Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

04 – Arithmetic

CS1021 – Introduction to Computing I

Dr Jonathan Dukes | jdukes@tcd.ie
School of Computer Science and Statistics

Binary Arithmetic – Addition

Binary	Decimal equivalent
0 0 0 0 0 1 1 0	6
0 0 0 0 1 0 1 1 +	11 +
0 0 0 1 0 0 0 1	17

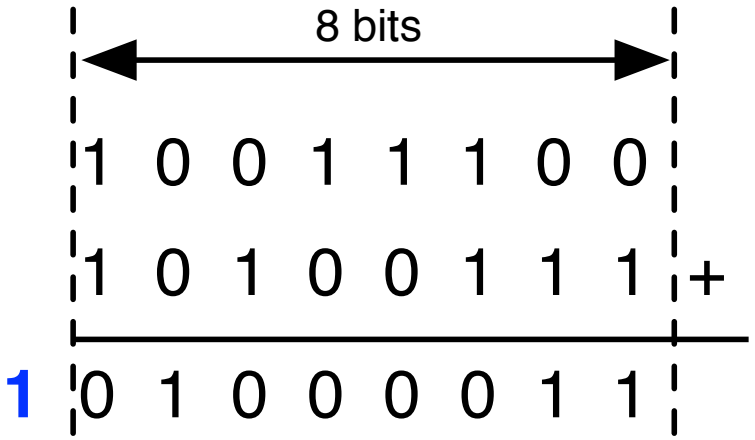
0 0 0 1 0 1 1 0	22
0 0 0 0 1 0 1 1 +	11 +
0 0 1 0 0 0 0 1	33

jdukes@tcd.ie

Carry

What happens if we run out of digits?

Adding two numbers each stored in 1 byte (8 bits) may produce a 9-bit result



Decimal equivalent

156

167 +

323

Added 156_{10} + 167_{10} and expected to get 323_{10}

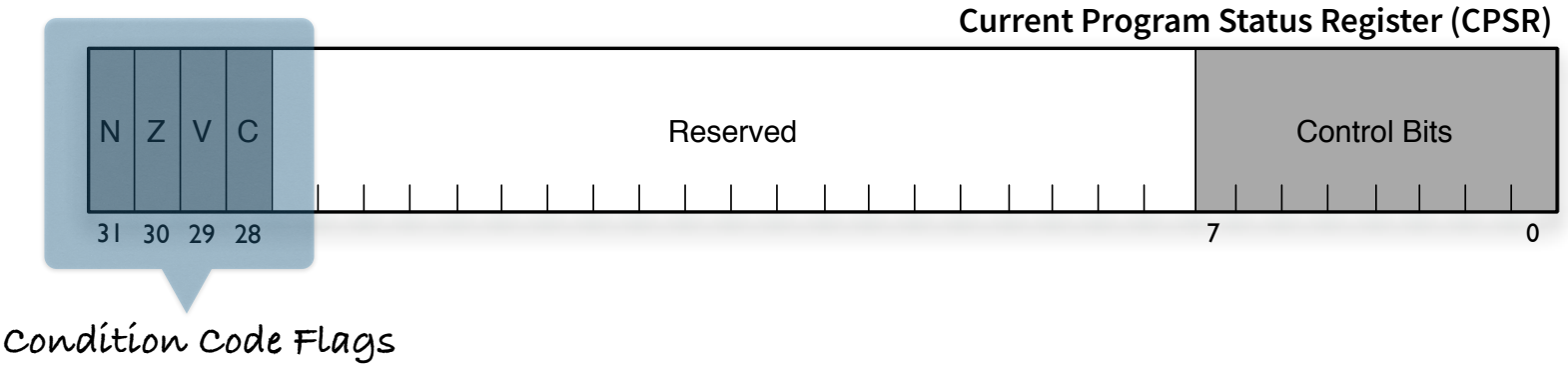
8-bit result was 01000011_2 or 67_{10}

Largest number we can represent in 8-bits is 255

The “missing” or “left-over” 1 is called a ***carry*** (or ***carry-out***)

8-bits just for illustration here. Our ARM processor has 32-bit registers and performs 32-bit arithmetic so we get a carry-out if our result requires 33 bits.

Condition Code Flags



Some instructions can optionally update the Condition Code Flags to provide information about the result of the execution of the instruction

e.g. whether the result of an addition was zero, or negative or whether a carry occurred

