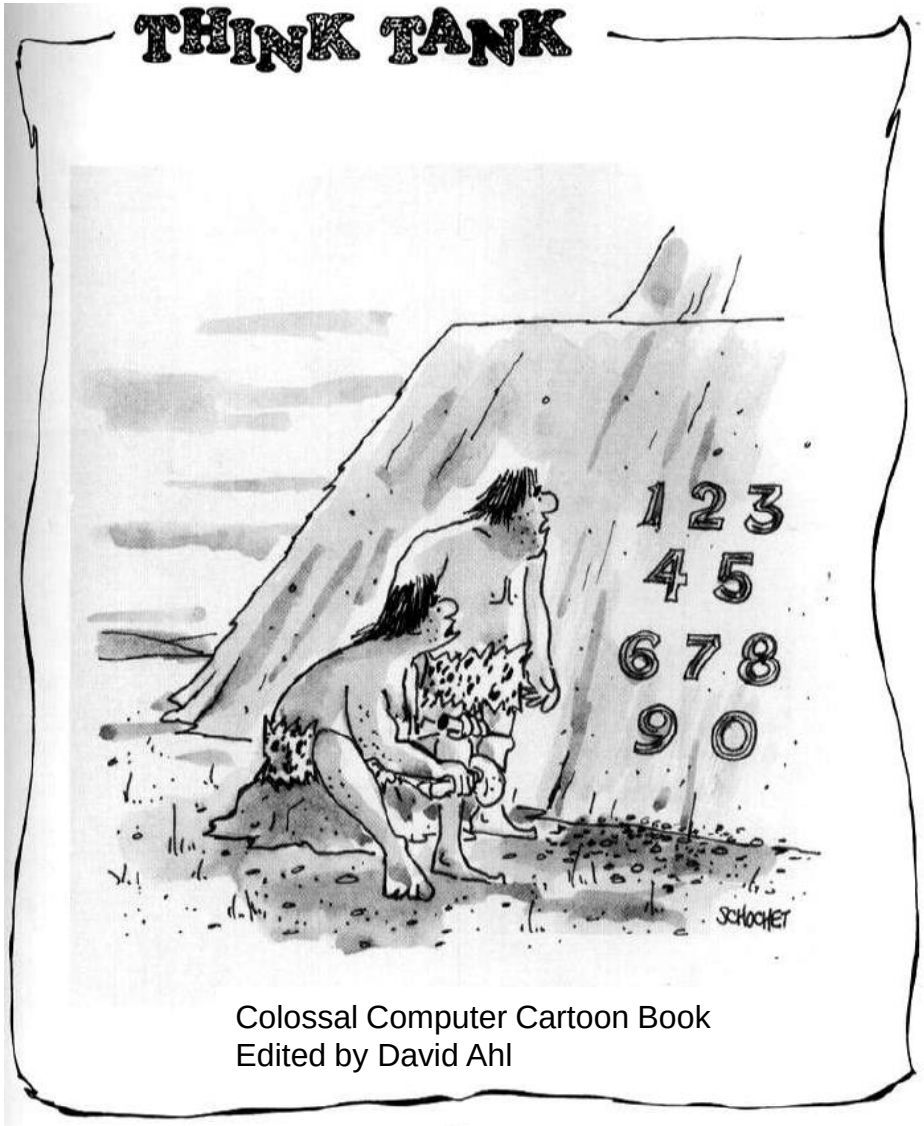
Arithmetic Operations

Computer Engineering 1

Motivation

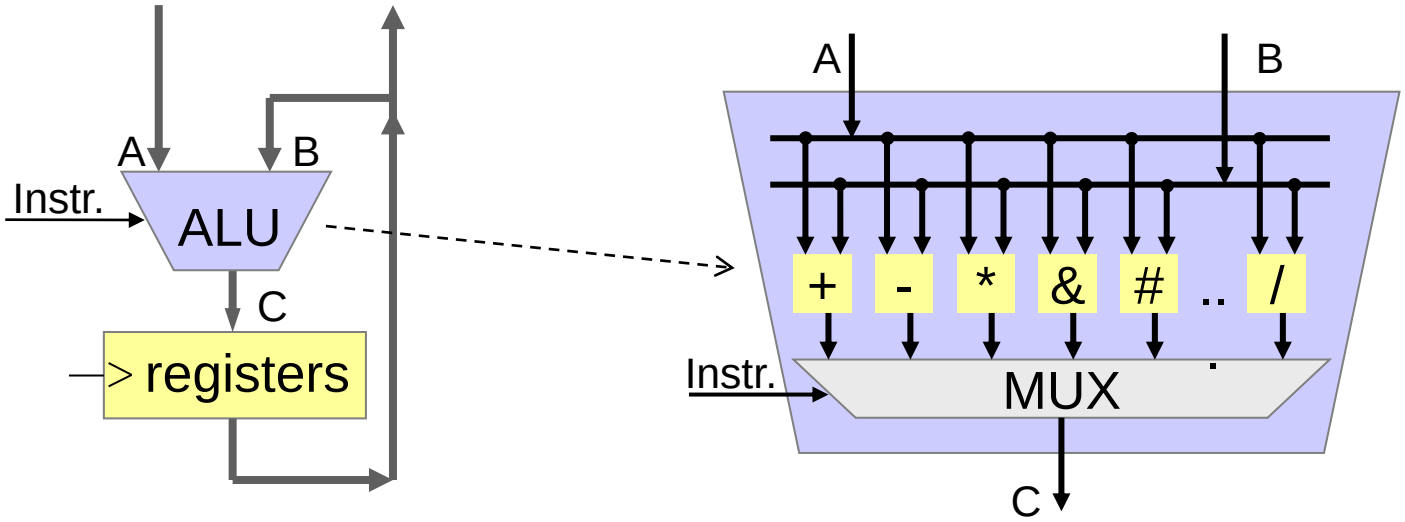


Colossal Computer Cartoon Book
Edited by David Ahl

"I call them numbers, you can add them, subtract them, multiply them, divide them ... find their square root ..."

Motivation

- Instructions to process data in the ALU
  - **arithmetic** Addition, Subtraction, Multiplication, Division
  - **logic** NOT, AND, OR, XOR
  - **shift** Shift left/right. Fill with 0 or MSB
  - **rotate** Cyclic shift left/right: What drops out enters on the other side



Motivation

School of Engineering
InES Institute of Embedded Systems

- **How often is the for loop executed?**
  - For **`#define INCREMENT 1`**
  - For **`#define INCREMENT 10`**

```
...
int main(void) {
    uint8_t uc;
    uint32_t count = 0;

    for (uc = 0; uc <  255; uc += INCREMENT) {
        printf("hello again %d \n",uc);
        count++;
    }
    printf("Loop executed %d times\n",count);
}
```

solution

Motivation

School of Engineering
InES Institute of Embedded Systems

- **How often is the for loop executed?**
  - For `#define INCREMENT 1` → **255 times**
  - For `#define INCREMENT 10` → **infinite times**
    >= 255 never reached

```
...
int main(void) {
    uint8_t uc;
    uint32_t count = 0;

    for (uc = 0; uc <  255; uc += INCREMENT) {
        printf("hello again %d \n",uc);
        count++;
    }
    printf("Loop executed %d times\n",count);
}
```

Why?

```
.....
hello again 210
hello again 220
hello again 230
hello again 240
hello again 250
hello again 4
hello again 14
hello again 24
hello again 34
hello again 44
.......
```

Start 16.10.2023

School of Engineering
InES Institute of Embedded Systems

Recap/Übung zu Data-Transfer

School of Engineering
InES Institute of Embedded Systems

- Die Area `myData` liegt im Memory ab Adresse 0x20000004.
- Ergänzen Sie die Instruktionen und Registerinhalte im Listing unten
- Zeichnen Sie Memory-Map und aktualisieren Sie diese nach jeder Store-Instruktion.

Label	Menmonic and operands	Comments	Results
	AREA myData,DATA,READWRITE		
my32BitVar	DCD 0x22446688		
	AREA myCode,CODE,READONLY		
start	MOVS R5,#3	; Initialize R5 with 3	R5 =
	LDR R7,	; load address of my32BitVar into R7	R7 =
		; load my32BitVar as unsigned 32-Bit into R0	R0 =
		; load my32BitVar as unsigned 16-Bit into R1	R1 =
		; load my32BitVar as unsigned 8-Bit into R2	R2 =
	MOVS R5,#3		
	STRB R0,[R7,R5]	; **Memory map on next page**	
	STR R2,[R7]	; **Memory map on next page**	

solution

Recap zu Data-Transfer

School of Engineering
InES Institute of Embedded Systems

Label	Menmonic and operands	Comments	Results
	AREA myData,DATA,READWRITE		
my32BitVar	DCD 0x22446688		
	AREA myCode,CODE,READONLY		
start	MOVS R6,#0	; Initialize R6 with 0	R6 = 0x00000000
	LDR R7,=my32BitVar	; load address of my32BitVar into R7	R7 = 0x20000004
	LDR R0,[R7,#0]	; load my32BitVar as unsigned 32-Bit into R0	R0 = 0x22446688
Explicit offset	LDRH R1,[R7,#0]	; load my32BitVar as unsigned 16-Bit into R1	R1 = 0x00006688
optional if zero	LDRB R2,[R7]	; load my32BitVar as unsigned 8-Bit into R2	R2 = 0x00000088
	MOVS R5,#3		
	STRB R0,[R7,R5]	; **Memory map on next page**	
	STR R2,[R7]	; **Memory map on next page**	

solution

Memory Map

School of Engineering
InES Institute of Embedded Systems

Address	Label	Nach Initialisierung	Nach Task 3a (STRB)	Nach Task 3b (STR)
0x2000'0003				
0x2000'0004	My32BitVar	0x88	0x88	**0x88**
0x2000'0005		0x66	0x66	**0x00**
0x2000'0006		0x44	0x44	**0x00**
0x2000'0007		0x22	**0x88**	**0x00**
0x2000'0008				

