CS2100

TUTORIAL #1

C AND NUMBER SYSTEMS

(PREPARED BY: AARON TAN)

Q1. Sign extension – extending the sign bit to fill in the bit representation.

Example: from 4 bits to 8 bits.

$$(5)_{10} = (0101)_{2s} = (00000101)_{2s}$$

$$(-3)_{10} = (1101)_{2s} = (11111101)_{2s}$$

Q1. Show that sign-extension preserves values in general

Case 1: Negative values in 2's complement
Let $0 \le x \le 2^{n-1} - 1$ be a n-bit number

We know that in 2's complement,
$-2^{n-1} = 1_{n-1}0_{n-2} \dots 0_0$ and
$2^{n-1} - 1 = 0_{n-1}1_{n-2} \dots 1_0$

MSB has weight -2^{n-1}
Bit r has weight 2^r

So $-x = 1_{n-1}\boxed{b_{n-2} \dots b_0} = -2^{n-1} + \boxed{\sum_{r=0}^{n-2}(b \times 2^r)}$
Now consider $y = 1_{m-1}1_{m-2} \dots 1_{n-1}\boxed{b_{n-2} \dots b_0}$
If we compare the representation of $-x$ and y, we only need to show that in the representation of y,

$$-2^{n-1} = 1_{m-1} \dots 1_{n-1}$$

Q1. Show that sign-extension preserves values in general

Case 1: Negative values in 2's complement

We want to show that in the representation of y,

$$-2^{n-1} = 1_{m-1} \dots 1_{n-1}$$

$$1_{m-1} \dots 1_{n-1} = -2^{m-1} + \sum_{r=n-1}^{m-2}(2^r)$$

$$1_{m-1} \dots 1_{n-1} = -2^{m-1} + 2^{n-1}\sum_{r=0}^{m-n-1}(2^r)$$

$$1_{m-1} \dots 1_{n-1} = -2^{m-1} + 2^{n-1}(2^{m-n} - 1)$$

$$1_{m-1} \dots 1_{n-1} = -2^{m-1} + 2^{m-1} - 2^{n-1}$$

$$1_{m-1} \dots 1_{n-1} = -2^{n-1}$$

Recall that sum of a GP is $\sum_{r=0}^{k-1}(2^r) = \frac{2^k-1}{2-1}$.

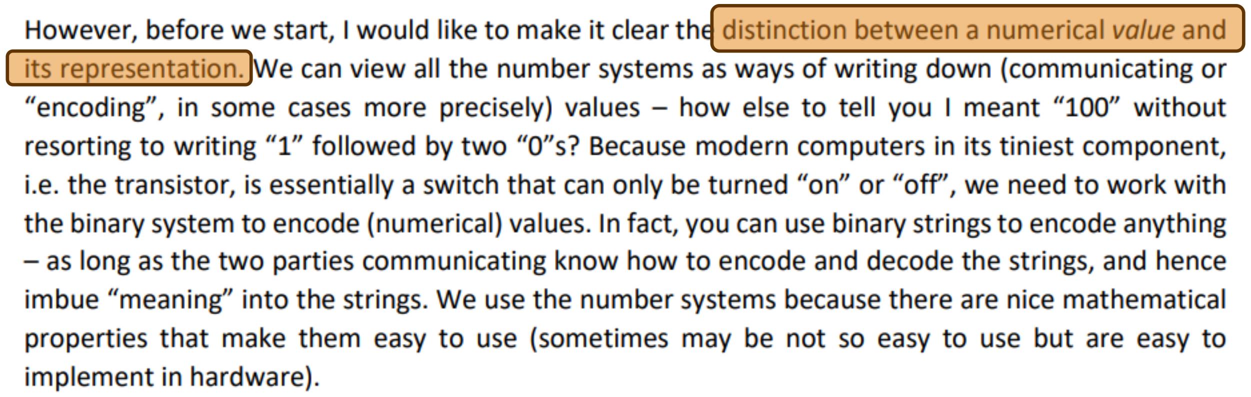
Q1. Show that sign-extension preserves values in general