N – Negative	**Z – Zero**
V – oVerflow	**C – Carry**

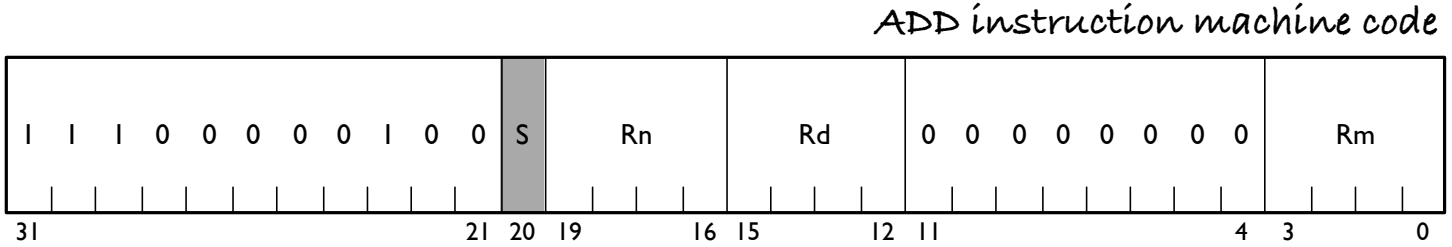
Condition Code Flags

The Condition Code Flags (N, Z, V, C) can be **optionally** updated to reflect the result of an instruction

S-bit in a machine code instruction is used to tell the processor whether the Condition Code Flags should be updated, based on the result

e.g. want to update Condition Code Flags during an ADD instruction

Condition Code Flags only updated if (machine code) S-bit (bit 20) is 1



In assembly language, we cause the Condition Code Flags to be updated by appending “S” to the instruction mnemonic (e.g. ADDS, SUBS, MOVS)

Carry Flag


```
start
        LDR   r0, =0xC0000000
        LDR   r1, =0x70000000
        ADDS  r0, r0, r1

stop    B     stop
```

ADDS causes the Condition Code Flags to be updated

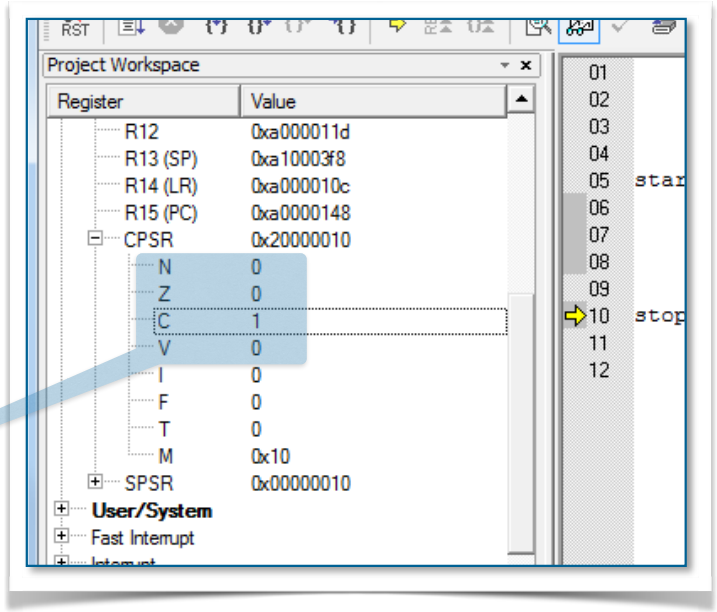
32-bit arithmetic

Expected result?

Does the result fit in 32-bits?

Will the carry flag be set?

Examine flags using µVision IDE



Negative Numbers

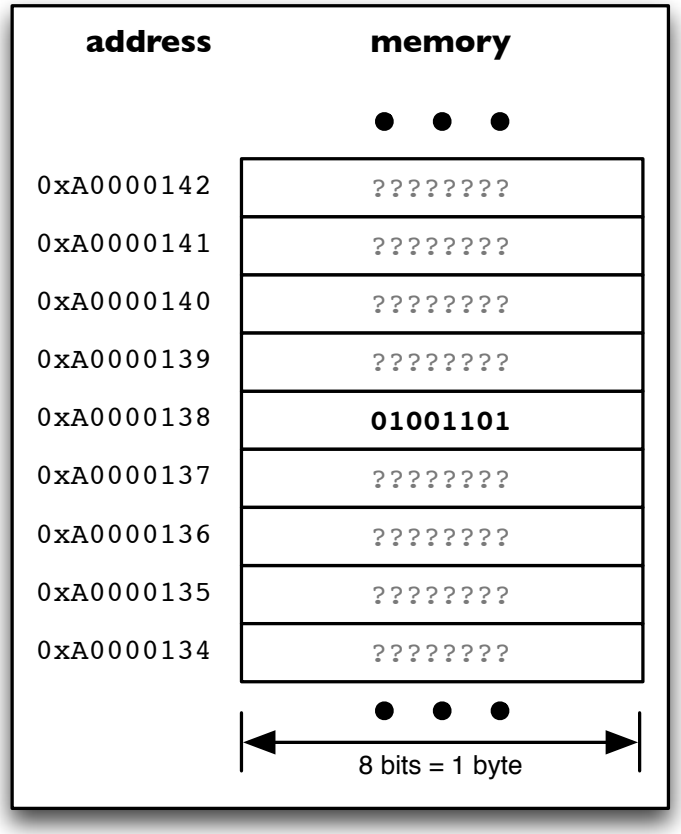
What does the binary value stored in memory at address 0xA0000138 represent?

