COMP-162
Embedded Systems

Lecture 3: Introduction to C programming

Agenda

- Recap
  - GPIO
  - Logic and Shift Operations
  - Addressing Modes
  - Subroutines and the Stack
- Outline
  - Variables in C (types, local/global)
  - Functions (parameters, prototypes)
  - for-loop and while-loop
  - if-then and if-then-else
  - Arrays with indexed access
    - RAM/ROM and initialization

Variables

- Type
  - int32_t
  - uint32_t
  - int16_t
  - uint16_t
  - int8_t
  - uint8_t
  - char
- Scope
  - Global -> everywhere
  - Local -> within {}
- Allocation
  - Global -> ROM or RAM
  - Local -> registers or stack

- 32-bit access
  - LDR
  - STR
- 16-bit access
  - LDRH LDRSH
  - STRH
- 8-bit access
  - LDRB LDRSB
  - STRB
- Naming style
  - Globals (capital)
  - Locals (lower)

Expressions

- Must operate on similar types

  `variable = value;`
- Arithmetic

  `+   -   *   /   %`
- Logical

  `&   |   ^   ~   >>   <<`
- Relational (two values to boolean)

  `<   <=   >   >=   ==   !=`
- Boolean (false is 0, true is nonzero)

  `&&   ||   !`

Conditionals

- if-then

```
if(A>M){
  M = A;
}
if((letter>='A')&&(letter<='Z')){
  letter = letter+('a'-'A');
}
```

- if-then-else

```
if(A>B){
  M = A;
}else{
  M = B;
}
```

```
uint16_t Height;
int main(void){
  if(Height > -5){
    Height = 0;
  }
}
```

Loops

- for-loop

```
m = 0;
for(i=0; i<10; i++){
  m = m+i;
}
```

- while-loop

```
int main(void){uint32_t m;
  while(1){
    m = 1000000;
    while(m){
      m--;
    }
   // stuff
  }
}
```

```
while(GPIO_PORTF_DATA_R&0x10){
}
```

Functions

- Prototype/declaration

```
uint32_t max(uint32_t a, uint32_t b);
uint32_t CardiacOutput(uint32_t t[1000]);
int32_t dot(int32_t a[], int32_t b[],int32_t l)
```

- Definition

```
uint32_t max(uint32_t a, uint32_t b){uint32_t r;
  if(a>b){
    r = a;
  }else{
    r = b;
  }
  return r;
}
```

- Invocation

```
c = max(x+3;y);
```

Example Not Gate

Should be local, made global to help debugging

```
uint32_t In,Out;
void Not_Init(void);
int main(void){
  Not_Init();
  while(1){ // operations to be executed over and over go here
    In = GPIO_PORTE_DATA_R & 0x01;
    Out = ~In;
    GPIO_PORTE_DATA_R = (GPIO_PORTE_DATA_R&~0x02)|Out<<1;
  }
}
void Not_Init(void){
volatile uint32_t delay;
  SYSCTL_RCGCGPIO_R |= 0x10;   // Turn clock on PortE
  delay = 100;                 // Wait
  GPIO_PORTE_DIR_R |= 0x02;    // PE1 is output
  GPIO_PORTE_DIR_R &= ~(0x01); // PE0 is input
  GPIO_PORTE_DEN_R |= 0x03;
}
```

Memory segments

Code (Flash EEPROM)

0x00000000 Initial stack

0x00000004 Initial PC

…

0x00000200 Your code

...

0x0003FFFC

Data

0x20000000 Your globals

0x20000004...

Stack

0x20000402

0x20000404

0x20000408   SP-> top

Heap

Call by value versus reference

```c
void noChange(uint32_t val){
    val = 5;
}
void Change(uint32_t *val){
   *val = 5;
}
uint32_t a;
int main(void){
  a = 55;
  noChange(a);
  Change(&a);
}
```

```
noChange
    MOV R0,#5
   BX LR
Change
   MOV R1,# 5
   STR R1,[R0]
   BX LR
main
   LDR R0,=a
   MOV R1,#55
   STR  R1,[R0]
   LDR R0,=a
   BL   noChange
   LDR R0,=a
   BL   Change
   BX LR
```