Agenda

School of Engineering
InES Institute of Embedded Systems

- **Cortex-M0**
  - Data flow
  - Flags
  - Overview of arithmetic instructions
- **Add Instructions Cortex-M0**
- **Negative Numbers**
- **Addition**
- **Subtraction**
- **Subtract Instructions Cortex-M0**
- **Multi-Word Arithmetic**
- **Multiplication**

Learning Objectives

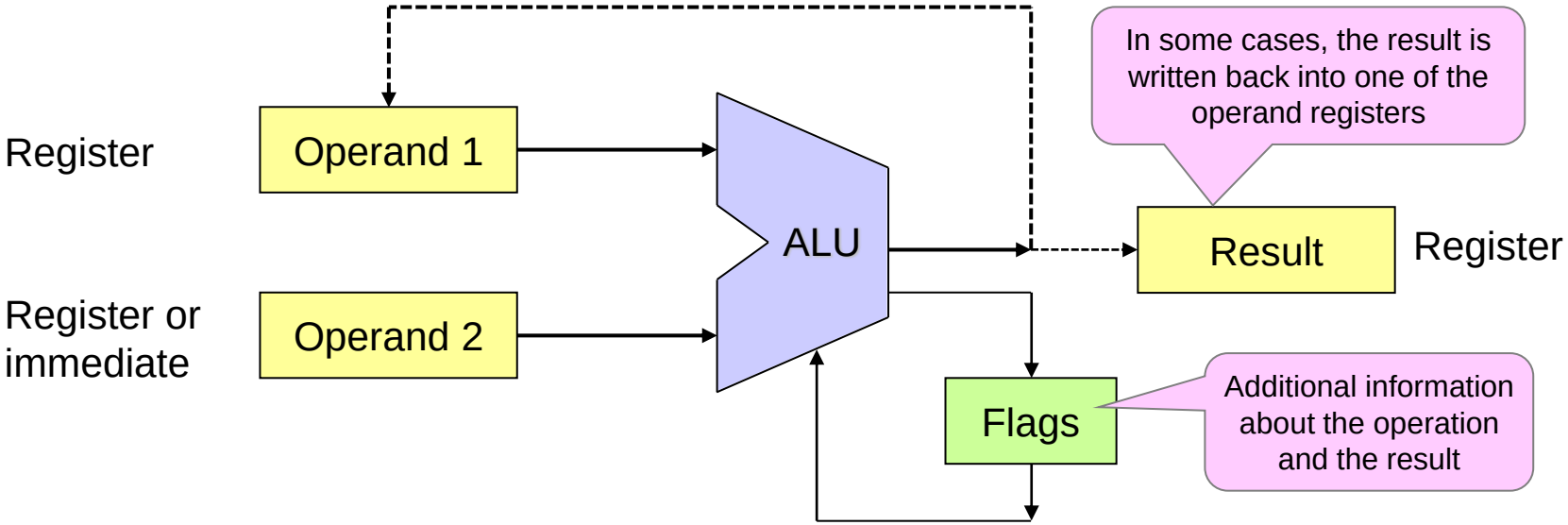
School of Engineering
InES Institute of Embedded Systems

At the end of this lesson, you will be able

- to enumerate and apply the Cortex-M0 arithmetic instructions
- to interpret Cortex-M0 assembly programs with arithmetic instructions
- to enumerate and explain the meaning of the Cortex-M0 Flags (N, Z, C, V)
- to carry out additions and subtractions of signed and unsigned integers and to explain the operations with the circle of numbers
- to calculate and interpret carry/borrow and overflow/underflow
- to determine (with the help of documents) the state of Cortex-M0 Flags (N, Z, C, V) after an arithmetic instruction
- to describe how addition and subtraction are done in hardware (in the ALU)
- to program integer calculations with operands that exceed the number of bits available in the ALU
- to explain how numbers in two's complements representation are multiplied

Data flow (Cortex-M0)

- **Operands and results stored in registers**
  - Exception: Immediate operations with data in OP-code
  - Load/store architecture



Examples

```
ADDS    R0,R0,R1      ; R0 = R0 + R1    result back in R0
ADDS    R0,#0x34      ; R0 = R0 + 0x34
SUBS    R3,R4,#5      ; R3 = R4 - 0x05  result in other reg
```

Wir springen vor…

School of Engineering
InES Institute of Embedded Systems

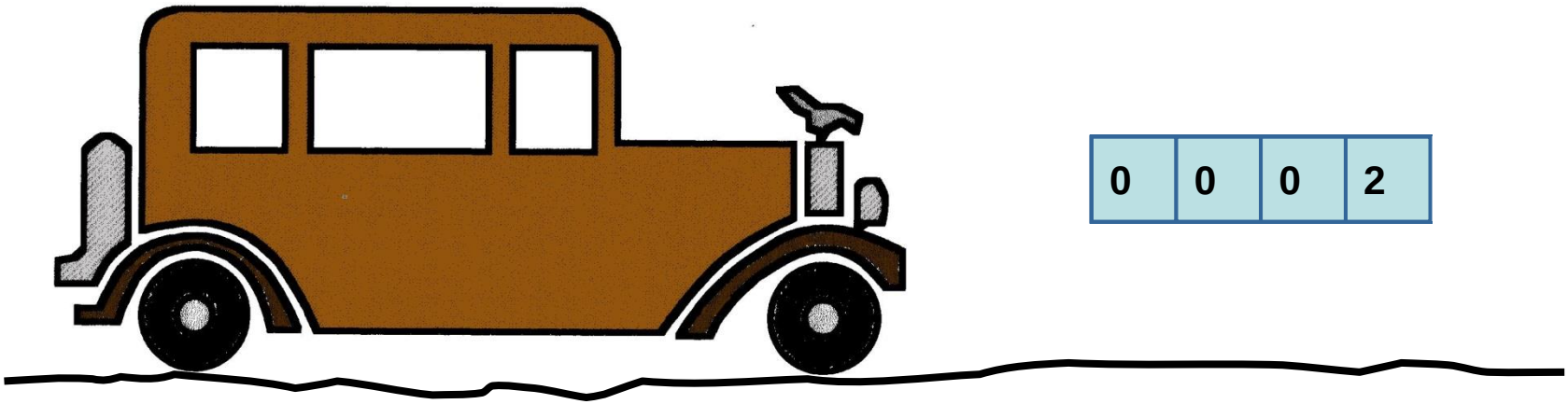
- **In den nächsten Folien nehmen wir 4-Bit-Zahlen als Referenz, also die Zahlen von 0b0000 – 0b1111**