Interpretation!

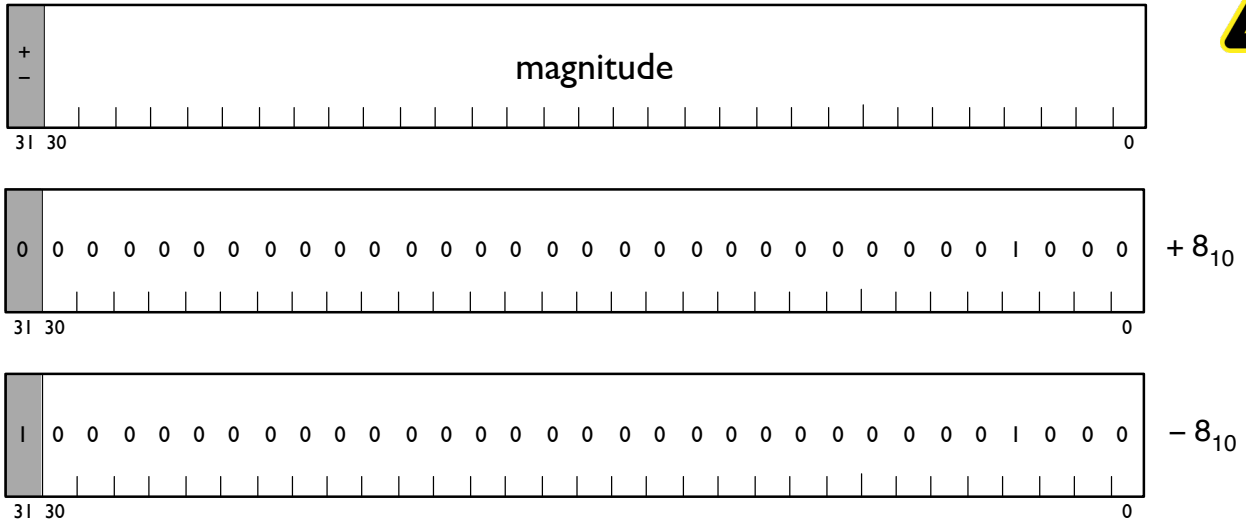
How can we represent signed values, and negative values such as -17_{10} in particular, in memory?

How can we tell whether any given value in memory represents an unsigned value, a signed value, an ASCII character?

(we can't **tell** … as programmers we have to **know**)



Sign-Magnitude *(this is **not** how we usually represent negative numbers!!)*



Represent signed values in the range [$(-2^{31}-1)$ ... $(+2^{31}-1)$]

Two representations of zero (+0 and -0)

Need special way to handle signed arithmetic

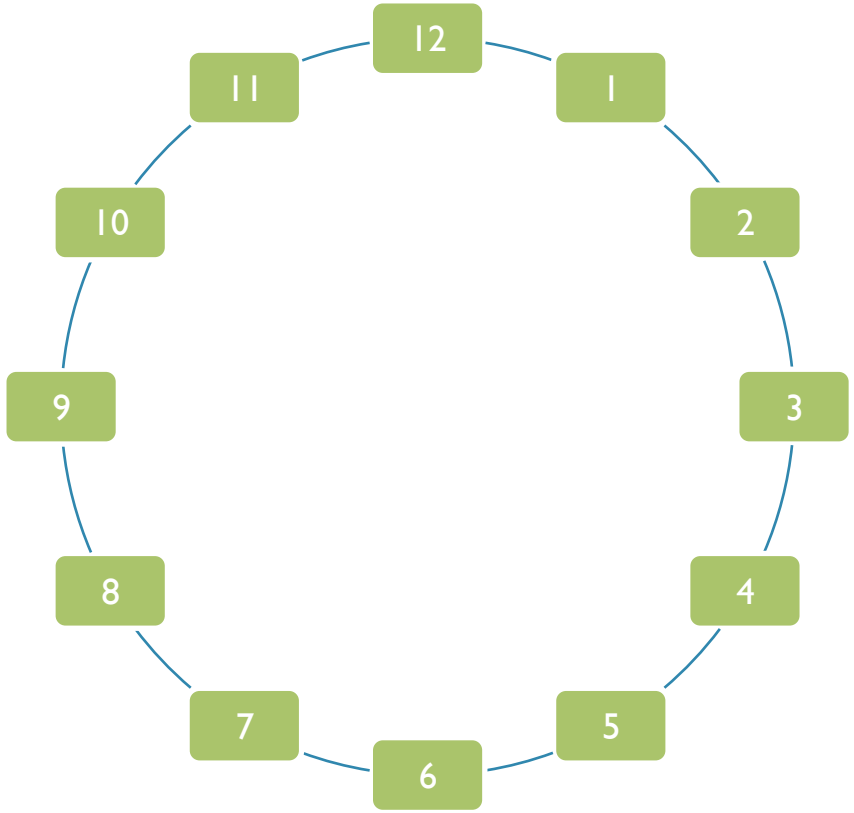
Remember: **interpretation!** (is it -8 or 2,147,483,656?)

