1\ 0\ 0 \\
+\ 0\ 1\ 0\ 0\quad 0\ 0\ 1\ 0 \\
\hline
\end{array}
$$

46

Binary Addition – Sign and Magnitude

$$
\begin{aligned}
& 1101\ \ 1100 + 0100\ \ 0010 \\
= & -0101\ \ 1100 + 0100\ \ 0010 \\
= & -(0101\ \ 1100 - 0100\ \ 0010) \\
= & -(0001\ \ 1010) \\
= & \ 1001\ \ 1010
\end{aligned}
$$

Borrow

1
0101 1100
–0100 0010

0001 1010

Subtraction Table
0 – 0 = 0
1 – 0 = 1
0 – 1 = 1 borrow 1
1 – 1 = 0

47

Binary Subtraction –Two's Complement

- Two's complement does not use subtraction.
- Find two's complement of subtrahend and add.
- Overflow if both numbers have the same sign but the result is the opposite sign.

48

Binary Subtraction –Two's Complement

- Let's try –100 – 30
- The first step is to find the two's complement of 30 and 100.
- $30_{10} = 0001\ 1110_2$ $\quad 100_{10} = 0110\ 0100_2$
- Flip (complement) the bits and add 1 to get the two's complement.

$$
\begin{array}{rl}
 & 1 \\
 & 1\,1\,1\,0\quad 0\,0\,0\,1 \leftarrow \text{complement} \\
+ & 1 \\
\hline
 & 1\,1\,1\,0\quad 0\,0\,1\,0 \leftarrow \text{-30}
\end{array}
\qquad
\begin{array}{rl}
 & 1\,1 \\
 & 1\,0\,0\,1\quad 1\,0\,1\,1 \leftarrow \text{complement} \\
+ & 1 \\
\hline
 & 1\,0\,0\,1\quad 1\,1\,0\,0 \leftarrow \text{-100}
\end{array}
$$

- Now we can add –100 and –30.

49

Binary Subtraction –Two's Complement

$$
\begin{array}{rcccccccc}
1 & & & & & & & & \\
 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\
+ & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\
\hline
 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0
\end{array}
$$

•Overflow because the sign of the sum differs from the sign of operands.

50

Binary Multiplication

- 0 x (0 or 1) = 0
- 1 x 1 = 1
- Algorithm for unsigned integers:
  - Start with the least significant bit of the multiplier.
  - If it is a 1 then copy the multiplicand to the partial product.
  - Check the next bit and repeat.
  - Shift the partial product left each time.

51

Binary Multiplication

- $5 \times 6 = 30_{10}$
- $5_{10} = 0101_2$ = Multiplicand
- $6_{10} = 0110_2$ = Multiplier
- $30_{10} = 11110_2$

$$
\begin{array}{cccccccc}
 & & & & 0 & 1 & 0 & 1 \\
 & & & \times & 0 & 1 & 1 & 0 \\
\hline
 & & & & 0 & 0 & 0 & 0 \\
 & & & 0 & 1 & 0 & 1 & \\
 & & 0 & 1 & 0 & 1 & & \\
 & 0 & 0 & 0 & 0 & & & \\
\hline
 & 0 & 0 & 1 & 1 & 1 & 1 & 0
\end{array}
$$

52

Binary Multiplication

- Binary multiplication by powers of 2 uses shifting.
- Shift left arithmetic is used for multiplication.
  - $\times$ 2 = shift left once
  - $\times$ 4 = shift left twice
  - $\times$ 8 = shift left three times
- Division by powers of 2 works in a similar fashion.
  - Shifts are to the right instead of left.
  - Shift logical instead of arithmetic.
  - This maintains the sign of the value.

53

Binary Multiplication

- Multiply 3 by 2, 8, and 16:
  - 3_{10} = 0000 0011_2
  - $\times$ 2 = one shift left = 0000 0110_2 = 6_{10}
  - $\times$ 8 = three shifts left = 0001 1000_2 = 24_{10}
  - $\times$ 16 = four shifts left = 0011 0000_2 = 48_{10}
- Divide –32 by 4 and 8:
  - -32_{10} = 1110 0000_2
  - $\div 4$ = two shifts right = 1111 1000_2 = -8_{10}
  - $\div 8$ = three shifts right = 1111 1100_2 = -4_{10}
  - When shifting right we **replicate** the sign of the number.

54

Fraction Conversions

- Convert from decimal to binary.
- Convert the whole number with division method.
- Convert the fractional part with a multiplication method.
- Convert 12.6875_{10} to binary.
- Convert 12_{10} to binary first:
  - 12/2 = 6 R=0
  - 6/2 = 3 R=0
  - 3/2 = 1 R=1
  - 1/2 = 0 R=1
- Reading upwards gives 1100_2 as the whole part in binary.

55

Fraction Conversions

- Next we have to convert the fractional part.
- Similar to division method but uses multiplication.
- The result is read from the top to bottom.
- Keep *whole part* and use remaining fraction for next digit.
- Convert $.6875_{10}$ to binary:
  - .6875 x 2 = 1.375 WP=1
  - .375 x 2 = 0.75 WP=0
  - .75 x 2 = 1.5 WP=1
  - .5 x 2 = 1.0 WP=1
- Reading from top to bottom gives $.1011_2$ as the result.

56

Fraction Conversions

- Combining the two results gives the binary representation.
- $12.6875_{10} = 1100.1011_2$
- Find the 32-bit representation of the result:
  - The normalized result is 1.1001011_2
  - The exponent in binary is $127+3 = 130_{10} = 10000010_2$
  - The sign bit is 0 because the number is positive.
  - The result is $0100\ 0001\ 0100\ 1011\ 0000\ 0000\ 0000\ 0000_2$.

57

Fraction Conversions

- We can now convert the result to octal or hexadecimal.
- Use the grouping method to find the answer.

Base 2	001	000	001	010	010	110	000	000	000	000	000
Base 8	1	0	1	2	2	6	0	0	0	0	0

Base 2	0100	0001	0100	1011	0000	0000	0000	0000
Base 16	4	1	4	B	0	0	0	0

- The answer in octal is 10122600000_8.
- The answer in hexadecimal is $414B0000_{16}$.

58