Pointers

```
void swap(uint32_t *a,uint32_t *b){
    uint32_t t;
    t = *a;
   *a = *b;
   *b = t;
}
uint32_t a,b;
int main(void){
  a = 3; b=4;
  swap(&a,&b);
}
```

```
        AREA    DATA, ALIGN=2
a       SPACE 4
b       SPACE 4
```

```
swap
   LDR R3,[R0] ;t=*a
   LDR R4,[R1] ;*b
   STR R4,[R0] ;*a =*b
   STR R3,[R1]
   BX LR
```

```
main
   LDR R0,=a
   MOV R1,#3
   STR  R1,[R0]
   LDR R0,=b
   MOV R1,#5
   STR  R1,[R0]
   LDR R0,=a
   LDR R1,=b
   BL   swap

   BX LR
```

Arrays

- Definition (type, size, allocation

```
#define LEN 10
int16_t Data[10];  // global RAM
const int16_t Prime[5]={2,3,5,7,11};  // global ROM
void fun(void){ char name[8];
}
```

- Access

```
Data[0] = 55;
Data[1] = 72;
```

- Zero index address calculation Buf[i]

```
32 bit:  Buf+4*i
16 bit:  Buf+2*i
8 bit:  Buf+i
```

Array example

- Definition (type, size, allocation)

```
#define LEN 10
int32_t aa[LEN];
int32_t bb[LEN];
```

- Access

```
int32_t s=0;
for(int32_t i=0;i<l;i++){
    s += a[i]*b[i];
}
```

- Review: what is?

```
aa[0]
&aa[0]
aa
aa[i]
```

Array example

- Address calculation

  64-bit base+8*index
  32-bit base+4*index
  16-bit base+2*index
  8-bit base+index

- Access

```
for(int i=0; i< 5;i++){
    aa[i] = i;
    bb[i] = 5;
}
```

```
        AREA    DATA, ALIGN=2
aa   SPACE 4*10
bb   SPACE 4*10
i RN 4
main MOV i,#0  ;i=0
     MOV R3,#5
forloop2
     CMP i,#5  ;is i<5
     BGE forDone2
     LDR R0,=aa  ;aa[i] = aa+4*i
     ASL R2,i,#2 ;R2=i*4
     STR i,[R0,R2]
     LDR R6,=bb
     STR R3,[R6,R2] ;bb+i*4
    ADD i,i,#1
    B  forloop2
forDone2
```

Array parameters

- Parameter is pass by reference

```
int32_t dot(int32_t a[], int32_t b[], int32_t l){
  int32_t s=0;
  for(int32_t i=0;i<l;i++){
     s += a[i]*b[i];
  }
  return s;
}
```

- Invocation pass by reference

```
int main(void){
  int32_t result;
  result = dot(aa,bb,5);
  while(1){
  }
}
```

```
s RN 4
i  RN 5
dot MOV s,#0  ;s=0
    MOV i,#0  ;i=0
LDR R0,=aa  ;aa[i] = aa+4*i
LDR R1,=bb  ;bb[i] = bb+4*i

forloop3
    CMP i,R2  ;is i<l
    BGE forDone3
    ASL R6,i,#2 ;i*4
    LDR  R7,[R0,R6]
    LDR  R8,[R1,R6]
    MUL R7,R7,R8
   ADD s,s,R7
    ADD i,i,#1
   B  forloop3
forDone3
   MOV R0,s
   BX  LR
```

Summary

- ❑ Variables (type, size, allocation)
- ❑ Expressions
- ❑ Conditionals
- ❑ If-then and if-then-else
- ❑ For-loop
- ❑ While-loop
- ❑ Functions
- ❑ Arrays