Aside: Modulo Arithmetic

A 12-hour clock is an example of modulo-12 arithmetic

If we add 4 hours to 10 o'clock we get 2 o'clock

If we subtract 4 from 2 o'clock we get 10 o'clock (not -2 o'clock!)



jdukes@tcd.ie

2's Complement

Can represent 16 values with a 4-bit number system ($2^4 = 16$)

Ignoring carries from 4-bit binary addition gives us modulo-16 arithmetic (handy)

(15 + 1) mod 16 = 0

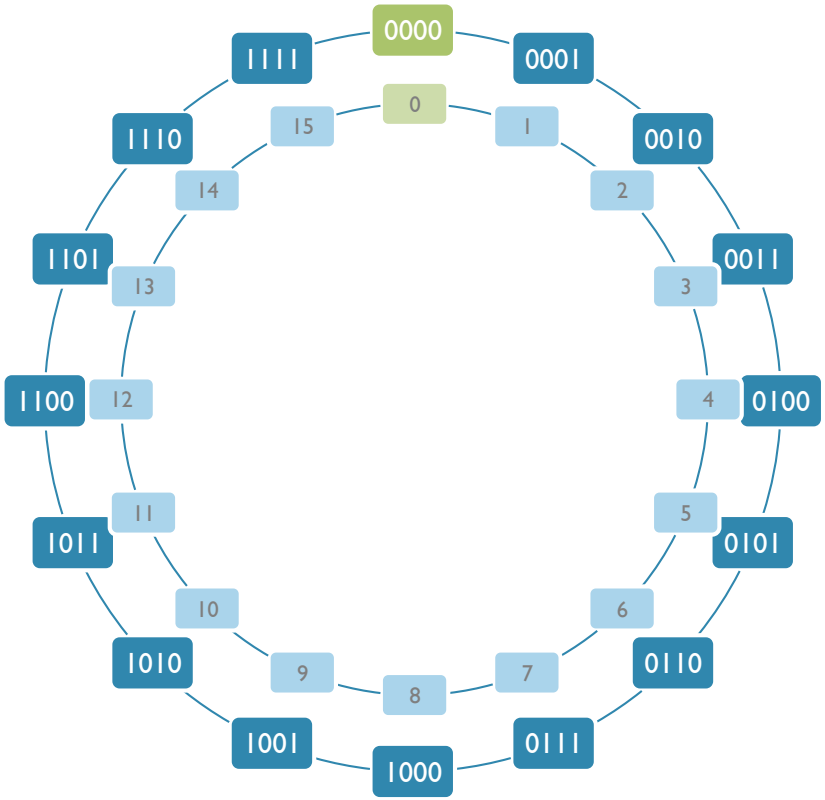
and -1 + 1 = 0

(14 + 2) mod 16 = 0

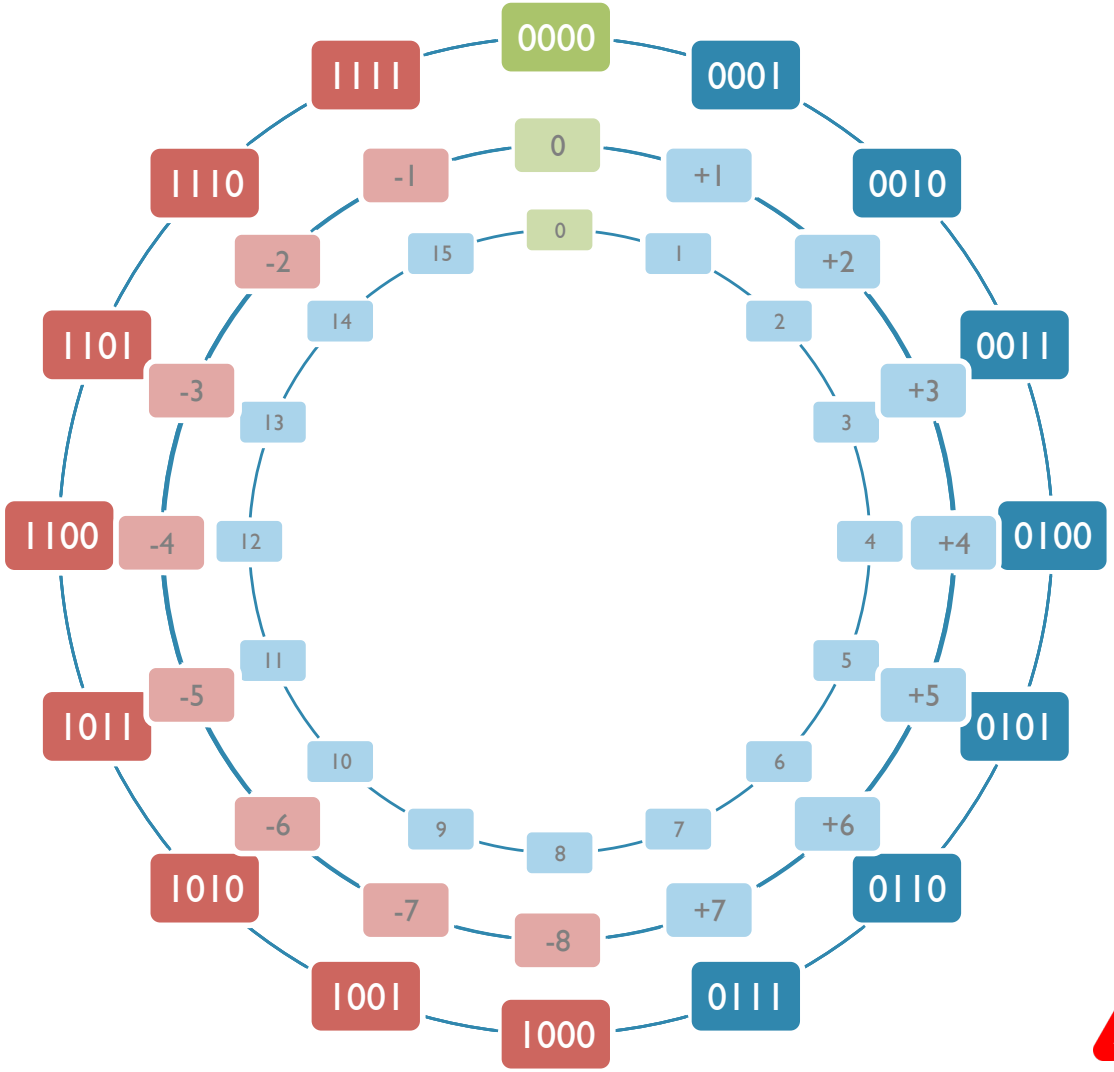
and -2 + 2 = 0

(14 + 4) mod 16 = 2

and -2 +4 = 2



2's Complement



Again, 4-bits just for illustration here. Our ARM processor has 32-bit registers and performs 32-bit arithmetic so we would be taking advantage of modulo-2^{32} arithmetic

Remember: Interpretation!

what does 1001 represent?

jdukes@tcd.ie

2’s Complement

What is the range of signed values that can be represented with 32 bits using the 2’s Complement system?

How would the values -4 and +103 be represented using a 32-bit 2’s Complement system?

How many representations for zero are there?

How can you tell whether a value represented using a 2’s Complement system is positive or negative?

How can we change the sign of a number represented using a 2’s Complement number system?

2's Complement Examples