Case 2: Positive values (straightforward)

Main Idea: $0 + x = x$

Important Notes

From Prof. Wong:


However, before we start, I would like to make it clear the distinction between a numerical *value* and its representation. We can view all the number systems as ways of writing down (communicating or "encoding", in some cases more precisely) values – how else to tell you I meant "100" without resorting to writing "1" followed by two "0"s? Because modern computers in its tiniest component, i.e. the transistor, is essentially a switch that can only be turned "on" or "off", we need to work with the binary system to encode (numerical) values. In fact, you can use binary strings to encode anything – as long as the two parties communicating know how to encode and decode the strings, and hence imbue "meaning" into the strings. We use the number systems because there are nice mathematical properties that make them easy to use (sometimes may be not so easy to use but are easy to implement in hardware).

In the earlier slides, we have shown that in 2's complement, $1_{m-1} \dots 1_{n-1}$, $-2^{m-1} + \sum_{r=n-1}^{m-2}(2^r)$, -2^{n-1} are all different representations for the same numerical value.

Important Notes

We can treat binary 2's complement like weighted positional number systems.

$(1101)_2$ represents the value $2^3 + 2^2 + 0 + 2^0 = 13$
$(1101)_{2s}$ represents the value $\boxed{-2^{-3}} + 2^2 + 0 + 2^0 = -3$

How do we know the weight for the MSB position?
This is why we consider $(1000)_{2s}$. If binary 2's complement are weighted positional number systems, then the values represented by $(1000)_{2s}$ must give us the weights of the MSB.

Important Notes

We can treat binary 2’s complement like weighted positional number systems. Note that this happens to be a convenient result for binary 2’s complement because:

1. Half of the values that can be represented are non-negative (0 to $2^{n-1} - 1$) and the other half of the values are negative (-1 to 2^{n-1}).
2. A bit has two possible values: 0 or 1.

So all of the negative numbers end up with MSB = 1 and all of the positive numbers end up with MSB = 0 when we apply the rule
$rep(-N) = rep(2^n - N)$. Where $rep(x)$ is the binary representation of value x, $1 \leq x < 2^{n-1}$.

Now if we are not using base 2, this property, together with properties like flipping bits to obtain radix diminished complement representation no longer holds. We would need to go back to the definition of $rep(-N) = rep(r^n - N)$ for n-digit, r-complement in base r and $rep(-N) = rep(r^n - r^{-m} - N)$ for n-digit, $(r-1)$-complement in base r. In this case, $rep(x)$ is the base r representation of x.

As an exercise, consider 3’s-complement in base 3 and see that you cannot flip 0s to 1s or assign a weight to the most significant digit.

Also try last semester’s assignment question at the end of the slides.

Q1. Sign extension – extending the sign bit to fill in the bit representation.

Example: from 4 bits to 8 bits.

$$(5)_{10} = (0101)_{2s} = (00000101)_{2s}$$

$$(-3)_{10} = (1101)_{2s} = (11111101)_{2s}$$

We have shown that sign extension works for complement systems (1’s complement and 2’s complement.)

Does it work for sign-and-magnitude system?
No because magnitude changes when extending a negative value

Q2. Performing subtraction in 1's complement.

Strategy: Convert $A - B$ to $A + (-B)$.

(a) 0101.11 – 010.0101

A = 0101.1100, B = 0010.0101

Step 1: Convert -B to 1s-complement

-B = 1101.1010

Step 2: Calculate A + (-B)

0101.1100 + 1101.1010 = 0011.0110

+ 0000.0001 = 0011.0111

Q2. Performing subtraction in 1's complement.

Strategy: Convert $A - B$ to $A + (-B)$.

