ELEC2041
Microprocessors and Interfacing

Lecture 6: Number Systems – I

http://webct.edtec.unsw.edu.au/

March 2005

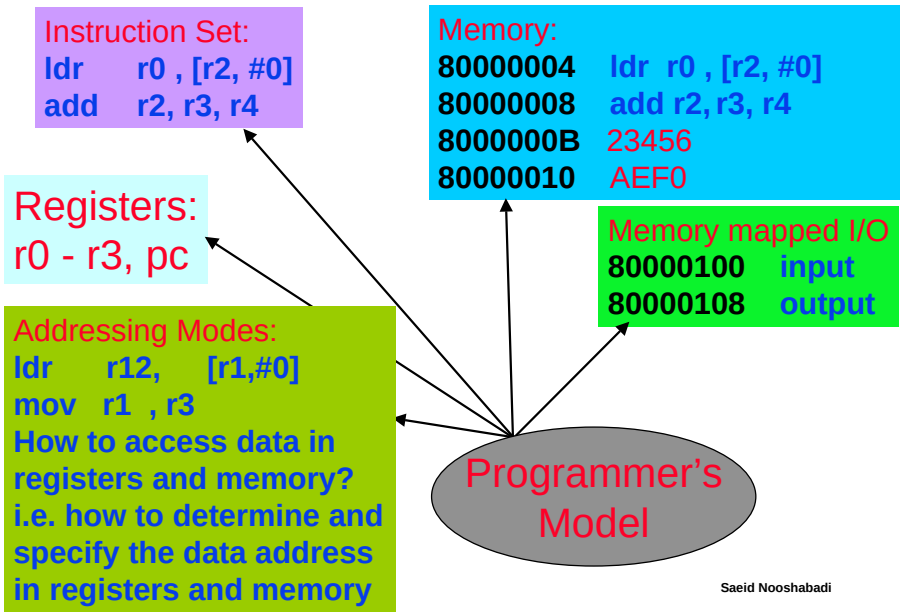
Saeid Nooshabadi

Saeid@unsw.edu.au

Overview

- Computer representation of "things"
- Unsigned Numbers
- Signed Numbers: search for a good representation
- Shortcuts
- In Conclusion

Review: The Programmer's Model of a Microcomputer



Review: Compilation

- How to turn notation programmers prefer into notation computer understands?
- Program to translate C statements into Assembly Language instructions; called a compiler
- Example: compile by hand this C code:
  a = b + c;
  d = a - e;
- Easy: add r1, r2, r3
  sub r4, r1, r6
- Big Idea: compiler translates notation from 1 level of abstraction to lower level

What do computers




- Computers manipulate representations of things!
- What can you represent with N bits?
  - 2^N things!
- Which things?
  - Numbers! Characters! Pixels! Dollars! Position! Instructions! ...
  - Depends on what operations you do on them

Decimal Numbers: Base 10

- Digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
- Example:

3271 =

$(3x10^3) + (2x10^2) + (7x10^1) + (1x10^0)$

Numbers: positional notation

- Number Base B => B symbols per digit:
  - Base 10 (Decimal): 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
    Base 2 (Binary): 0, 1
- Number representation:
  - $d_{31}d_{30}$ ... $d_2d_1d_0$ is a 32 digit number
  - value = $d_{31}x\ B^{31} + d_{30} x\ B^{30} + ... + d_2 x\ B^2 + d_1 x\ B^1 + d_0 x\ B^0$
- Binary: 0,1
  - $1011010 = 1x2^6 + 0x2^5 + 1x2^4 + 1x2^3 + 0x2^2 + 1x2 + 0x1 = 64 + 16 + 8 + 2 = 90$
  - Notice that 7 digit binary number turns into a 2 digit decimal number
  - A base that converts to binary easily?

Hexadecimal Numbers: Base 16 (#1/2)

- Digits: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
- Normal digits have expected values
- In addition:
  - A → 10
  - B → 11
  - C → 12
  - D → 13
  - E → 14
  - F → 15

Hexadecimal Numbers: Base 16 (#2/2)

° Example (convert hex to decimal):

B28F0DD = (Bx16^6) + (2x16^5) + (8x16^4) + (Fx16^3) + (0x16^2) + (Dx16^1) + (Dx16^0)

= (11x16^6) + (2x16^5) + (8x16^4) + (15x16^3) + (0x16^2) + (13x16^1) + (13x16^0)

= 187232477 decimal

° Notice that a 7 digit hex number turns out to be a 9 digit decimal number

Decimal vs. Hexadecimal vs. Binary

°Examples:

°1010 1100 0101 (binary) = ? (hex)

°10111 (binary) = 0001 0111 (binary) = ? (hex)

°3F9(hex) = ? (binary)

00	0	0000
01	1	0001
02	2	0010
03	3	0011
04	4	0100
05	5	0101
06	6	0110
07	7	0111
08	8	1000
09	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Hex to Binary Conversion

° HEX is a more compact representation of Binary!