Represent -97_{10} using 2'c complement

$97_{10} = 01100001_2$

Inverting gives 10011110_2

Adding 1 gives 10011111_2

Interpreted as a 2's complement signed integer

$10011111_2 = -97_{10}$

Interpreted as an unsigned integer

$1001\ 1111_2 = 159_{10}$

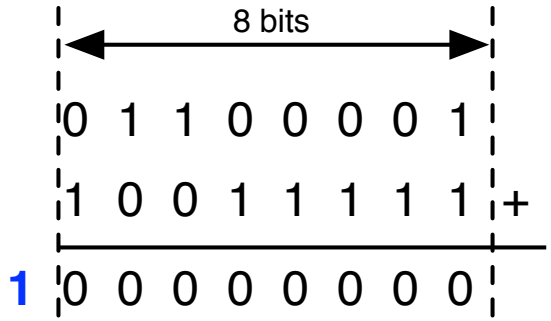
$(159 + 97) \bmod 256 = 0$

Correct interpretation is the responsibility of the programmer, not the CPU, which does not "know" whether a value 10011111_2 in R0 is -97_{10} or 159_{10}

Again, 8-bit values for illustration only here! In practice, we'll be working with 32-bit values

2's Complement Examples

Adding 01100001_2 ($+97_{10}$) and 10011111_2 (-97_{10})



Decimal equivalent
+97
−97 +
0

Ignoring the carry bit gives us the correct result of 0

Changing sign of $1001\ 1111_2$ (-97_{10})

- Invert bits and add 1 again
- Inverting gives 01100000_2
- Adding 1 gives 01100001_2 ($+97_{10}$)

Changing Sign in Assembly Language

Write an Assembly Language program to change the sign of the value stored in R0

Sign of a 2's Complement value can be changed by inverting the value (bits) and adding 1

```
start
        LDR   r0, =7                ; value = 7 (simple test value)
        MVN   r0, r0                ; value = NOT value (invert bits)
        ADD   r0, r0, #1            ; value = value + 1 (add 1)

stop    B     stop
```