(b) 010111.101 – 0111010.11

A = 0010111.101, B = 0111010.110

Step 1: Convert -B to 1s-complement

-B = 1000101.001

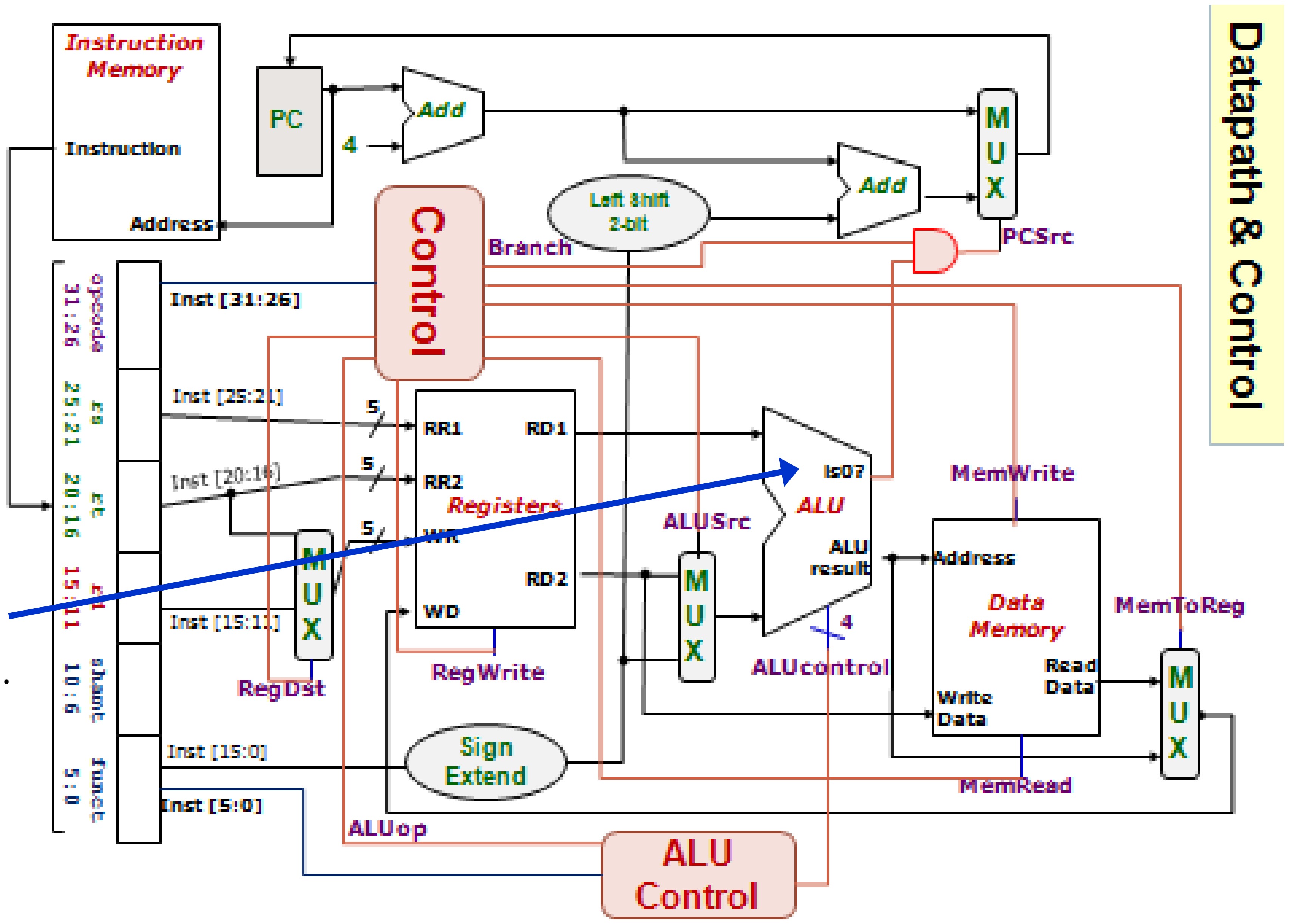
Step 2: Calculate A + (-B)

0010111.101 + 1000101.001 = 1011100.110

Why not perform subtraction directly?

A student suggested overflow as the reason. Overflow could happen if A < 0 and B > 0. Then A – B would be more negative than A which could lead to overflow (just let A be the most negative number that can be represented). However, this can also occur with our 1's complement method. We can also find values such that the addition overflows.