Animation

Negative Numbers

School of Engineering
InES Institute of Embedded Systems

- **Lord Ardry's new Rolls Royce**

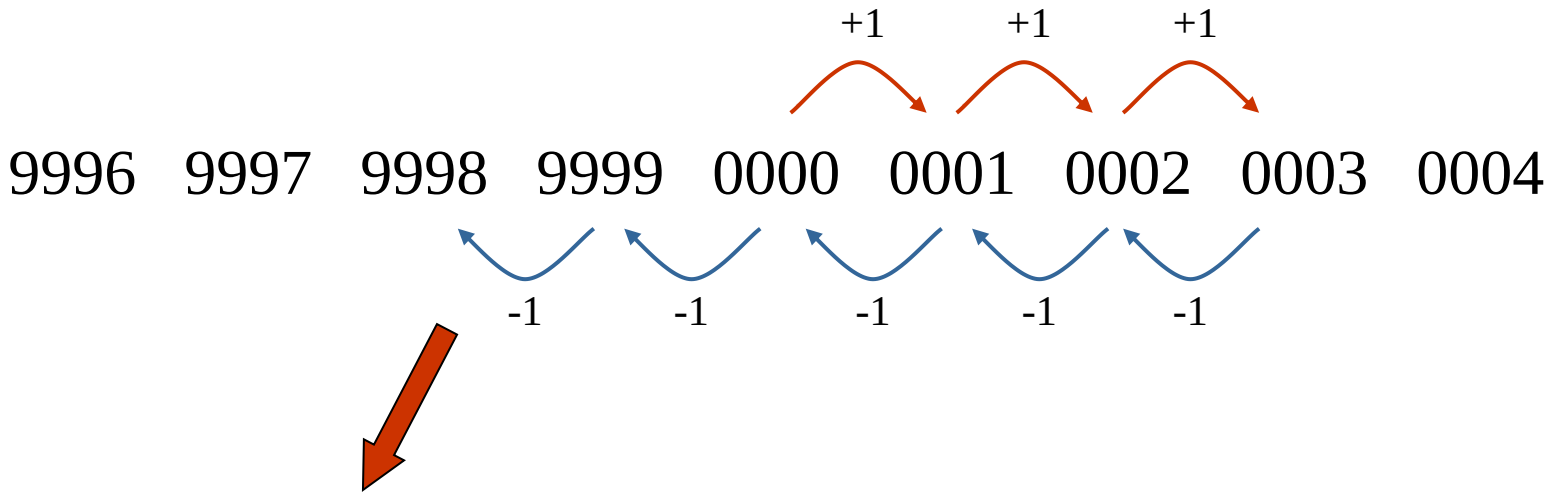


0	0	0	2

Negative Numbers

School of Engineering
InES Institute of Embedded Systems

- **What happened?**



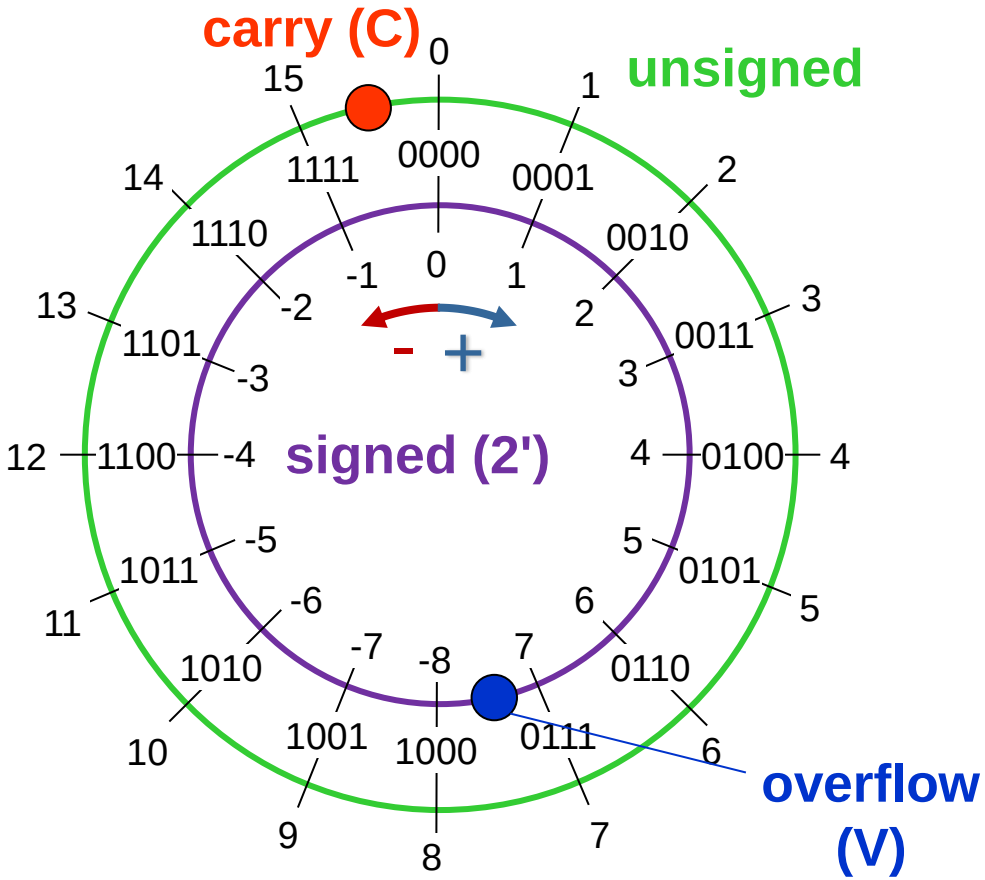
0000 - 1 - 1 ≙ -2 ≙ 9998

possible **because**
- number of digits is finite
- respectively given by the word size

Negative Numbers

signed $-b_3 \cdot 2^3 + b_2 \cdot 2^2 + b_1 \cdot 2^1 + b_0 \cdot 2^0$

unsigned $+b_3 \cdot 2^3 + b_2 \cdot 2^2 + b_1 \cdot 2^1 + b_0 \cdot 2^0$

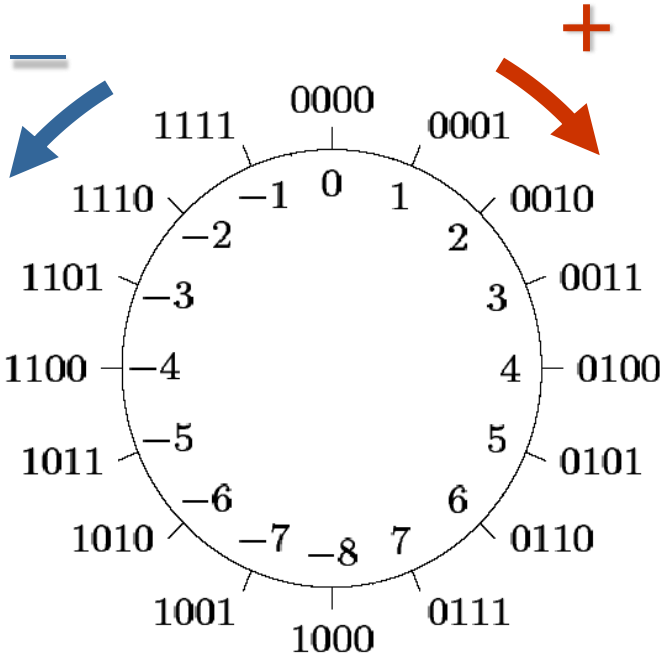


binary	unsigned	signed 2'-Compl.
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	-8
1001	9	-7
1010	10	-6
1011	11	-5
1100	12	-4
1101	13	-3
1110	14	-2
1111	15	-1

Negative Numbers

School of Engineering
InES Institute of Embedded Systems

- **Numbers with finite number of digits**
  - can be represented on a circle
  - Addition
    - Clockwise
  - Subtraction
    - Counter-clockwise



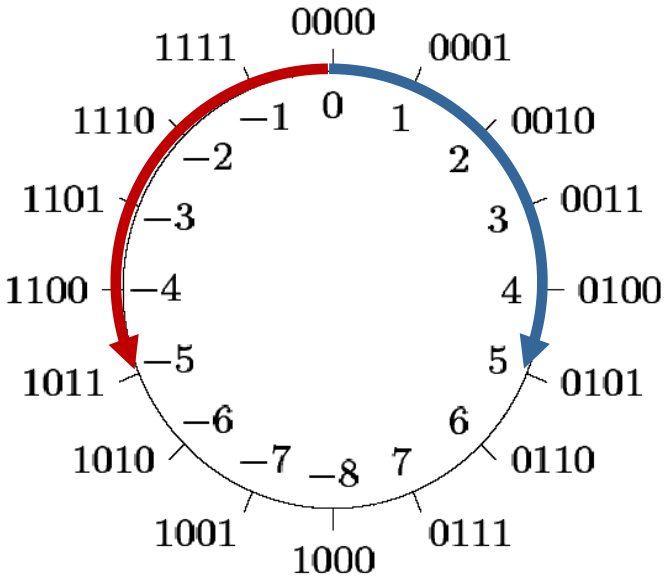
Negative Numbers

School of Engineering
InES Institute of Embedded Systems

- **Negation of a number**
  - **Idea: – a = 0 – a**
  - Starting at 0000 enter the absolute value of **a** counter-clockwise
  - Example: 5d = 0101 ➔ -5d = 1011

written calculation

```
  0000
- 0101
  1111
  ----
  1011
  ====
```

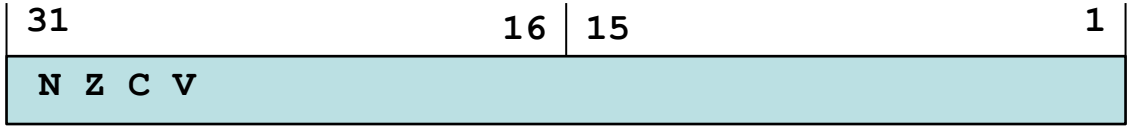


Back on Track…

School of Engineering
InES Institute of Embedded Systems

Flags

- **Cortex-M0**
  - APSR: Application Program Status Register



Flag	Meaning	Action	Operands
Negative	MSB = 1	N = 1	signed
Zero	Result = 0	Z = 1	signed , unsigned
Carry	Carry	C = 1	unsigned
Overflow	Overflow	V = 1	signed

- Processor does not know whether the user is working with **unsigned (C)** or with **signed (V)** numbers

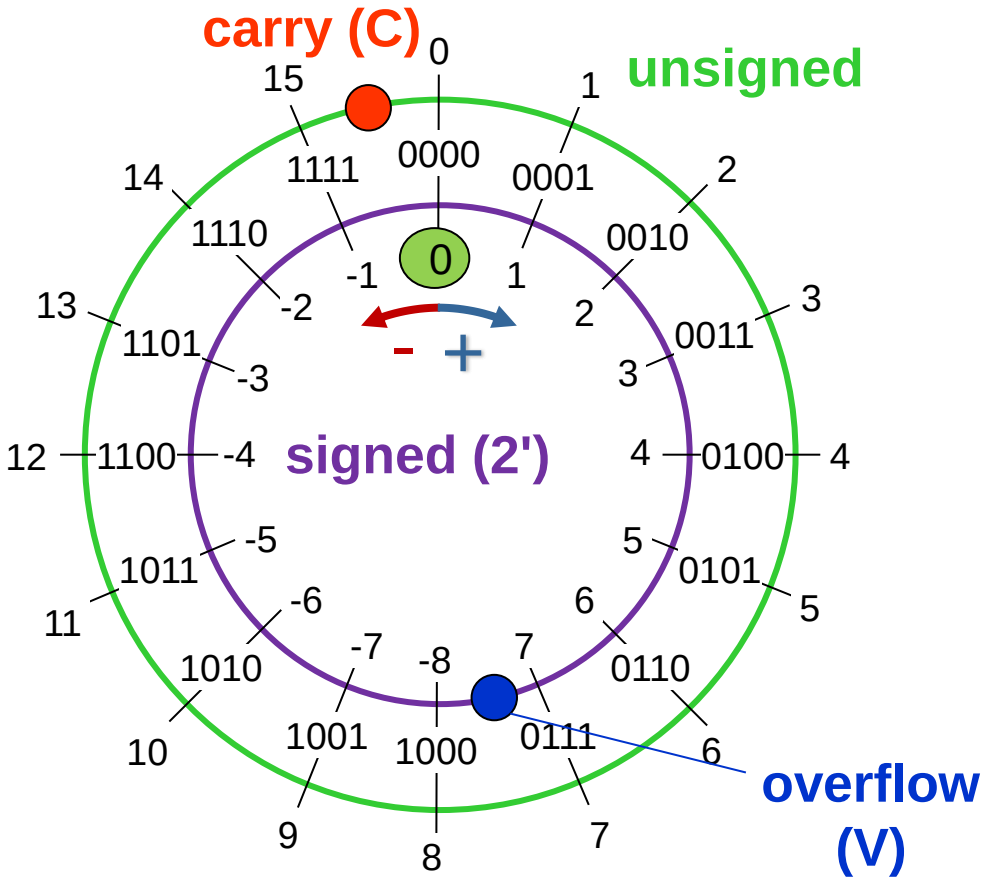
  → Processor therefore always calculates C **and** V!