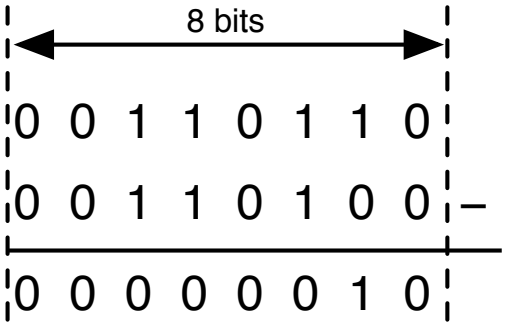
Use of **`LDR r0, =7`** to load a test value (7) into r0

MVN instruction moves a value from one register to another register and negates (inverts) the value

Use of **ADD** with immediate constant value #1 (note syntax)

Subtraction

A – B



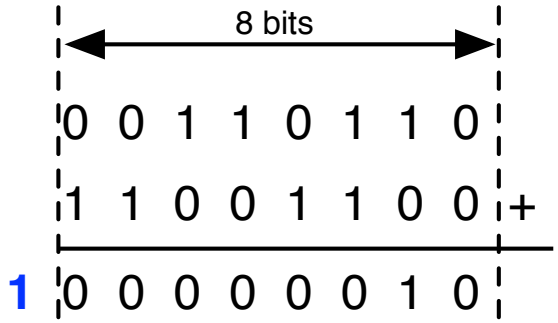
Decimal equivalent

+54

+52 –

+2

A + TwosComplement(B)



Decimal equivalent

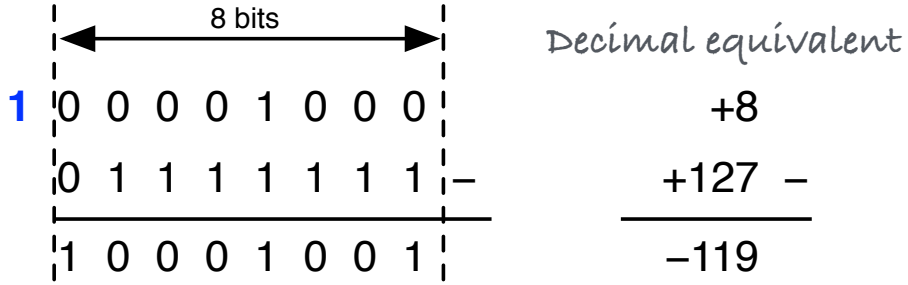
+54

–52 +

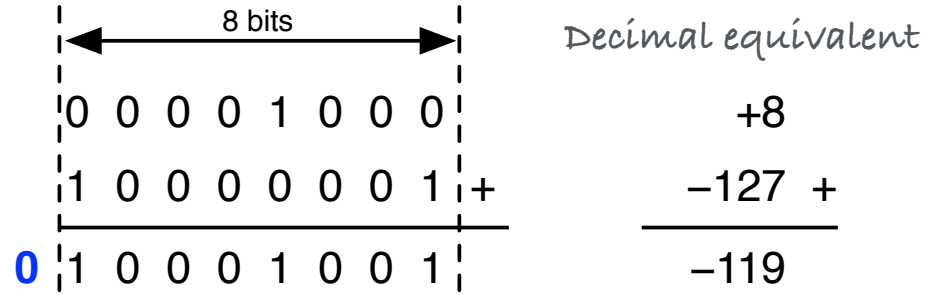
+2

Subtraction

A – B



A + TwosComplement(B)



ARM Instruction Set Reference

“ARM ARM” (ARM Architecture Reference Manual) tells us how each instruction (optionally) affects the Condition Code Flags

e.g. ADD

Operation

```
if ConditionPassed(cond) then
    Rd = Rn + shifter_operand
    if S == 1 and Rd == R15 then
        if CurrentModeHasSPSR() then
            CPSR = SPSR
        else UNPREDICTABLE
    else if S == 1 then
        N Flag = Rd[31]
        Z Flag = if Rd == 0 then 1 else 0
        C Flag = CarryFrom(Rn + shifter_operand)
        V Flag = OverflowFrom(Rn + shifter_operand)
```

ARM Instruction Set Reference

e.g. SUBtract

```
if ConditionPassed(cond) then
    Rd = Rn - shifter_operand
    if S == 1 and Rd == R15 then
        if CurrentModeHasSPSR() then
            CPSR = SPSR
        else UNPREDICTABLE
    else if S == 1 then
        N Flag = Rd[31]
        Z Flag = if Rd == 0 then 1 else 0
        C Flag = NOT BorrowFrom(Rn - shifter_operand)
        V Flag = OverflowFrom(Rn - shifter_operand)
```

Negative and Zero Flags

Zero Condition Code Flag is (optionally) set if the result of the last instruction was zero