° Each hex digit represents 16 decimal values.

° Four binary digits represent 16 decimal values.

° Therefore, each hex digit can replace four binary digits.

° Example:

0011 1011 1001 1010 1100 1010 0000 0000_{two}

3 b 9 a c a 0 0_{hex}

C uses notation 0x3b9aca00

Which Base Should We Use?

° Decimal: Great for humans; most arithmetic is done with these.

° Binary: This is what computers use, so get used to them. Become familiar with how to do basic arithmetic with them (+,-,*,/).

° Hex: Terrible for arithmetic; but if we are looking at long strings of binary numbers, it's much easier to convert them to hex in order to look at four bits at a time.

How Do We Tell the Difference?

- In general, append a subscript at the end of a number stating the base:
  - 10_{10} is in decimal
  - 10_2 is binary (= 2_{10})
  - 10_{16} is hex (= 16_{10})
- When dealing with ARM computer:
  - Hex numbers are preceded with "&" or "0x"
    - &10 == 0x10 == 10_{16} == 16_{10}
    - Note: Lab software environment only supports "0x"
  - Binary numbers are preceded with "0b"
  - Octal numbers are preceded with "0"
  - Everything else by default is Decimal

Inside the Computer

- To a computer, numbers are always in binary; all that matters is how they are printed out: binary, decimal, hex, etc.
- As a result, it doesn't matter what base a number in C is in...
  - 32_{10} == 0x20 == 100000_2
- … only the value of the number matters.

What to do with representations of numbers?

- Just what we do with numbers!
  - Add them
  - Subtract them
  - Multiply them
  - Divide them
  - Compare them

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

- Example: 10 + 7 = 17
  - so simple to add in binary that we can build circuits to do it
  - subtraction also just as you would in decimal

Addition Table

+	0	1	2	3	4	5	6	7	8	9
0	0	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9	10
2	2	3	4	5	6	7	8	9	10	11
3	3	4	5	6	7	8	9	10	11	12
4	4	5	6	7	8	9	10	11	12	13
5	5	6	7	8	9	10	11	12	13	14
6	6	7	8	9	10	11	12	13	14	15
7	7	8	9	10	11	12	13	14	15	16
8	8	9	10	11	12	13	14	15	16	17
9	9	10	11	12	13	14	15	16	17	18

Addition Table (binary)

+	0	1
0	0	1
1	1	10

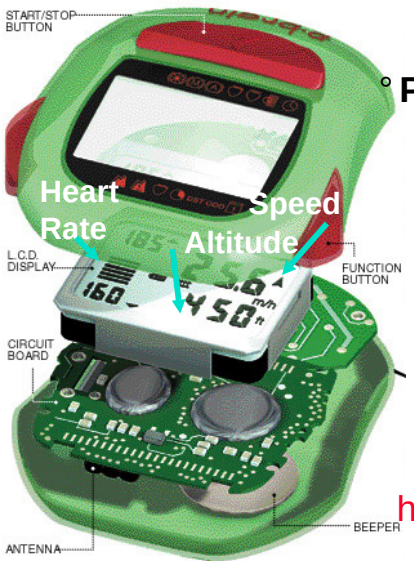
Addition Table (Hex)

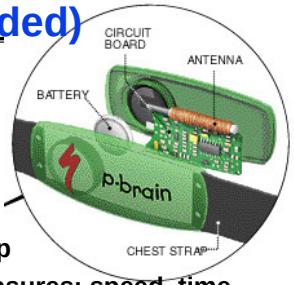
+	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10
2	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11
3	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12
4	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13
5	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14
6	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15
7	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16
8	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17
9	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18
A	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19
B	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A
C	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B
D	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C
E	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D
F	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E