Q1.



Why not perform subtraction directly? Allows us to re-use hardware for addition.

Q3. Convert the following decimal numbers to fixed-point binary in 2’s complement, with 4 bits for the integer portion and 3 bits for the fraction portion.

(a) 1.75

-2^3	2^2	2^1	2^0	.	2^{-1}	2^{-2}	2^{-3}
0	0	0	1	.	1	1	0

Q3. Convert the following decimal numbers to fixed-point binary in 2’s complement, with 4 bits for the integer portion and 3 bits for the fraction portion.

(b) -2.5

	-2^3	2^2	2^1	2^0	.	2^{-1}	2^{-2}	2^{-3}
2.5	0	0	1	0	.	1	0	0
2’s	1	1	0	1	.	1	0	0

Q3. Convert the following decimal numbers to fixed-point binary in 2's complement, with 4 bits for the integer portion and 3 bits for the fraction portion.

(c) 3.876

$0.876 = (0.1110)_2 \approx (0.111)_2$

Therefore $3.876 = (0011.111)_2$

$= \mathbf{(0011.111)_{2s}}$

$0.876 \times 2 = 1.752$
$0.752 \times 2 = 1.504$
$0.504 \times 2 = 1.008$
$0.008 \times 2 = 0.016$

Q3. What if we want to express decimal value 7.876 in fixed-point binary, 2's complement, 4 integer bits, 2 fractional bits?

$0.876 = (0.111)_2 \quad \approx (1.00)_2$

Then $7.876 = (1000.00)_2$

$= \mathbf{(1000.00)_{2s}}$

$0.876 \times 2 = 1.752$
$0.752 \times 2 = 1.504$
$0.504 \times 2 = 1.008$

If we convert this value back to decimal, we would get -8.
Which means that we cannot represent 7.876 in this representation. This makes sense because the largest positive value in our representation is 7.75, represented as 0111.11.

Q3. Convert the following decimal numbers to fixed-point binary in 2's complement, with 4 bits for the integer portion and 3 bits for the fraction portion.

(d) 2.1

$0.1 = (0.0001)_2 \approx (0.001)_2$

Therefore $2.1 = (0010.001)_2$

$= \mathbf{(0010.001)_{2s}}$

$0.1 \times 2 = 0.2$
$0.2 \times 2 = 0.4$
$0.4 \times 2 = 0.8$
$0.8 \times 2 = 1.6$

Q3. Using the binary representations you have just derived, convert them back into decimal.

(a) 1.75

$= \mathbf{(0001.110)_{2s}}$

$= (0001.110)_2$

$(0001.110)_2$

$= 2^0 + 2^{-1} + 2^{-2}$

$= 1.75$

(b) -2.5

$= \mathbf{(1101.100)_{2s}}$

$= -(0010.100)_2$

$-(0010.100)_2$

$= -(2^1 + 2^{-1})$

$= -2.5$

(c) 3.876

$= \mathbf{(0011.111)_{2s}}$

$= (0011.111)_2$

$(0011.111)_2$

$= 2^1 + 2^0$

$+ 2^{-1} + 2^{-2} + 2^{-3}$

$= 3.875$

(d) 2.1

$= \mathbf{(0010.001)_{2s}}$

$= (0010.001)_2$

$(0010.001)_2$

$= 2^1 + 2^{-3}$

$= 2.125$

Q3. Using the binary representations you have just derived, convert them back into decimal.

-2^3	2^2	2^1	2^0	.	2^{-1}	2^{-2}	2^{-3}	Value
0	0	1	1	.	1	1	0	3.75
0	0	1	1	.	1	1	1	3.875
0	1	0	0	.	0	0	0	4
0	0	1	0	.	0	0	0	2
0	0	1	0	.	0	0	1	2.125

Q4. How would you represent the decimal value -0.078125 in the IEEE 754 single-precision representation? Express your answer in hexadecimal.

