ECE 271 – Microcomputer Architecture and Applications Lecture 11

Vince Weaver

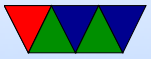
http://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

22 February 2022

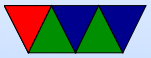
Announcements

- Read Chapter 2, Chapter 16



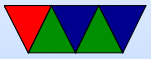
Number Representation

- Why use Base-2 in computers/digital logic?
  Why not Base-3 or Base-4? Or Base-10?
- Babbage's difference/analytical engine base-10 computer?
- Octal (useful if multiple of 3 bits), Hexadecimal (useful if multiple of 4 bits)
- Why are bytes (technically octets) 8-bits?
  Power of two, can fit a text character?
- What do you call 4-bits? (sometimes a nibble or nybble, a half-byte)



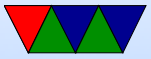
Converting Binary to Decimal

- Most straightforward algorithm is dividing by 10, convert remainder to ASCII, repeat until 0
- Somewhat complex, especially on processors without divide instruction
- Converting to hex is a lot easier (why? shift/mask)

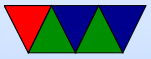


Binary Coded Decimal

- If you want decimal without having to divide
- Wastes some space
- Each byte max of 9
- BCD 0x1234 is equivalent to decimal 1234
- Some processors have special mode or instructions for handling this
- 6502 has “decimal mode” that will fix results after addition
- x86 has “aaa” (ascii adjust on addition) which will fix

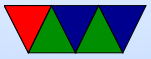


results after BCD add



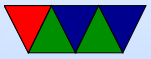
Bases In C

- Decimal numbers
- Octal have leading 0. Be careful with that! That can be surprising if not expecting it.
- Hexadecimal start with 0x
- Binary, 0b. GNU extension, probably not in Keil. Proposed for C99 but rejected.



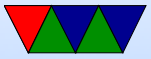
Variable Size in C

- How big is a char? (usually 8)
- How big is a short? (usually 16)
- How big is an int? (usually 32)
- How big is a long? (32 or 64 depending)
- How big is a long long? (usually 64)
- How big is a pointer? (32 or 64 depending)
- Officially each size needs to be greater than or equal to one below, but otherwise implementation defined



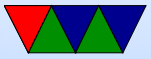
More Variable Size in C

- What if you want *exact* sizes?
  Can use `uint32_t` and similar type declarations with C99
- Note: operating system can affect too
- On 64-bit systems
  - Linux is LP64 (longs and pointers 64-bits)
  - Windows is LLP64 (longs are 32-bits, long long and pointers 64-bits)



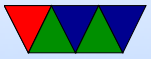
Why all the variation?

- Back when C was written there were lots of more unusual machines
- Some had 9-bit chars
- Some had 6-bit chars (RAM was expensive)
- Some pointers could not fit in an int/long
- Some NULL pointers not 0



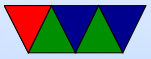
Signed vs Unsigned in C

- By default are variables signed or unsigned?
- All except char default to signed
- char is implementation dependent.



Unsigned Integers

- What's the biggest number you can represent?
  $2^N - 1$ so roughly 4 billion on 32-bit machine
- What happens if you overflow?
  Wraps to zero
- What *should* happen if you overflow?
  Is this an error? Should it be?
- What does C do if you overflow?
  Wraps to 0.
- What's the maximum size of adding two N bit unsigned

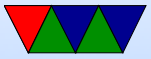


integers?

N+1 bits.

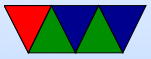
- Can you tell in C if an unsigned int overflows (by checking the carry bit or similar?)

  No. The only way to check something like `if ((a+b)<a)`



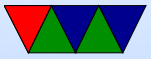
Signed Integers – Various Ways

- Sign-magnitude
  High bit is a sign bit
  Two zeros? How does that complicate things? Checking if equal?
- One's complement
  negative number is bitwise-inverse
  have to do "end-around carry" (add carry bit to rightmost bit)
- Two's complement



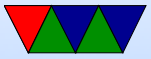
negative number is inverse, plus one

Can you have 9's complement?