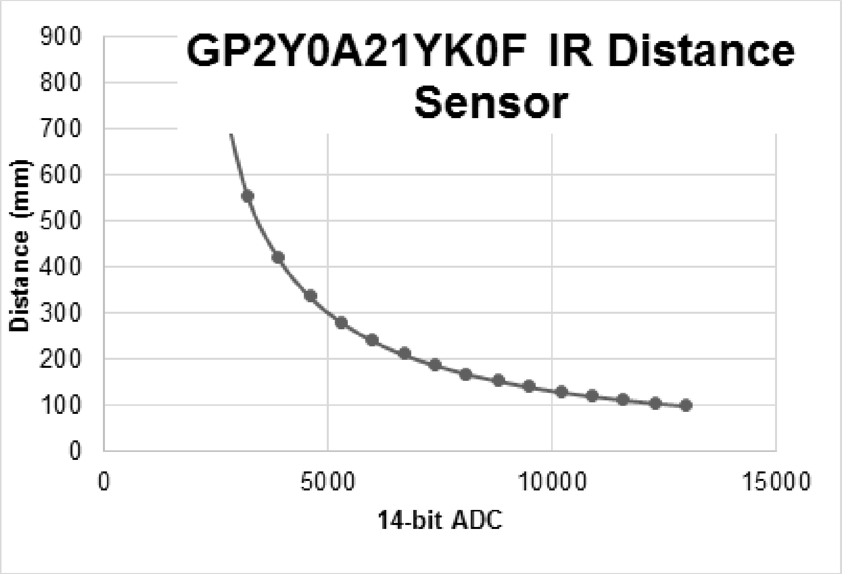
Call by value vs call by reference

```
void swap(int32_t aa, int32_t bb){ int32_t tmp;
    tmp = aa;
    aa = bb;
    bb = tmp;
}
void swap2(int32_t *aa, int32_t *bb){ int32_t tmp;
    tmp = *aa;
    *aa = *bb;
    *bb = tmp;
}
int32_t a=33;
int32_t b=44;
int main(void){
  Output_Init(); // initialize output device
  swap(a,b);
  printf("swap a=%d, b=%d\n",a,b);
  swap2(&a,&b);
  printf("swap2 a=%d, b=%d\n",a,b);
  while(1){};
}
```

IR distance sensor

```
#define K1 1195172
#define K2 -1058

uint32_t IRconvert(uint32_t n){
  return  = K1/(n + K2);
}
uint32_t IRconvert(uint32_t n){
uint32_t d;
  if(n<3000) return 800;
  if(n>13000) return 100;
  d = K1/(n + K2);
  return d;
}
```



d = 1195172/(n – 1058)

Maximum of an array

```
int32_t max(int32_t data[],
           uint32_t size){
int32_t ans,i;
  ans = -2147483648; // smallest possible
  for(i=0; i < size; i++){
    if(data[i] > ans){
      ans = data [i];
    }
  }
  return ans;
}
```

```
#define SIZE 10
int32_t MyData[SIZE];
int main(){int32_t myMax;
  while(1){
// fill up MyData
     myMax = max(MyData,SIZE);
  }
}
```

Dot product

```
#define LEN 5
int32_t aa[LEN];
int32_t bb[LEN];
int32_t dot(int32_t a[], int32_t b[],
   int32_t length){ int32_t s=0;
  for(int32_t i=0;i< length;i++){
   s += a[i]*b[i];
  }
  return s;
}
```

```
int main(void){
  int32_t result;
  for(int i=0; i< LEN;i++){
    aa[i] = i;
    bb[i] = 5;
  }
  result = dot(aa,bb,LEN);
  while(1){ }
}
```

Capitalize letters in string

```
char name[10] = "Jonathan";
// uncapitalize every letter
void uncap(char str[]){int i=0;
  while(str[i]){
    str[i] |= 0x40;
     i++;
  }
}
```

```
void uncap(char *p){
  while(*p){
    *p |= 0x40;
     p++;
  }
}
```

Agenda

- Recap
  - GPIO
  - Logic and Shift Operations
  - Addressing Modes
  - Subroutines and the Stack
  - Introduction to C
- Outline
  - Debugging
  - Digital Logic
    - GPIO TM4C123/LM4F120 Specifics
  - Switch and LED interfacing
  - Arithmetic Operations
    - Random Number Generator example

Debugging

- Testing, Diagnostics, Verification, Validation
- Debugging Actions
  - Functional debugging, input/output values
  - Performance debugging, input/output values with time (how fast does it execute)
  - Resource debugging, I/O values w/ time and resources (how much memory, power, …)
  - Tracing, measure sequence of operations
  - Profiling,
    - measure percentage for tasks,
    - time relationship between tasks
  - Optimization, make tradeoffs for overall good
    - improve speed,
    - improve accuracy,
    - reduce memory,
    - reduce power,
    - reduce size,
    - reduce cost

Debugging Intrusiveness

- Intrusive Debugging
  - degree of perturbation caused by the debugging itself
  - how much the debugging slows down execution
- Non-intrusive Debugging
  - characteristic or quality of a debugger
  - allows system to operate as if debugger did not exist
  - e.g., logic analyzer, ICE, JTAG