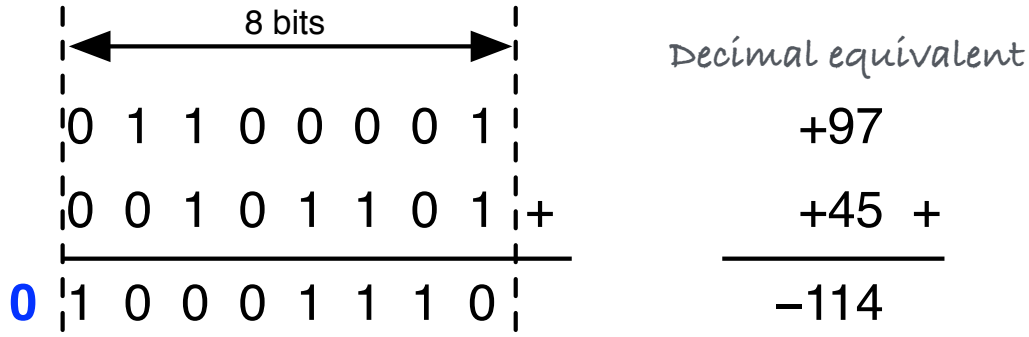
Negative Condition code flag is (optionally) set if the result of the last instruction was negative

i.e. if the Most Significant Bit (MSB) of the result was 1

e.g. SUBtract instruction

```
if ConditionPassed(cond) then
    Rd = Rn - shifter_operand
    if S == 1 and Rd == R15 then
        if CurrentModeHasSPSR() then
            CPSR = SPSR
        else UNPREDICTABLE
    else if S == 1 then
        N Flag = Rd[31]
        Z Flag = if Rd == 0 then 1 else 0
        C Flag = NOT BorrowFrom(Rn - shifter_operand)
        V Flag = OverflowFrom(Rn - shifter_operand)
```

2's Complement Example



Result is 10001110_2 (142_{10}, or -114_{10})

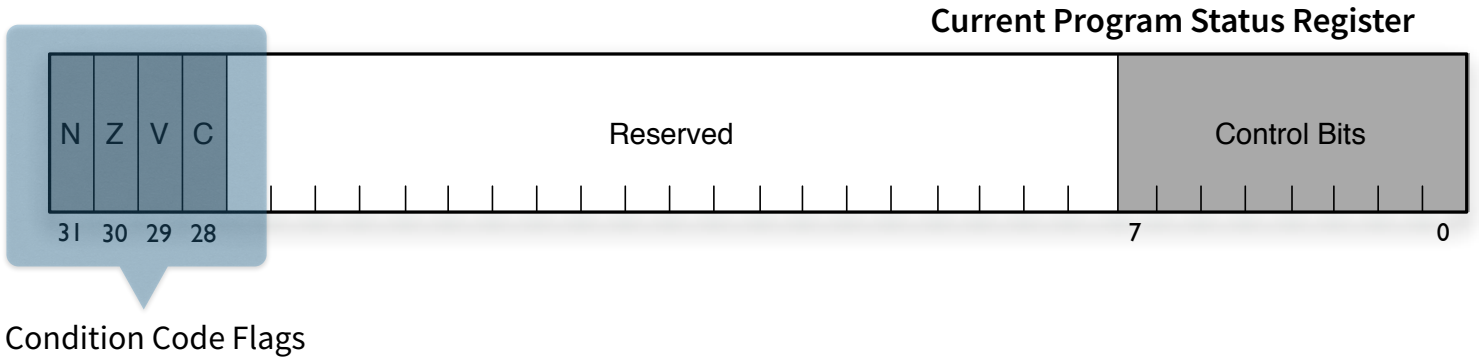
If we were interpreting the two added values and the result as **signed integers**, we got an incorrect result:

- We added two +ve numbers and obtained a –ve result
- With 8-bits, the highest +ve integer we can represent is +127
- 10001110_2 (-114_{10})
- **The result is outside the range of the signed number system**

oVerflow

If the result of an addition or subtraction gives a result that is outside the range of the signed number system, then an oVerflow has occurred

The processor sets the oVerflow Condition Code Flag after performing an arithmetic operation to indicate whether an overflow has occurred



Carry and oVerflow – Interpretation!

Carry and oVerflow flags always set by the processor regardless of our signed or unsigned interpretation of stored values

Processor does not “know” what our interpretation is

e.g. we could interpret the binary value 10001110_2 as either 142_{10} (unsigned) or -114_{10} (signed)

(we could also interpret it as the code for “Ä” or as the colour blue)

The C and V flags are set by the processor and it is our responsibility to choose:

whether to interpret C or V (are we interpreting the values as unsigned or signed?)

how to interpret C or V

oVerflow Rules

Addition rule (r = a + b)

V = 1 if MSB(a) = MSB(b) and
MSB(r) ≠ MSB(a)