Quiz # 1 Result

Title	N	% Correct Of:			Discrimination	Score	
		Whole Group	Upper 25%	Lower 25%		Mean	SD
Arrays 1	118	61	92	41	0.44	61.0%	49.0
Data Type	118	74	94	38	0.54	74.6%	43.7
Pointer to functions	118	33	51	22	0.38	33.1%	47.2
Pointers	118	86	98	58	0.54	86.4%	34.4
Pointers and addresses	118	81	98	50	0.60	81.4%	39.1
Pointer Initialisation	118	75	94	41	0.54	75.4%	43.2
					Overall Mean:	68.6%	

Bicycle Computer (Embedded)





- **P. Brain**
- **wireless heart monitor strap**
- **record 5 measures: speed, time, current distance, elevation and heart rate**
- **Every 10 to 60 sec.**
- **8KB data => 33 hours**
- **Stores information so can be uploaded through a serial port into PC to be analyzed**

http://www.specialized.com

Limits of Computer Numbers

- Bits can represent anything!
- Characters?
  - 26 letter => 5 bits
  - upper/lower case + punctuation => 7 bits (in 8) (ASCII)
  - rest of the world's languages => 16 bits (unicode)
- Logical values?
  - 0 -> False, 1 => True
- colors ?
- locations / addresses? commands?
- but N bits => only 2^N things

What if too big?

- Binary bit patterns above are simply representatives of numbers
- Numbers really have an infinite number of digits
  - with almost all being zero except for a few of the rightmost digits: e.g: 0000000 … 000098 == 98
  - Just don't normally show leading zeros
- Computers have fixed number of digits
  - In general, adding two n-bit numbers can produce an (n+1)-bit result.
  - Since computers use fixed, 32-bit integers, this is a problem.
  - If result of add (or any other arithmetic operation), cannot be represented by these rightmost hardware bits, overflow is said to have occurred

Overflow Example

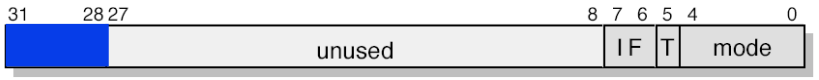
- Example (using 4-bit numbers):

+15	1111
+3	0011
+18	10010

  - But we don't have room for 5-bit solution, so the solution would be 0010, which is +2, which is wrong.

How avoid overflow, allow it sometimes?

- Some languages detect overflow (Ada), some don't (C and JAVA)
- ARM has N, Z, C and V flags to keep track overflow
  - Refer Book!
  - Will cover details later



Comparison

- How do you tell if X > Y ?
- See if X - Y > 0

➔ We need representation for both +ve and –ve numbers

How to Represent Negative Numbers?

- So far, unsigned numbers
- Obvious solution: define leftmost bit to be sign!
  - 0 => +, 1 => -
  - Rest of bits can be numerical value of number
- Representation called sign and magnitude
- ARM uses 32-bit integers. $+1_{ten}$ would be:

0000 0000 0000 0000 0000 0000 0000 0001

- And -1_{ten} in sign and magnitude would be:

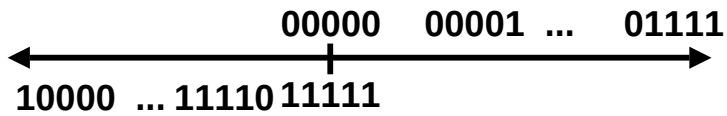
1000 0000 0000 0000 0000 0000 0000 0001

Shortcomings of sign and magnitude?

- Arithmetic circuit more complicated
  - Special steps depending whether signs are the same or not
- Also, Two zeros
  - 0x00000000 = $+0_{ten}$
  - 0x80000000 = -0_{ten}
  - What would it mean for programming?
- Sign and magnitude abandoned because another solution was better

Another try: complement the bits

- Example: $7_{10} = 00111_2$ $-7_{10} = 11000_2$
- Called one's Complement
- Note: positive numbers have leading 0s, negative numbers have leadings 1s.



- What is -00000 ?
- How many positive numbers in N bits?
- How many negative ones?

Shortcomings of ones complement?

° Arithmetic not too hard

° Still two zeros
- 0x00000000 = $+0_{ten}$
- 0xFFFFFFFF = -0_{ten}
- What would it mean for programming?