- Minimally intrusive
  - negligible effect on the system being debugged
  - e.g., dumps(ScanPoint) and monitors
- Highly intrusive
  - print statements, breakpoints and single-stepping

… Debugging

- Instrumentation: Code we add to the system that aids in debugging
  - E.g., print statements
  - Good practice: Define instruments with specific pattern in their names
  - Use instruments that test a run time global flag
    - leaves a permanent copy of the debugging code
    - causing it to suffer a runtime overhead
    - simplifies “on-site” customer support.
  - Use conditional compilation (or conditional assembly)
    - Keil supports conditional assembly
    - Easy to remove all instruments
    - IF symbol / ELSE / ENDIF; --predefine “symbol SETL {TRUE}” in ASM options
    - #ifdef / #else #endif
- Visualization: How the debugging information is displayed

Debugging Aids in Keil

Interface

- Breakpoints
- Registers including xPSR
- Memory and Watch Windows
- Logic Analyzer, GPIO Panel
- Single Step, StepOver, StepOut, Run, Run to Cursor
- Watching Variables in Assembly

```
EXPORT  VarName[DATA,SIZE=4]
```

- Command Interface (Advanced but useful)

```
WS 1, `VarName,0x10
LA (PORTD & 0x02)>>1
```

ARM ISA : ADD, SUB and CMP

ARITHMETIC INSTRUCTIONS

```
ADD{S} {Rd,} Rn, <op2>  ;Rd = Rn + op2
ADD{S} {Rd,} Rn, #im12  ;Rd = Rn + im12
SUB{S} {Rd,} Rn, <op2>     ;Rd = Rn - op2
SUB{S} {Rd,} Rn, #im12      ;Rd = Rn - im12
RSB{S} {Rd,} Rn, <op2>      ;Rd = op2 - Rn
RSB{S} {Rd,} Rn, #im12  ;Rd = im12 - Rn
CMP Rn, <op2>        ;Rn - op2
CMN Rn, <op2>        ;Rn - (-op2)
```

Addition
C bit set if unsigned overflow
V bit set if signed overflow

Subtraction
C bit *clear* if unsigned overflow
V bit set if signed overflow

ARM ISA : Multiply and Divide

32-BIT MULTIPLY/DIVIDE INSTRUCTIONS

```
MUL{S} {Rd,} Rn, Rm      ;Rd = Rn * Rm
MLA     Rd, Rn, Rm, Ra   ;Rd = Ra + Rn*Rm
MLS     Rd, Rn, Rm, Ra   ;Rd = Ra - Rn*Rm
UDIV   {Rd,} Rn, Rm      ;Rd = Rn/Rm unsigned
SDIV   {Rd,} Rn, Rm      ;Rd = Rn/Rm signed
```

Multiplication does not set C,V bits

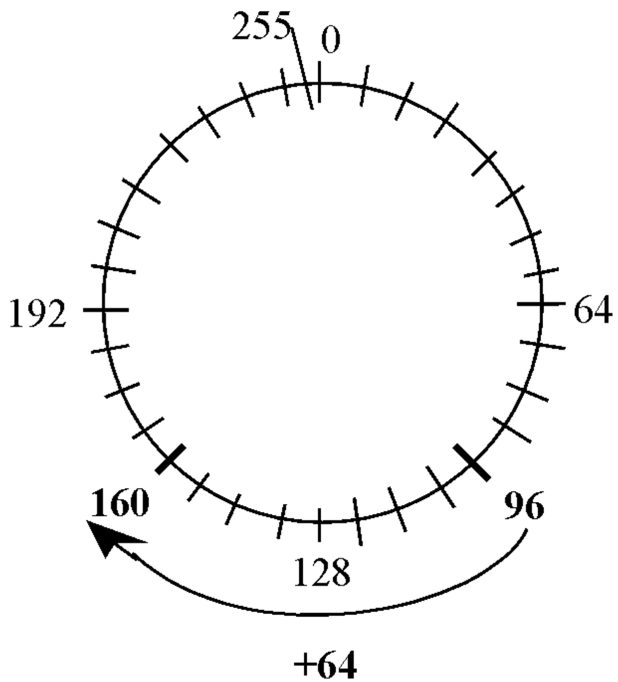
Condition Codes

Bit	**Name**	**Meaning after add or sub**
N	negative	result is negative
Z	zero	result is zero
V	overflow	signed overflow
C	carry	unsigned overflow