Negative Numbers

School of Engineering
InES Institute of Embedded Systems

signed $-b_3 \cdot 2^3 + b_2 \cdot 2^2 + b_1 \cdot 2^1 + b_0 \cdot 2^0$

unsigned $+b_3 \cdot 2^3 + b_2 \cdot 2^2 + b_1 \cdot 2^1 + b_0 \cdot 2^0$

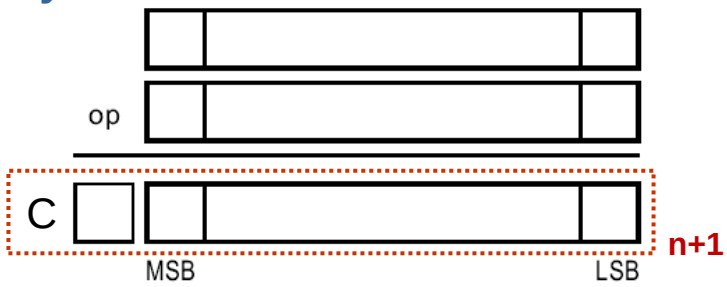


binary	unsigned	signed 2'-Compl.
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	-8
1001	9	-7
1010	10	-6
1011	11	-5
1100	12	-4
1101	13	-3
1110	14	-2
1111	15	-1

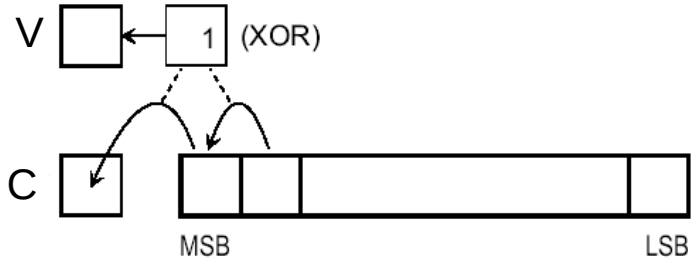
Flags

- **Cortex-M0**
  - Instructions ending with ‚S‘ allow modification of flags
  - Examples: **`MOVS, ADDS, SUBS,`** …

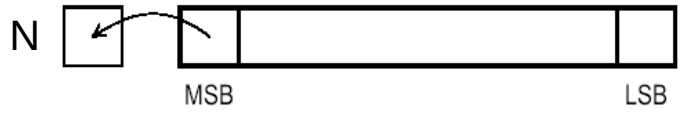
Carry



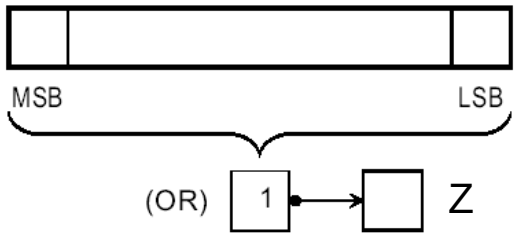
Overflow



Negative



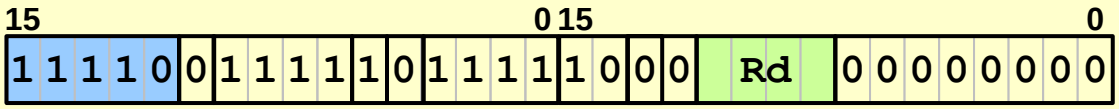
Zero



Flags

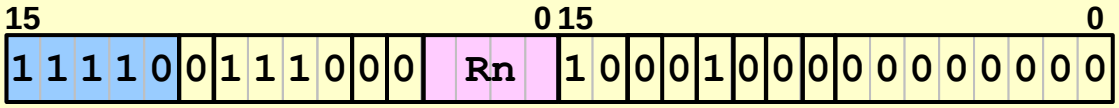
- **Instructions to access APSR**
  - **`MRS`** Move to register from special register (APSR)
  - **`MSR`** Move to special register (APSR) from register
  - 32-bit opcode

```
MRS <Rd>, APSR
```




```
Rd = APSR
```

```
MSR APSR, <Rn>
```




```
APSR = Rn
```

Arithmetic Instructions

School of Engineering
InES Institute of Embedded Systems

- **Overview Cortex-M0**

Mnemonic	Instruction	Function
ADD / ADDS	Addition	A + B
ADCS	Addition with carry	A + B + c
ADR	Address to Register	PC + A
SUB / SUBS	Subtraction	A - B
SBCS	Subtraction with carry (borrow)	A - B - NOT(c) [1)]
RSBS	Reverse Subtract (negative)	-1 • A
MULS	Multiplication	A • B

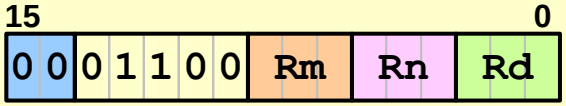
[1)] borrow = NOT(c)

Add Instructions Cortex-M0

School of Engineering
InES Institute of Embedded Systems

- **ADDS (register)**
  - Update of flags
  - Result and two operands
  - Only low register

ADDS <Rd>,<Rn>,<Rm>



Rd = Rn + Rm

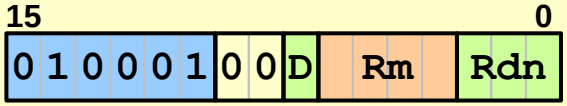
```
00000002 18D1    ADDS     R1,R2,R3
00000004 1889    ADDS     R1,R1,R2
00000006 1889    ADDS     R1,R2       ; the same (dest = R1)
00000008         ;ADDS    R9,R2       ; not possible (high reg)
00000008         ;ADDS    R1,R10      ; not possible (high reg)
```

Add Instructions Cortex-M0

School of Engineering
InES Institute of Embedded Systems

- **ADD (register)**
  - No update of Flags
  - High or low register
  - <Rdn> → result and operand
    - I.e. same register for operand and result!

```
ADD <Rdn>,<Rm>
```




```
Rdn = Rdn + Rm
```

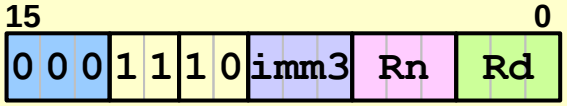
```
00000008 4411    ADD      R1,R1,R2     ; low regs
0000000A 44D1    ADD      R9,R9,R10    ; high regs
0000000C 44D1    ADD      R9,R10       ; the same (dest = R9)
0000000E 4411    ADD      R1,R1,R2
0000000E         ;ADD     R1,R2,R3     ; not possible
```

Add Instructions Cortex-M0

School of Engineering
InES Institute of Embedded Systems

- **ADDS (immediate) – T1**
  - Update of flags
  - Two different[1)] low registers and immediate value 0 - 7

```
ADDS <Rd>,<Rn>,#<imm3>
```




```
Rd = Rn + <imm3>
```

```
00000010 1D63     ADDS    R3,R4,#5
00000012          ;ADDS    R3,R4,#8     ; out of range immediate
00000012          ;ADDS    R10,R11,#5   ; not possible (high reg)
```

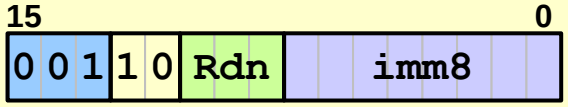
[1)] If the same register is used for Rd and Rn, the assembler will choose the encoding T2 on the next slide

Add Instructions Cortex-M0

School of Engineering
InES Institute of Embedded Systems

- **ADDS (immediate) – T2**
  - Update of flags
  - Low register with immediate value 0 - 255d
  - <Rdn> → Result and operand in same register

```
ADDS <Rdn>,#<imm8>
```




```
Rdn = Rdn + <imm8>
```

```
00000012 33F0     ADDS    R3,R3,#240
00000014 33F0     ADDS    R3,#240       ; the same (dest = R3)
00000016          ;ADDS    R8,R8,#240 ; not possible (high reg)
00000016          ;ADDS    R3,#260    ; out of range immediate
```

Add Instructions Cortex-M0

School of Engineering
InES Institute of Embedded Systems

- **ADD / ADDS (Summary)**
  - ADD Flags not changed
  - ADDS Flags changed according to Operation Result