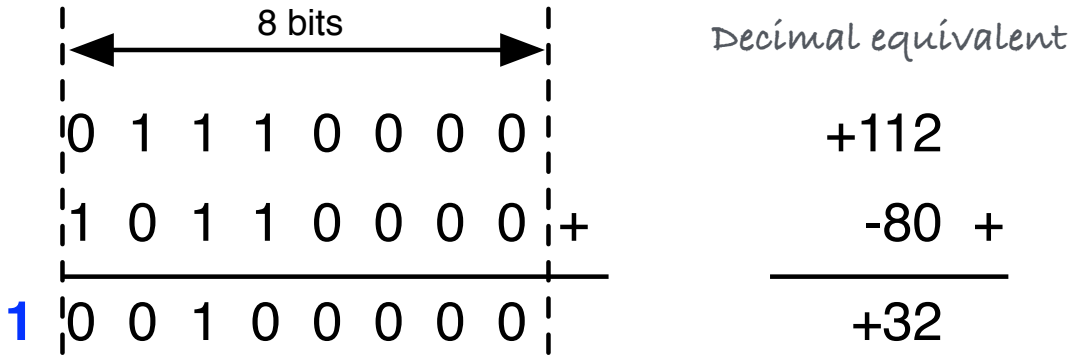
i.e. oVerflow accurs for addition if the operands have the same sign and the result has a different sign

Subtraction rule (r = a – b)

V = 1 if MSB(a) ≠ MSB(b) and
MSB(r) ≠ MSB(a)

i.e. oVerflow occurs for subtraction if the operands have different signs and the sign of the result is different from the sign of the first operand

Carry and oVerflow Example



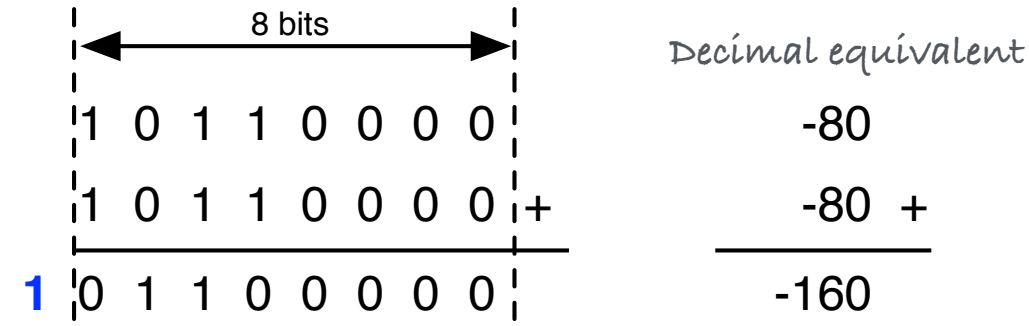
Signed interpretation: (+112) + (-80) = +32

Unsigned interpretation: 112 + 176 = 288

By examining the V flag (V = 0), we know that if were interpreting the values as signed integers, the result is correct

If we were interpreting the values as 8-bit unsigned values, C = 1 tells us that the result was too large to fit in 8-bits

Carry and oVerflow Example



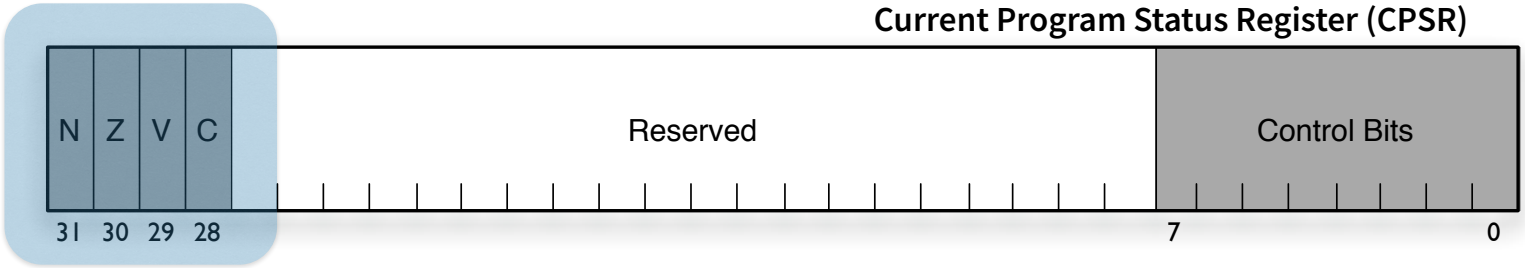
Signed: (-80) + (-80) = -160

Unsigned: 176 + 176 = 352

By examining the V flag (V = 1), we know that if were interpreting the values as signed integers, the result is outside the range of the signed number system

If we were interpreting the values as 8-bit unsigned values, C = 1 tells us that the result was too large to fit in 8-bits

Condition Code Flags – Recap



Many instructions can optionally cause the processor to update the Condition Code Flags (N, Z, V, and C) to reflect certain properties of the result of an operation

Append "S" to instruction in assembly language (e.g. ADDS)

Set S-bit in machine code instruction

Remember: Processor does this regardless of our interpretation of values as signed or unsigned

N flag set to 1 if result is **N**egative (i.e. if MSB is 1)

Z flag is set to 1 if result is **Z**ero (i.e. all bits are 0)

C flag set if **C**arry occurs (addition) or borrow does not occur (subtraction)

V flag set if o**V**erflow occurs for addition or subtraction