- C set after an **unsigned** addition if the answer is wrong
- C cleared after an **unsigned** subtract if the answer is wrong
- V set after a **signed** addition or subtraction if the answer is wrong

8-bit unsigned number wheel

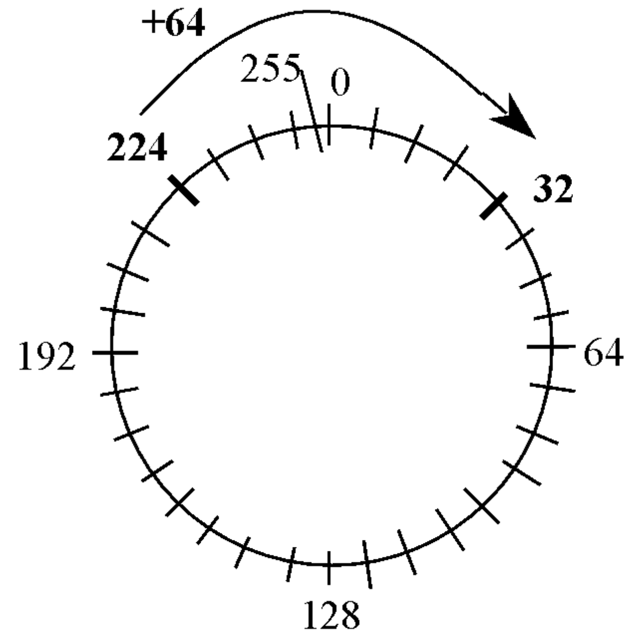
96+64



C bit

<u>Cleared</u>

224+64

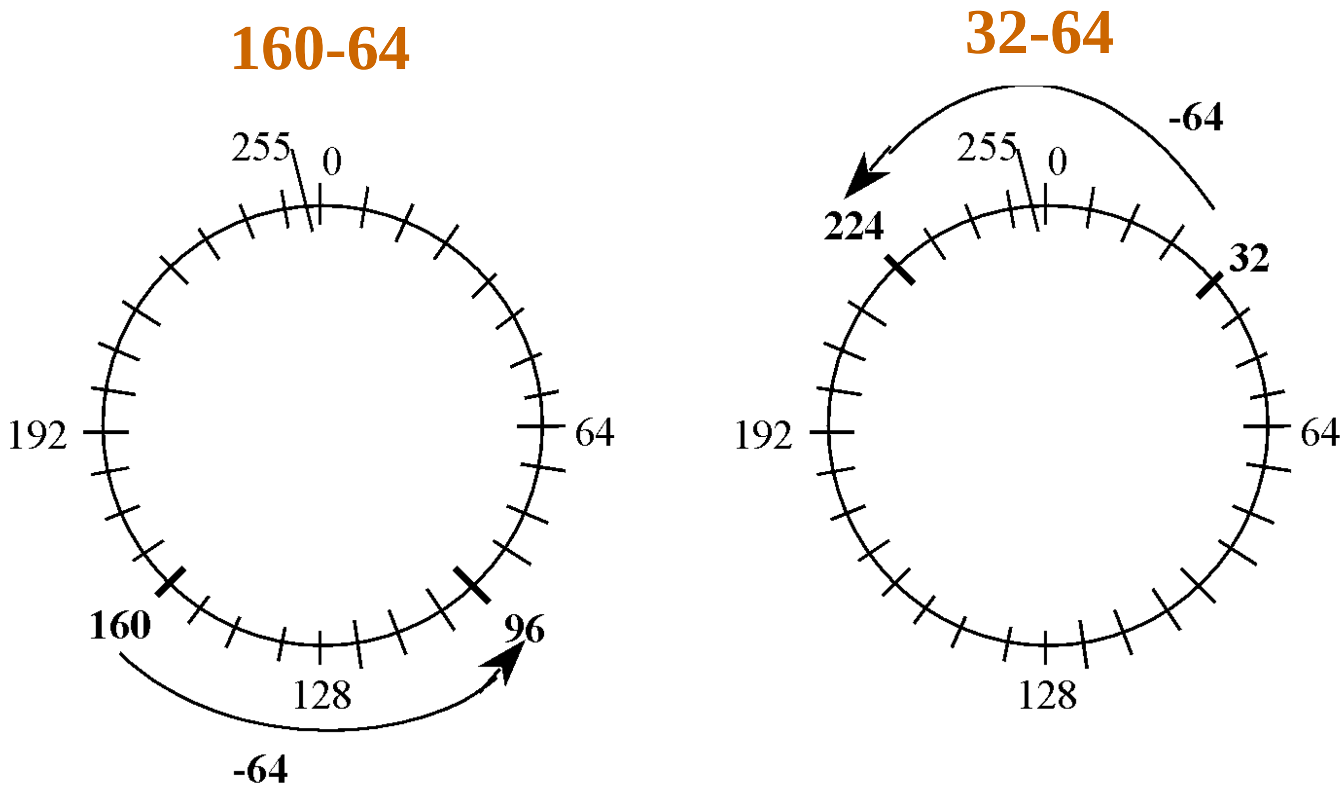


C bit

<u>Set</u>

- The carry bit, C, is set after an unsigned addition when the result is incorrect.
- The carry bit, C, is clear after an unsigned subtraction when the result is incorrect.