° One's complement eventually abandoned because another solution was better

Search for Negative Number Representation

° Obvious solution didn't work, find another

° What is result for unsigned numbers if tried to subtract large number from a small one?
- Would try to borrow from string of leading 0s, so result would have a string of leading 1s

3 – 7 = – 4

```
  111
 000011
-000111
 111100
```

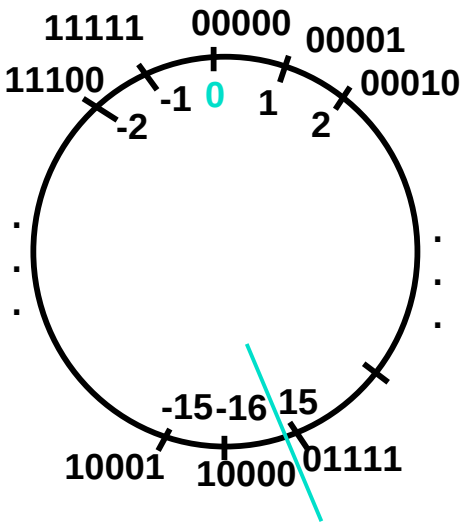
3 – 4 = – 1

```
   111
  000011
- 000100
 111111
```

- With no obvious better alternative, pick representation that made the hardware simple: leading 0s => positive, leading 1s => negative
- 000000...xxx is >=0, 111111...xxx is < 0

° This representation called two's complement

2's Complement Number line



° 2^{N-1} non-negatives

° 2^{N-1} negatives

° one zero

° how many positives?

° comparison?

° overflow?

Two's Complement

0000 ... 0000 0000 0000 0000$_{two}$ =	0_{ten}
0000 ... 0000 0000 0000 0001$_{two}$ =	1_{ten}
0000 ... 0000 0000 0000 0010$_{two}$ =	2_{ten}
. . .	
0111 ... 1111 1111 1111 1101$_{two}$ =	$2{,}147{,}483{,}645_{ten}$
0111 ... 1111 1111 1111 1110$_{two}$ =	$2{,}147{,}483{,}646_{ten}$
0111 ... 1111 1111 1111 1111$_{two}$ =	$2{,}147{,}483{,}647_{ten}$
1000 ... 0000 0000 0000 0000$_{two}$ =	$-2{,}147{,}483{,}648_{ten}$
1000 ... 0000 0000 0000 0001$_{two}$ =	$-2{,}147{,}483{,}647_{ten}$
1000 ... 0000 0000 0000 0010$_{two}$ =	$-2{,}147{,}483{,}646_{ten}$
. . .	
1111 ... 1111 1111 1111 1101$_{two}$ =	-3_{ten}
1111 ... 1111 1111 1111 1110$_{two}$ =	-2_{ten}
1111 ... 1111 1111 1111 1111$_{two}$ =	-1_{ten}

° One zero, 31st bit ➔ >=0 or <0, called sign bit
- but one negative with no positive $-2{,}147{,}483{,}648_{ten}$

Two's Complement Formula, Example

- Recognizing role of sign bit, can represent positive and negative numbers in terms of the bit value times a power of 2:
  - $d_{31} \times -2^{31} + d_{30} \times 2^{30} + \ldots + d_2 \times 2^2 + d_1 \times 2^1 + d_0 \times 2^0$
- Example

  $1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1100_{two}$

  $= 1x-2^{31} + 1x2^{30} + 1x2^{29} + \ldots + 1x2^2 + 0x2^1 + 0x2^0$

  $= -2^{31} + 2^{30} + 2^{29} + \ldots + 2^2 + 0 + 0$

  $= -2{,}147{,}483{,}648_{ten} + 2{,}147{,}483{,}644_{ten}$

  $= -4_{ten}$

Two's complement shortcut: Negation

- Invert every 0 to 1 and every 1 to 0, then add 1 to the result
  - Sum of number and its inverted representation must be ($111\ldots111_{two} = -1_{ten}$)
  - Let x′ mean the inverted representation of x
  - Then x + x′ = -1 => x + x′ + 1 = 0 => x′ + 1 = -x

```
  000011
+ 111100
--------
  111111
```

- Example: -4 to +4 to -4

x : [redacted] two
x′: [redacted] two
+1: [redacted] two
()′: [redacted] two
+1: [redacted] two

Two's complement shortcut: Negation

- Another Example: 20 to -20 to +20

x : [redacted] two
x′: [redacted] two
+1: [redacted] two
()′: [redacted] two
+1: [redacted] two

And in Conclusion...

- We represent "things" in computers as particular bit patterns: N bits =>2^N
  - numbers, characters, ... (data)
- Decimal for human calculations, binary to understand computers, hex to understand binary
- 2's complement universal in computing: cannot avoid, so learn
- Computer operations on the representation correspond to real operations on the real thing
- Overflow: numbers infinite but computers finite, so errors occur