COMP 3221

Microprocessors and Embedded Systems

Lecture 7: Number Systems - III

http://www.cse.unsw.edu.au/~cs3221

August, 2003

Saeid Nooshabadi

Saeid@unsw.edu.au

- **Condition Code Flag interpretation**
- **Characters and Strings**
- **In Conclusion**

Review: `int` and `unsigned int` in C

- **With `N` bits we can represent 2^N different Numbers:**
  - 2^N numbers 0 to $2^N - 1$:Only zero and Positive numbers
  - 2^N numbers $-2^N/2$ to 0 to $2^N/2 - 1$: Both Negative and positive numbers in 2's Complement

0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	-8
1001	9	-7
1010	10	-6
1011	11	-5
1100	12	-4
1101	13	-3
1110	14	-2

Is `1000 > 0110` ?

`1000 > 0110` if only +ve representation used

`1000 < 0110` if both +ve and -ve representation in 2's complement used

Review: Condition Flags

Flags	Arithmetic Instruction
Negative (N='1')	Bit 31 of the result has been set Indicates a negative number in signed operations
Zero (Z='1')	Result of operation was zero
Carry (C='1')	Result was greater than 32 bits
oVerflow (V='1')	Result was greater than 31 bits Indicates a possible corruption of the sign bit in signed numbers

31 28 27 8 7 6 5 4 0

N Z C V | unused | I F | T | mode

Experimentation with Condition Flags (#1/4)

Indicate the changes in N, Z, C, V flags for the following arithmetic operations: (Assume 4 bit-numbers)

NOTE: V = MSB Carry In (XOR) MSB Carry out

```
 1110
 1111 +
 0010
10001
```

= → No Overflow, V = 0

= 1 → Carry set, C = 1

= 0 → Result +Ve N = 0

Result Not Zero Z = 0

Signed interpretation: -1 + 2 = 1. The number is within the range of –8 to +7. No oVerflow (V), Ignore Carry out.

Unsigned interpretation: 15 + 2 = 17. The number is out of the range of 0 to +15. Carry Set and oVerflow Not set. Indication for overflow in unsigned.

Experimentation with Condition Flags (#2/4)

Indicate the changes in N, Z, C, V flags for the following arithmetic operations: (Assume 4 bit-numbers)

NOTE: V = MSB Carry In (XOR) MSB Carry out

```
 0110
 0111 +
 0010
01001
```

≠ → Overflow, V = 1

= 0 → Carry set, C = 0

= 1 → Result -Ve N = 1

Result Not Zero Z = 0

Signed interpretation: 7 + 2 = 9. The number is out of the range of –8 to +7. oVerflow (V), Ignore Carry out.

Unsigned interpretation: 7 + 2 = 9. The number is within the range of 0 to +15. Carry Not set and oVerflow Set. Indication for No overflow in unsigned.

Experimentation with Condition Flags (#3/4)

Indicate the changes in N, Z, C, V flags for the following arithmetic operations: (Assume 4 bit-numbers)

```
1110 -        1000
0111    →     1110 +
              1001
             10111
```

≠ → OVerflow, V = 1

= 1 → Carry set, C = 1

= 0 → Result +Ve N = 0

Result Not Zero Z = 0

Signed interpretation: -2 - 7 = = -2 + (-7) =-9. The number is out of the range of –8 to +7. oVerflow (V), Ignore Carry out.

Unsigned interpretation: 14 - 7 = 7. The number is in of the range of 0 to +15. Carry Set and oVerflow Set. Indication for No overflow

`Experimentation with Condition Flags (#4/4)

Indicate the changes in N, Z, C, V flags for the following arithmetic operations: (Assume 4 bit-numbers)

```
 0010
 0011 +
 0010
00101
```

= → Overflow, V = 0

= 0 → Carry set, C = 0

= 0 → Result +Ve N = 0

Result Not Zero Z = 0

Signed interpretation: 3 + 2 = 5. The number is within of the range of –8 to +7. No oVerflow (V), Ignore Carry out.

Unsigned interpretation: 3 + 2 = 5. The number is within the range of 0 to +15. Carry Not set and oVerflow Not set. Indication for No

Signed Arithmetic overflow Condition:

oVerflow flag V = 0 NO OVERFLOW

oVerflow flag V = 1 OVERFLOW

NOTE: V = MSB Carry In (XOR) MSB Carry out

UnSigned Arithmetic overflow Condition:

Oveflow:

(oVerflow flag V = 0) AND (Carry flag C = 0) NO OVERFLOW

(oVerflow flag V = 0) AND (Carry flag C = 1) OVERFLOW

(oVerflow flag V = 1) AND (Carry flag C = 0) NO OVERFLOW

(oVerflow flag V = 1) AND (Carry flag C = 1) NO OVERFLOW

- Consider:

1111 = -1 in 4-bit representation

1111 1111 = -1 in 8-bit representation

1111 1111 1111 1111 = -1 in 16-bit representation

2's comp. negative number has infinite 1s

0111 = 7 in 4-bit representation

0000 0111 = 7 in 8-bit representation

0000 0000 0000 0111 = 7 in 16-bit representation

• 2's comp. positive number has infinite 0s

Bit representation hides leading bits

Two's comp. shortcut: Sign extension

- Convert 2's complement number using n bits to more than n bits
- Simply replicate the most significant bit (sign bit) of smaller to fill new bits
  - •2's comp. positive number has infinite 0s
  - •2's comp. negative number has infinite 1s
  - •Bit representation hides leading bits; sign extension restores some of them
  - •16-bit -4_{ten} to 32-bit:

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

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

Beyond Integers (Characters)

- 8-bit bytes represent characters, nearly every computer uses American Standard Code for Information Interchange (ASCII)

No.	char	No.	char	No.	char	No.	char	No.	char	No.	char
32		48	0	64	@	80	P	96	`	112	p
33	!	49	1	65	A	81	Q	97	a	113	q
34	"	50	2	66	B	82	R	98	b	114	r
35	#	51	3	67	C	83	S	99	c	115	s
...		...		...		...		...		...	
47	/	63	?	79	O	95	_	111	o	127	DEL

- Uppercase + 32 = Lowercase (e.g, B+32=b)
- tab=9, carriage return=13, backspace=8, Null=0

- Characters normally combined into strings, which have variable length
  - e.g., “Cal”, “M.A.D”, “COMP3221”
- How represent a variable length string?
  1) 1st position of string reserved for length of string (Pascal)
  2) an accompanying variable has the length of string (as in a structure)
  3) last position of string is indicated by a character used to mark end of string (C)
- C uses 0 (Null in ASCII) to mark end of string

- How many bytes to represent string “Popa”?
- What are values of the bytes for “Popa”?

No.	char	No.	char	No.	char	No.	char	No.	char	No.	char
32		48	0	64	@	80	P	96	`	112	p
33	!	49	1	65	A	81	Q	97	a	113	q
34	"	50	2	66	B	82	R	98	b	114	r
35	#	51	3	67	C	83	S	99	c	115	s
...		...		...		...		...		...	
47	/	63	?	79	O	95	_	111	o	127	DEL

- 80, 111, 112, 97, 0 DEC
- 50, 6F, 70, 61, 0 HEX

Strings in C: Example

- String simply an array of char

```
void strcpy (char x[], char y[]){
int i = 0; /* declare,initialize i*/

while ((x[i] = y[i]) != '\0') /* 0 */
 i = i + 1; /* copy and test byte */
}
```

What about non-Roman Alphabet?

- Unicode, universal encoding of the characters of most human languages
  - Java uses Unicode
  - needs 16 bits to represent a character
  - 16-bits called half word in ARM

- Why not ASCII computers vs. binary computers?
  - Harder to build hardware for add, subtract, multiply, divide
  - Memory space to store numbers
- How many bytes to represent 1 billion?
- ASCII: “1000000000” => 11 bytes
- **Binary:** 0011 1011 1001 1010 1000 0000 0000 0000 => 4 bytes
- up to 11/4 or almost 3X expansion of data size

- Numbers, Characters, logicals, ...
- Addresses
- Commands (operations)
  - example:
    - 0 => clap your hands
    - 1 => snap your fingers
    - 2 => slap your hands down
  - execute: 1 0 2 0 1 0 2 0 1 0 2 0 1 0 2 0
  - another example
    - 0 => add
    - 1 => subtract
    - 2 => compare
    - 3 => multiply

How can we represent a machine instruction?

- Some bits for the operation
- Some bits for the address of each operand
- Some bits for the address of the result

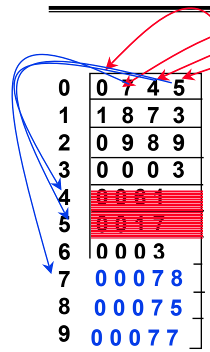
N-1 ... 0

operation	result addr	op1 addr	op2 addr

d = x + y   add d x y

- Where could we put these things called instructions?

The Stored Program Computer



- Memory holds instructions and data as bits
- Instructions are fetched from memory and executed
  - operands fetched, manipulated, and stored
- Example 4-digit Instruction

operation	result addr	op1 addr	op2 addr

  - result address
  - op1 address
  - op2 address

0 => add
1 => subtract
2 => compare
3 => multiply

- Example Data
  - 4 digit unsigned value
- What’s in memory after executing 0,1,2?

- We can write a program that will translate strings of 'characters' into 'computer instructions'
  - called a compiler or an assembler
- We can load these particular bits into the computer and execute them.
  - may manipulate numbers, characters, pixels... (application)
  - may translate strings to instructions (compiler)
  - may load and run more programs (operating system)

Saeid Nooshabadi

- We represent "things" in computers as particular bit patterns
  - numbers, characters, ... (data)
    - base, digits, positional notation
    - unsigned, 2s complement, 1s complement
  - addresses (where to find it)
  - instructions (what to do)
- Computer operations on the representation correspond to real operations on the real thing
  - representation of 2 plus representation of 3 = representation of 5
- two big ideas already!
  - Pliable Data: a program determines what it is
  - Stored program concept: instructions are just data

Saeid Nooshabadi

And in Conclusion...

- 2's complement universal in computing: cannot avoid, so learn
- Overflow: numbers infinite but computers finite, so errors occur
- Computers provide help to detect overflow
- Condition code flags N, Z, C and V provide help to deal with arithmetic computation and interpretation in signed and unsigned representation.