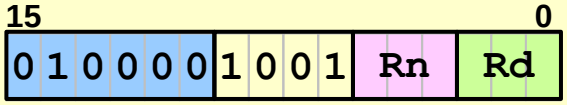
Instr	Rd	Rn	Rm	imm	Restrictions
ADD	R0-R15	R0-R15	R0-R15	-	Rd and Rn must specify the same register. Rn and Rm must not both specify the PC (R15)
ADDS	R0-R7	R0-R7	-	0 - 7	-
ADDS	R0-R7	R0-R7	-	0 - 255	Rd and Rn must specify the same register
ADDS	R0-R7	R0-R7	R0-R7	-	-

Negative Numbers

School of Engineering
InES Institute of Embedded Systems

- **Instruction RSBS**
  - *Reverse Subtract*
  - Generates 2‘ complement
  - Updates flags
  - Only low register possible

RSBS <Rd>,<Rn>,#0



Rd = 0 - Rn

```
00000022 4251    RSBS     R1,R2,#0
00000024 427F    RSBS     R7,R7,#0
00000026 427F    RSBS     R7,#0        ; the same (dest = R7)
00000028         ;RSBS    R8,R1,#0     ;not possible (high reg)
00000028         ;RSBS    R1,R8,#0     ;not possible (high reg)
```

Negative Numbers

School of Engineering
InES Institute of Embedded Systems

- **Example: Instruction RSBS**

C-Code

```
int32_t intA = 3;
int32_t intB;

int32_t invertSign(void)
{
    ...
    intB = -intA;
    ...
}
```

Assembler-code

```
AREA MyCode, CODE, READONLY
        ...
        LDR     R0,=intA    (1)
        LDR     R0,[R0,#0]
        RSBS    R0,R0,#0
        LDR     R1,=intB    (1)
        STR     R0,[R1,#0]
        ...

AREA MyData, DATA, READWRITE
intA    DCD 0x00000003
intB    DCD 0x00000000
```

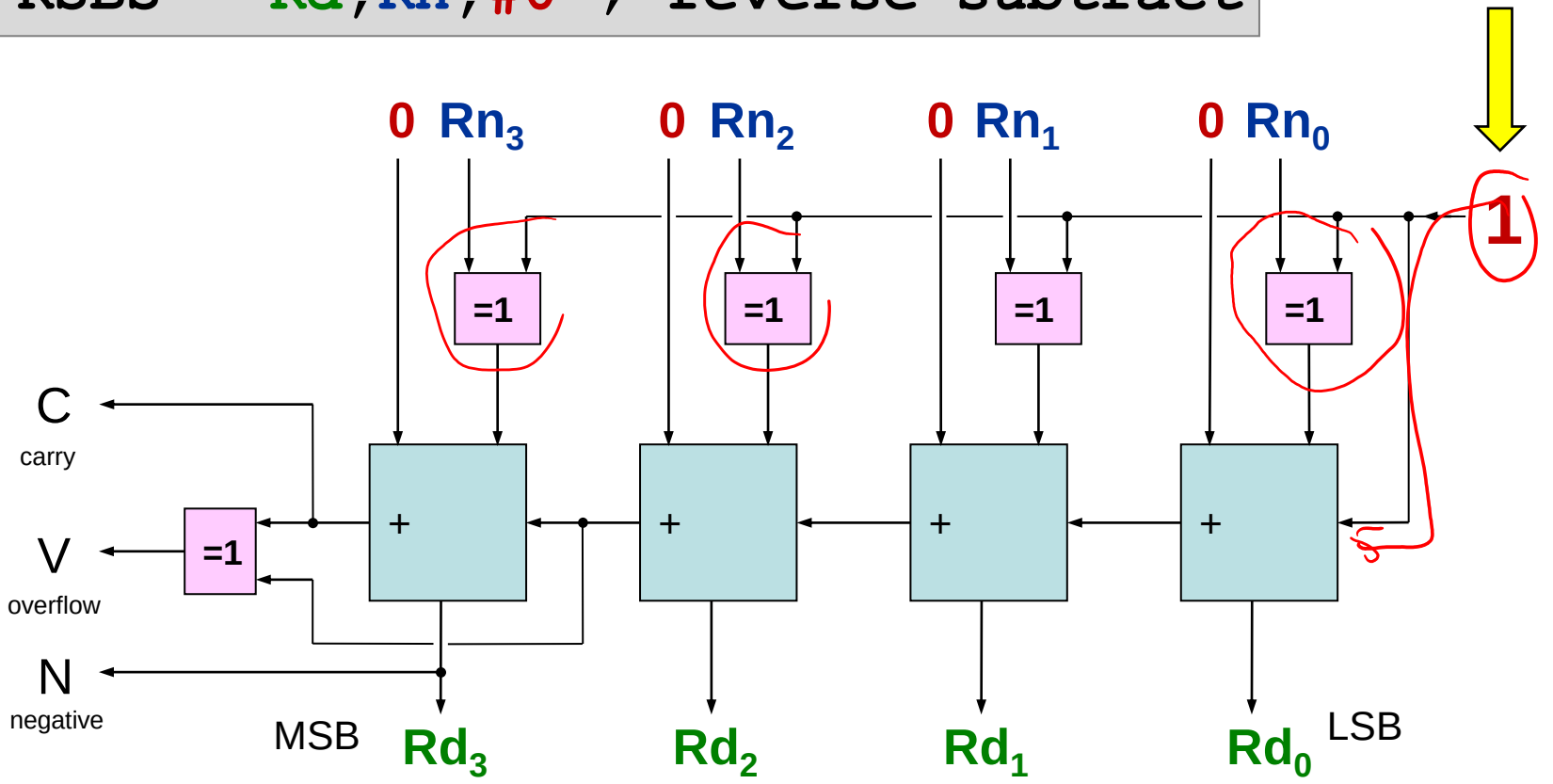
(1) The compiler does not use the pseudo-instruction LDR in this case

Negative Numbers

School of Engineering
InES Institute of Embedded Systems

- **2‘ complement in Hardware (ALU)**

```
RSBS    Rd,Rn,#0 ; reverse subtract
```



Addition

School of Engineering
InES Institute of Embedded Systems

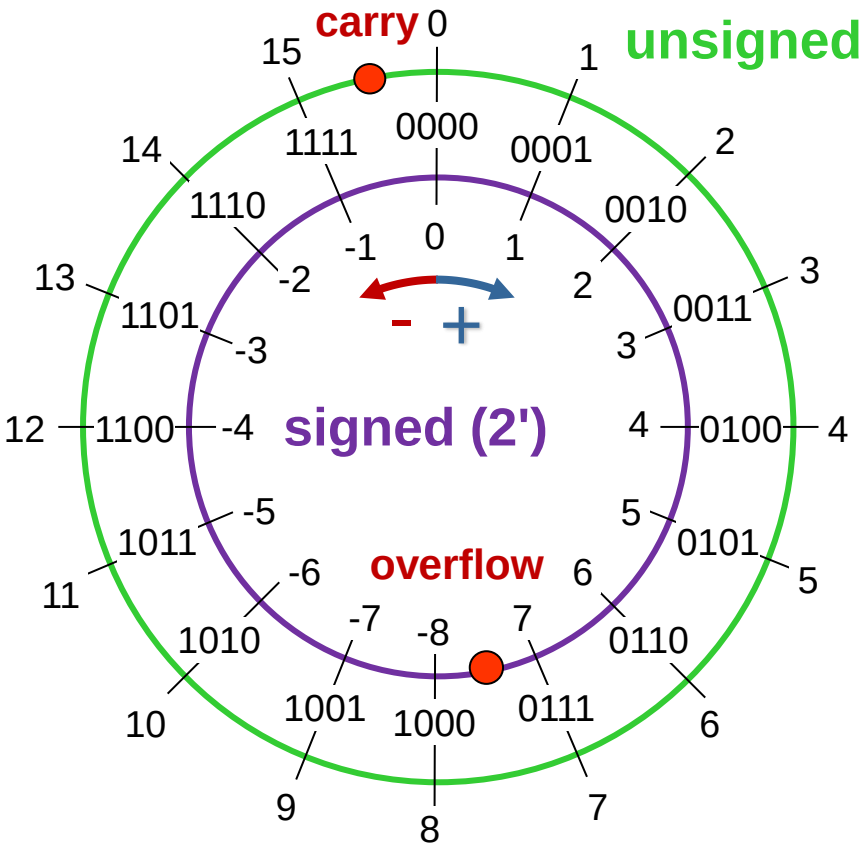
- **Unsigned**
  - C = 1 indicates carry
  - V irrelevant
  - Addition of 2 big numbers
    → can yield a small result
- **Signed**
  - V = 1 indicates overflow
    - possible with same „sign“

+	+	→	–
–	–	→	+

  - C irrelevant

Programmer interprets register content
either as signed or as unsigned.
CPU does not care!