8-bit unsigned number wheel



C bit Set

C bit Cleared

- The carry bit, C, is set after an unsigned addition when the result is incorrect.
- The carry bit, C, is clear after an unsigned subtraction when the result is incorrect.

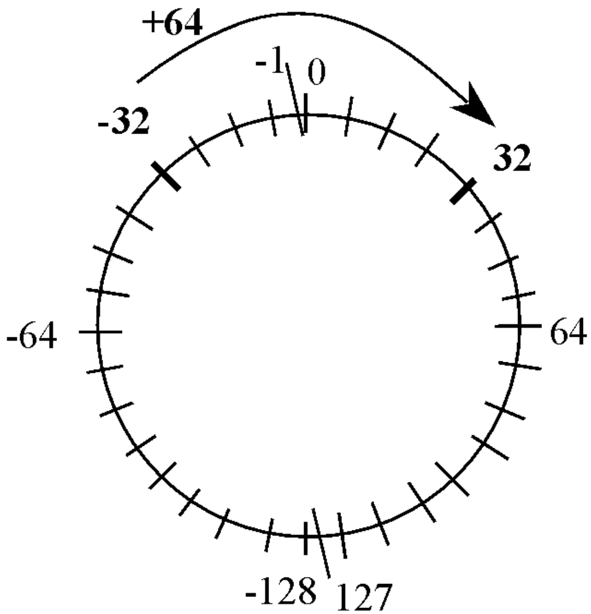
Algorithm (unsigned)

1. **Find values of both numbers interpreted as unsigned**
2. **Perform addition or subtraction**
3. **Does the result fit as an unsigned?**
   - **No -> addition C=1,**
     subtraction C=0
   - **Yes -> addition C=0,**
     subtraction C=1