Step 1:
Convert decimal value to binary representation
$(-0.078125)_{10} = (-0.000101)_2$

Step 2:
Convert binary representation to binary scientific notation
$(-0.000101)_2 = -(1.01)_2 \times 2^{-4}$

Q4. How would you represent the decimal value -0.078125 in the IEEE 754 single-precision representation? Express your answer in hexadecimal.

Step 3

$$-(1.01)_2 \times 2^{-4}$$

Exponent: $-4 + 127$
$= 123$
$= (01111011)_2$

1	01111011	01000000000000000000000
Sign-bit	8-bit exponent	23-bit mantissa

$$\underbrace{1\,011}_{B}\ \underbrace{1101}_{D}\ \underbrace{1\,010}_{A}\ \underbrace{0000}_{0}\ \underbrace{0000}_{0}\ \underbrace{0000}_{0}\ \underbrace{0000}_{0}\ \underbrace{0000}_{0}$$

1 01111011 01000000000000000000000

B D A 0 0 0 0 0

Q5. Implement readArray and reverseArray

```
readArray:
Loop through array
scanf("%d", &value);
arr[i] = value;

reverseArray:
Swapping elements
    temp = arr[left];
    arr[left] = arr[right];
    arr[right] = temp;
Loop condition
    left < right
```

There are other implementations

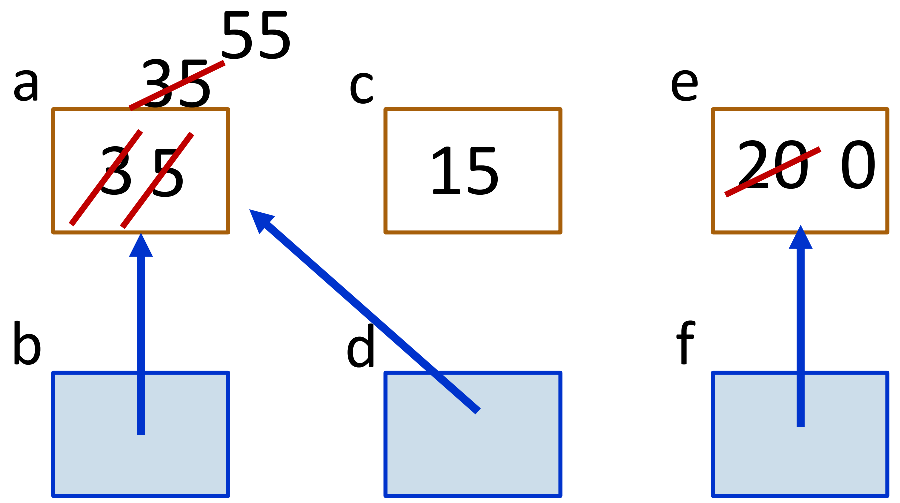
Q6. Trace the program manually and write out its output.

```
int a = 3, *b, c, *d, e, *f;

b = &a;
*b = 5;
c = *b * 3;
d = b;
e = *b + c;
*d = c + e;
f = &e;
a = *f + *b;
*f = *d - *b;

printf("a = %d, c = %d, e = %d\n", a, c, e);
printf("*b = %d, *d = %d, *f = %d\n", *b, *d, *f);
```

This does NOT make d point to b! This copies the content of b into d.



Output:

```
a = 55, c = 15, e = 0
*b = 55, *d = 55, *f = 0
```

The negation (or additive inverse) of a number x is defined to be some number y such that $x + y = 0$. Using this definition, show why the n-digit B's complement of a number x in base B is its negation. [2 marks]

The negation (or additive inverse) of a number x is defined to be some number y such that $x + y = 0$. Using this definition, show why the n-digit B's complement of a number x in base B is its negation. [2 marks]

Let x be an number in base B.
Let y be the n-digit B's complement of x. So, $y = B^n - x$.
Now consider addition in base B: $x + y = x + B^n - x = B^n$.
In base B, the rightmost n digits of B^n are 0s, so $B^n = 0$ in base B with n digits.
Therefore, $x + y = 0$ in this number system $\Rightarrow$ y is the additive inverse of x.