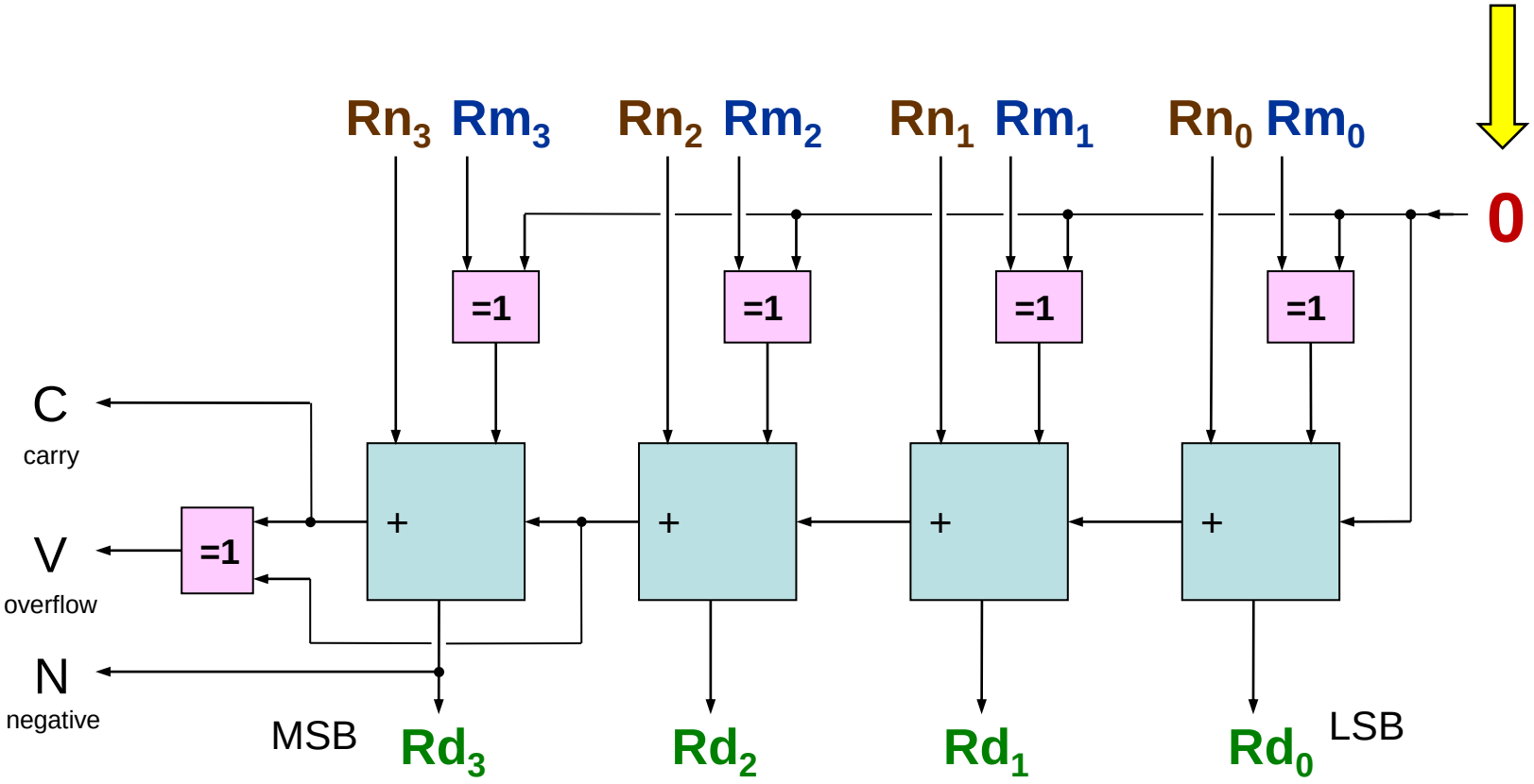
```
ADDS R1,R2,R3
```



Addition

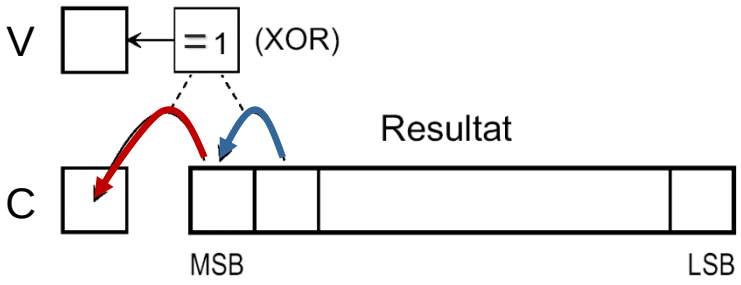
- **Addition in Hardware (ALU)**

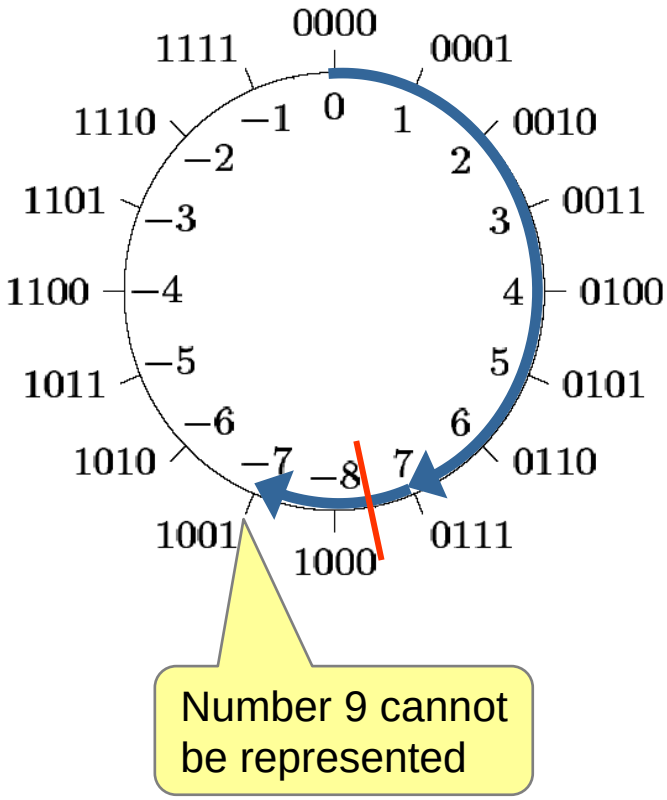
```
ADDS    Rd,Rn,Rm
```



Addition

- **Signed**
  - Overflow carry to MSB has different value than carry from MSB



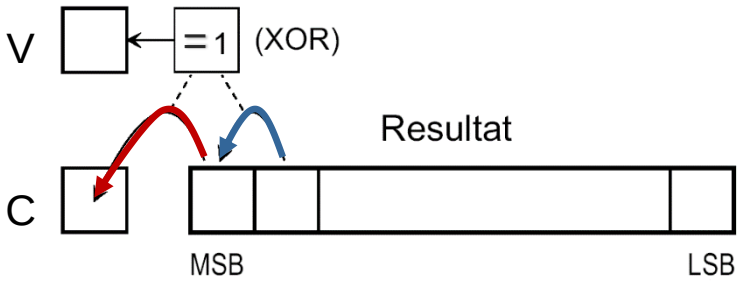


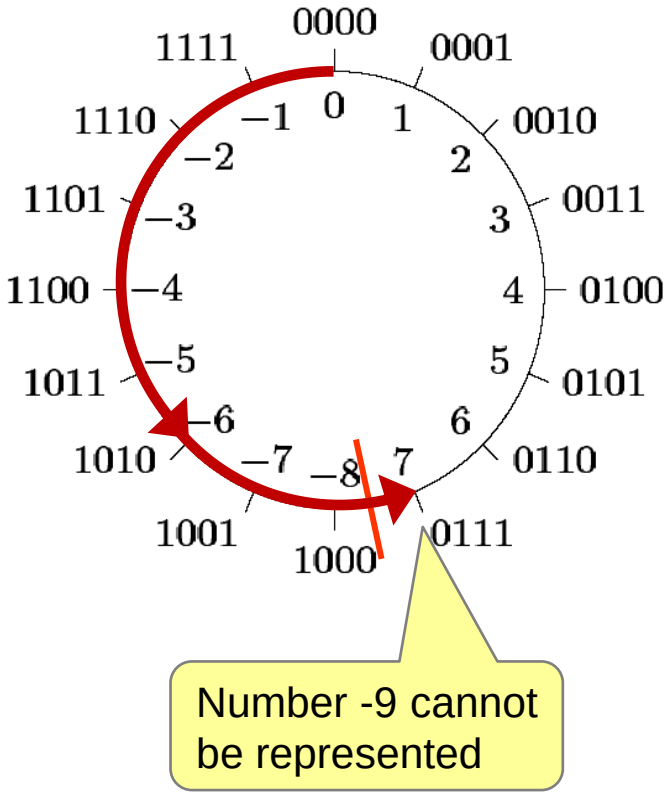
  - Examples for 4 cases

0 1	1 0	0 0	1 1
0111	1010	0001	0011
+0010	+1101	+1110	+1111
0 1 1 0	1 0 0 0	0 0 0 0	1 1 1 1
1001	0111	1111	0010
V = 1	V = 1	V = 0	V = 0