For example: 255 + 5 = 260, C = 1 and the actual answer is 260-256 = 4

8-bit signed number wheel

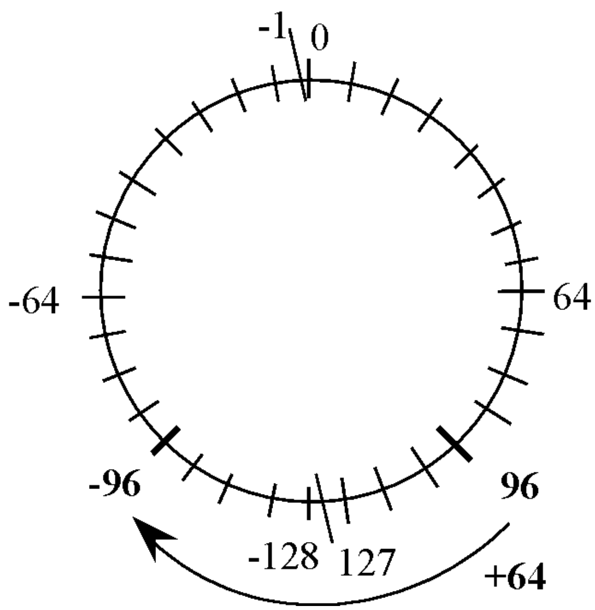
-32+64



V bit

<u>Cleared</u>

96+64



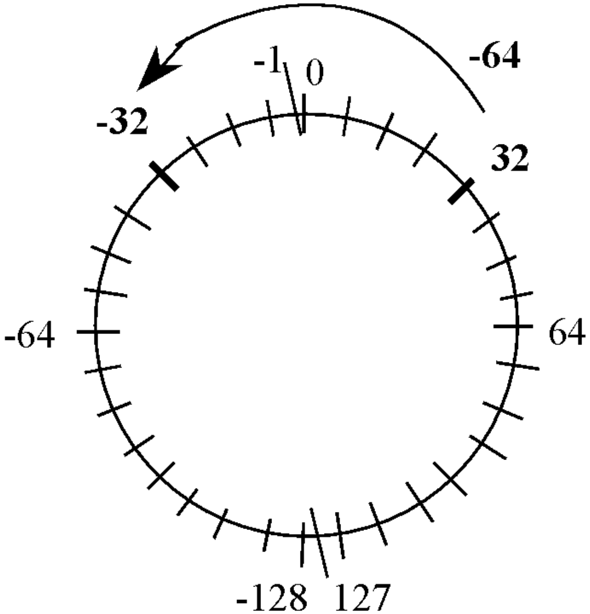
V bit

<u>Set</u>

- The overflow bit, V, is set after a signed addition or subtraction when the result is incorrect.

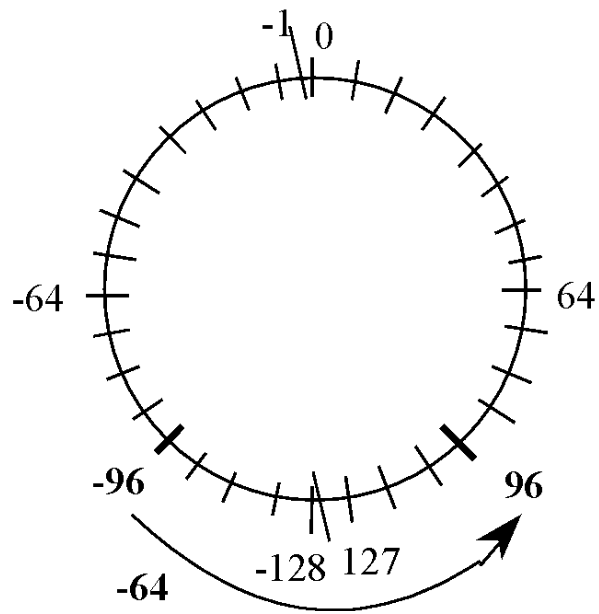
8-bit signed number wheel

32-64



V bit
<u>Cleared</u>

-96-64



V bit
<u>Set</u>

- The overflow bit, V, is normally set when we cross over from 127 to -128 while adding or cross over from -128 to 127 while subtracting.

Algorithm (signed)

1. **Find values of both numbers interpreted as signed**
2. **Perform addition or subtraction**
3. **Does the result fit as a signed?**
   - **No -> V=1**
   - **Yes -> V=0**

8-bit

Examples: 10 – 5 = 5, V=0

-100 – 100 = -200, V=1

Addition Summary

Let the 32-bit result R be the result of the 32-bit addition X+M.

- **N bit** is set
  - if unsigned result is above 2^{31}-1 or
  - if signed result is negative.
  - $N = R_{31}$
- **Z bit** is set if result is zero
- **V bit** is set after a signed addition if result is incorrect
  -
- **C bit** is set after an unsigned addition if result is incorrect
  -

Subtraction Summary

Let the 32-bit result R be the result of the 32-bit subtraction X-M.

- **N bit** is set
  - if unsigned result is above 2^{31}-1 or
  - if signed result is negative.
  - $N = R_{31}$
- **Z bit** is set if result is zero
- **V bit** is set after a signed subtraction if result is incorrect
  -
- **C bit** is clear after an unsigned subtraction if result is incorrect
  -

Trick Question

- **When the subtraction (32 – 129) is performed in an 8-bit system what is the result and the status of the NZVC bits?**

Trick Question

- **When the subtraction (32 – 129) is performed in an 8-bit system what is the result and the status of the NZVC bits?**

- **Answer = 159**
- **NZVC = 1010**