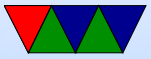
Signed Integers – C behavior

- What does C use?
  Implementation dependent (whatever the hardware uses)
- What does the hardware use?
  Most hardware these days is 2's complement
- There's a push to finally make 2's complement the default.
- What happens if you overflow a signed int on C?
  The dreaded UNDEFINED BEHAVIOR
  be careful, according to spec the compiler can do



anything it wants, including just ignoring your code

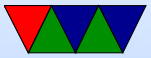
- Can you detect overflow of signed in C? No, not without a lot of annoying checks.



Binary	Sign	One's	Two's
0000	+0	+0	0
0001	1	1	1
0010	2	2	2
0011	3	3	3
0100	4	4	4
0101	5	5	5
0110	6	6	6
0111	7	7	7
1000	-0	-7	-8
1001	-1	-6	-7
1010	-2	-5	-6
1011	-3	-4	-5
1100	-4	-3	-4
1101	-5	-2	-3
1110	-6	-1	-2
1111	-7	-0	-1

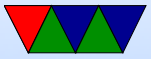
Two's complement

- Range is from $-2^{N-1} to 2^{N-1} - 1$
  only one zero
- Hardware for addition and subtraction is the same
  No need for special subtractor
- Addition/Subtraction/Multiplication of unsigned vs signed is mostly the same
- Is this only in binary? Can you do 9's complement with decimal?



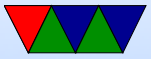
The Carry Flag

- Unsigned addition: when two unsigned integers added, carry happens when result is too big to fit in maximum integer size $(2^n - 1)$
- Unsigned subtraction: when two unsigned integers subtracted, borrow happens when result is less than 0 (ARM has no dedicated borrow flag, carry flag is re-used)



The Overflow Flag

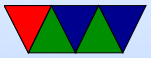
- Signed addition: when adding two positive numbers and wraps to being negative
- Signed addition: when adding two negative numbers and wraps to being positive
- Signed subtraction: sub pos from neg creates pos result
- sub neg from pos getting neg result



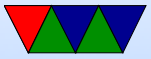
Calculating the Overflow Flag

- Overflow occurs when the carry into the sign bit *differs* from the carry out of the sign bit

```
  5       0101
 +2       0010
=====    ====== Cout=0,C=0
  7       0111  Cin=0, V=0

  5       0101
```

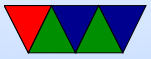


```
+ 6       0110
====    ========== Cout=0,C=0
 11 (-5) 1011     Cin=1, V=1

  9     -7         1001
+10     -6        +1010
====    ===     =======  Cout=1, C=1
 19(3) -13    (1)0011  Cin=0,  V=1

 15      -1         1111
+14      -2        +1110
```



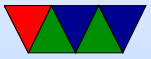
```
====     ===      ====== cout=1, C=1
 29      -3     (1)1101  Cin=1,  V=0
```

- How does the CPU know if you are doing signed vs unsigned addition?
  It doesn't. It just always sets the C and V bits.
  With two's complement it's up to you to track things if you care.
- Does the C language track the C and V bits?



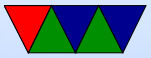
Multiplication

- Unsigned, can just do shift and add
- Signed two's complement, you can double (And sign extend) the bits then discard extra results. Inefficient.
- Can do other things, like booth multiplier



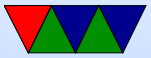
Division

- Unsigned, can just do shift and subtract (long division)
- Signed is much more complex. One thing you can do is just convert to unsigned, then adjust sign at end

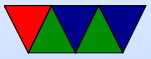


Character Encodings

- ASCII – American Standard Code for Information Interchange
  Handy that numbers are consecutive, then lower case is offset from uppercase
  Technically 7-bit. What do you do with 8-bit? Parity? Extended characters?
- EBCDIC?
- Unicode? 16-bit?
- UTF-8?

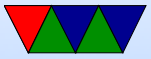


- Emojis?



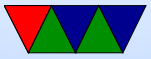
Unicode

- Can we fit every writing style ever?
- Can it all fit in 16-bit?
- wchar_t – Java/Windows when 16-bit was thought to be enough
  now hold UTF-16



UTF-8

- Original ASCII is the same, and compact
-



Unicode issues

- Right-to-left / left-to right text
- Characters that look the same
- Country flags
- Emoji issues