Addition

- **Signed**
  - Overflow carry to MSB has different value than carry from MSB





  - Examples for 4 cases

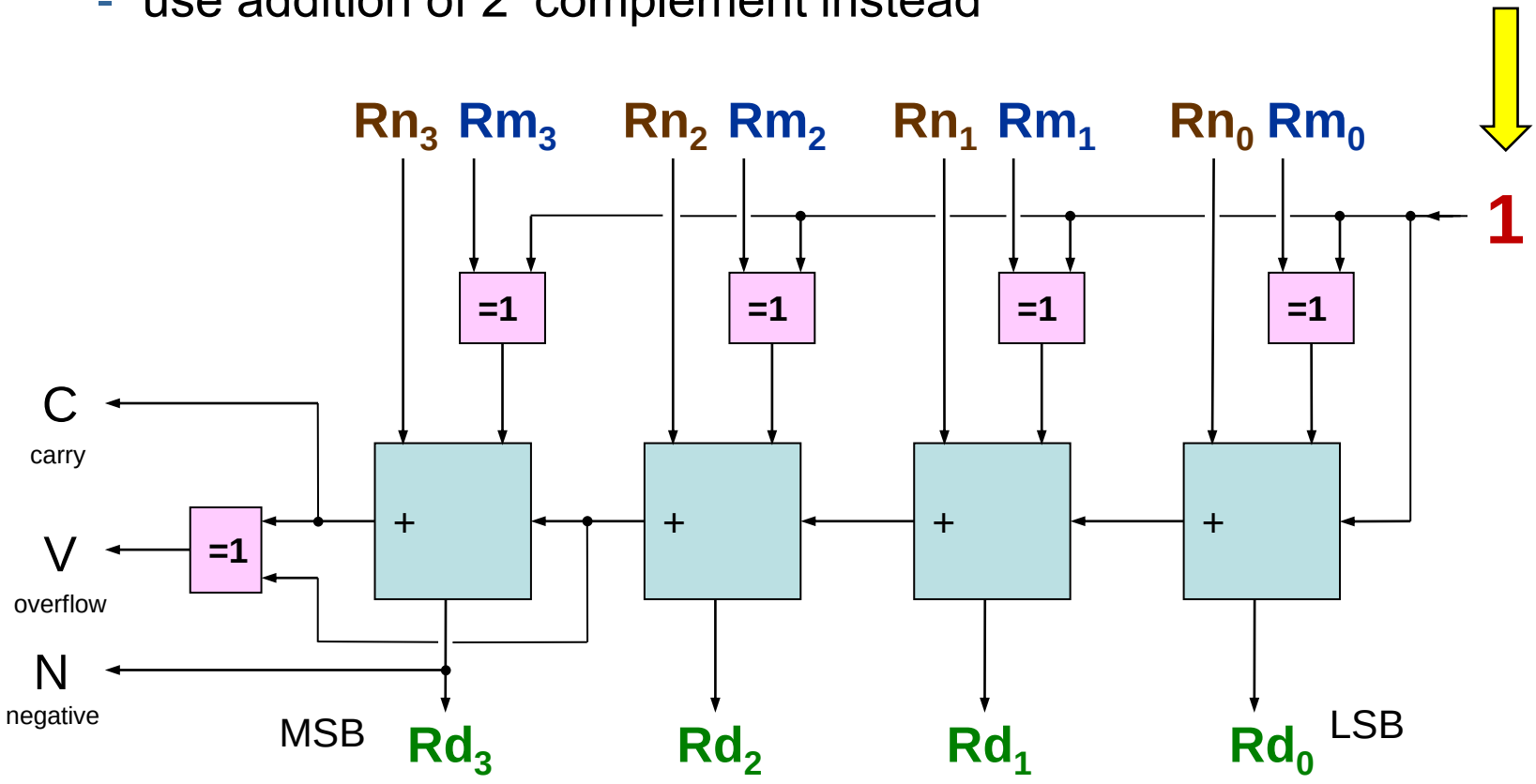
0 1	1 0	0 0	1 1
0111	1010	0001	0011
+0010	+1101	+1110	+1111
0 1 1 0	1 0 0 0	0 0 0 0	1 1 1 1
1001	0111	1111	0010
V = 1	V = 1	V = 0	V = 0

Subtraction

- **Subtraction in Hardware (ALU)**

```
SUBS    Rd,Rn,Rm
```

  - There is no subtraction!
    - use addition of 2‘ complement instead

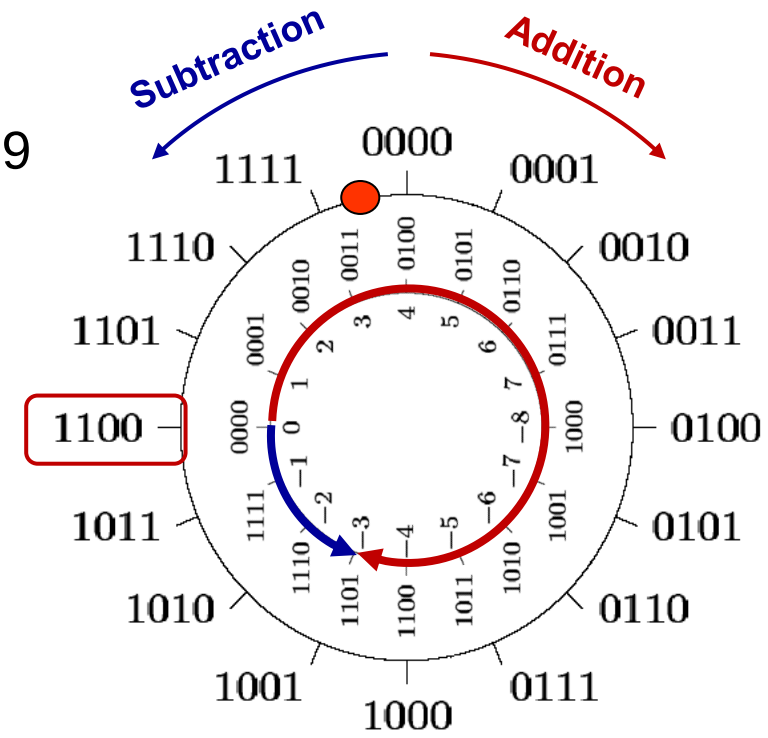


Subtraction

School of Engineering
InES Institute of Embedded Systems

- **unsigned**
  - Use 2‘ complement as well
  - Example 4-Bit unsigned: 12 - 3 = 9

12	1100	1100
- 3	- 0011	+ 1101
		1 1
9	1001	1 1001
	human	computer



  - Attention
    - C = 1 → NO borrow
    - C = 0 → borrow
  - Borrow
    - operation yields negative result → cannot be represented in unsigned
    - in multi-word operations missing digits are borrowed from more significant word

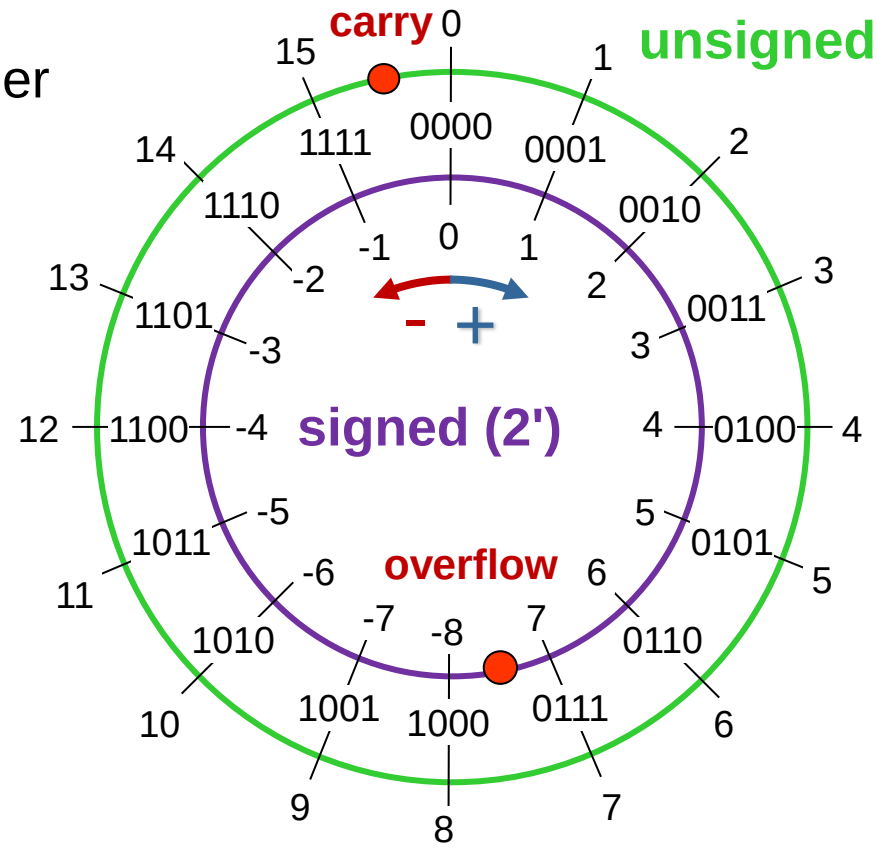
Subtraction

School of Engineering
InES Institute of Embedded Systems

- **Unsigned**
  - C = 0 indicates borrow
  - V irrelevant
  - Subtraction from a small number
    → can yield a big result
- **Signed**
  - V = 1 indicates overflow or underflow
    - Possible with opposite signs
    - NOT possible when operands have same sign
  - C irrelevant

Programmer interprets register content either as signed or as unsigned.
CPU does not care!

```
SUBS R1,R2,R3
```

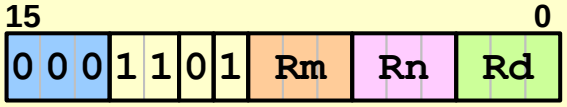


Subtraction Operations Cortex-M0

School of Engineering
InES Institute of Embedded Systems

- **SUBS (register)**
  - Updates flags
  - Result and 2 operands
  - Only low register

```
SUBS <Rd>,<Rn>,<Rm>
```




```
Rd = Rn - Rm
   = Rn + NOT(Rm) + 1
```

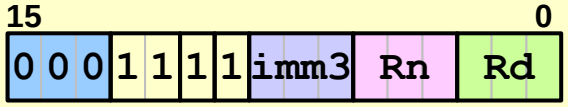
```
00000016 1AD1     SUBS      R1,R2,R3
00000018 1B64     SUBS      R4,R4,R5
0000001A 1B64     SUBS      R4,R5        ; the same (dest = R4)
0000001C
0000001C          ;SUBS     R8,R4,R5     ; not possible (high reg)
0000001C          ;SUBS     R4,R8,R5     ; not possible (high reg)
0000001C          ;SUBS     R4,R5,R8     ; not possible (high reg)
```

Subtraction Operations Cortex-M0

School of Engineering
InES Institute of Embedded Systems

- **SUBS (immediate) – T1**
  - Updates flags
  - 2 different low registers and immediate value 0 - 7d

```
SUBS <Rd>,<Rn>,#<imm3>
```




```
Rd = Rn - <imm3>
   = Rn + NOT<imm3> + 1
```

```
0000001C 1F63     SUBS      R3,R4,#5
0000001E          ;SUBS     R3,R4,#8     ; out of range immediate
0000001E          ;SUBS     R10,R11,#5   ; not possible (high reg)
```

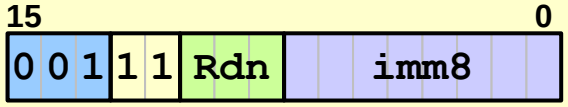
Subtraction Operations Cortex-M0

School of Engineering
InES Institute of Embedded Systems

- **SUBS (immediate) – T2**
  - Updates flags
  - One low register and immediate value 0 - 255d
  - <Rdn> → Result and operand

    → same register for result and operand

```
SUBS <Rdn>,#<imm8>
```




```
Rdn = Rdn – <imm8>
    = Rdn + NOT<imm8> + 1
```

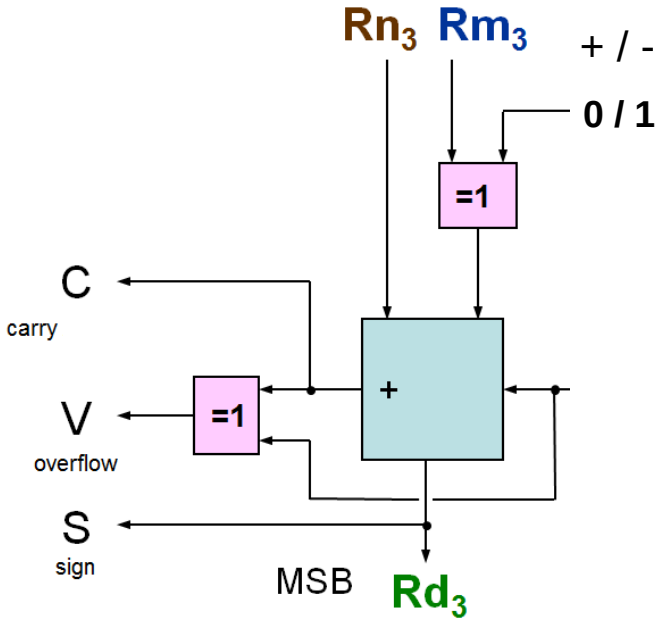
```
0000001E 3BF0     SUBS     R3,R3,#240
00000020 3BF0     SUBS     R3,#240      ; the same (dest = R3)
00000022          ;SUBS    R8,R8,#240   ; not possible (high reg)
00000022          ;SUBS    R3,#260      ; out of range immediate
```

Addition / Subtraction

School of Engineering
InES Institute of Embedded Systems

- **Interpretation of carry flag**
  - Addition C = 1 → Carry

```
   1  1  0  1    13d
   0  1  1  1     7d
1  1  1  1
-------------------------------------
1  0  1  0  0    20d → 16d + 4d
```



  - Subtraction C = 0 → Borrow

```
6d - 14d = 0110b - 1110b = 0110b + 0010b

   0  1  1  0     6d
   0  0  1  0     2d = TC(14d)
0  1  1  0
-------------------------------------
0  1  0  0  0     8d → - 16d + 8d
```

Addition / Subtraction

School of Engineering
InES Institute of Embedded Systems

- **unsigned** Interpretation
  - Program must check **carry flag (C)** after operation
  - **C = 1 for Addition** **C = 0 for Subtraction**
    - Result cannot be represented (not enough digits / no negative numbers)
    - Full turn on number circle must be added or subtracted
      → odometer effect
  - Overflow flag (V) irrelevant
- **signed** Interpretation
  - Program must check **overflow flag (V)** after operation
  - **V = 1** means
    - Not enough digits available to represent the result
    - Full turn on number circle must be added or subtracted
      → odometer effect
  - Carry flag (C) irrelevant

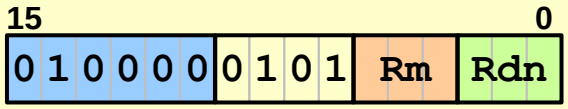
Multi-Word Arithmetic

School of Engineering
InES Institute of Embedded Systems

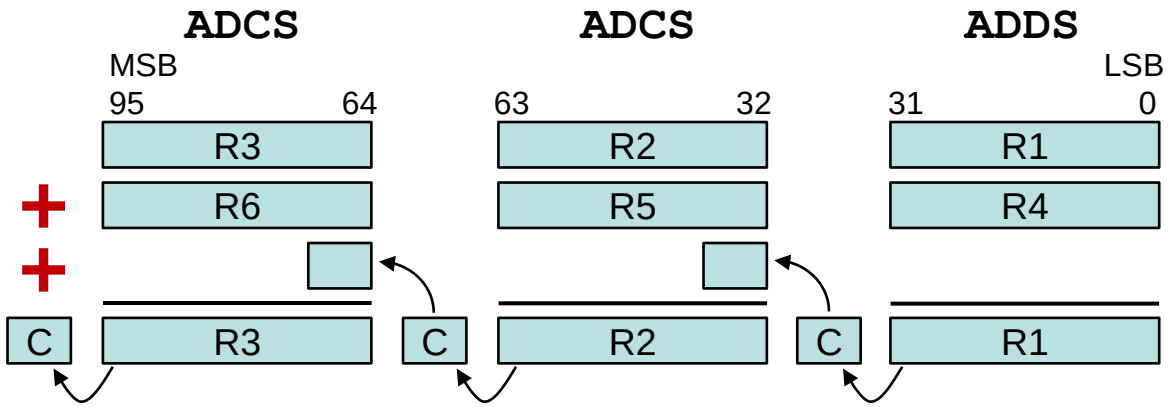
Multi-Word Addition ADCS

- Example: Addition of two 96-bit Operands

ADCS <Rdn>,<Rm>



Rdn = Rdn + Rm + C




```
; Operand A:          96-bit in R3(MSW),R2,R1(LSW)
; Operand B:          96-bit in R6(MSW),R5,R4(LSW)
; Result = A + B:    96-bit in R3(MSW),R2,R1(LSW)
00000028 1909     ADDS     R1,R1,R4
0000002A 416A     ADCS     R2,R2,R5
0000002C 4173     ADCS     R3,R3,R6
```

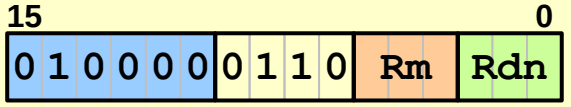
Multi-Word Arithmetic

School of Engineering
InES Institute of Embedded Systems

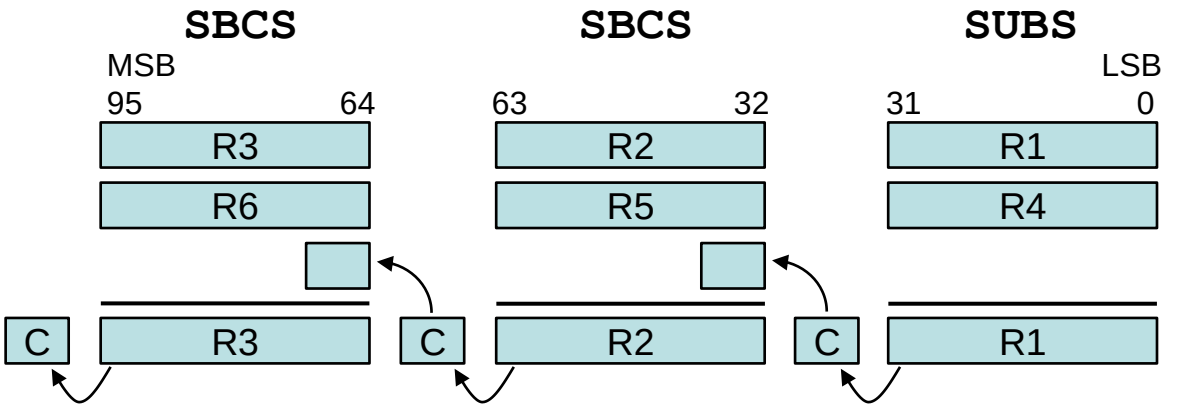
Multi-Word Subtraction SBCS

- Example: Subtraction of two 96-bit Operands

```
SBCS <Rdn>,<Rm>
```




```
Rdn = Rdn - Rm - NOT(C)
    = Rdn + NOT(Rm) + C
```
1)




```
; Operand A:     96-bit in R3(MSW),R2,R1(LSW)
; Operand B:     96-bit in R6(MSW),R5,R4(LSW)
; Result A - B:  96-bit in R3(MSW),R2,R1(LSW)
0000002E 1B09    SUBS     R1, R1, R4
00000030 41AA    SBCS     R2, R2, R5
00000032 41B3    SBCS     R3, R3, R6
```

1) -Rm - NOT(C)
= (NOT(Rm) + 1) - NOT(C)
= NOT(Rm) + C

The CPU actually calculates Rdn + NOT(Rm) + C

Start 23.10.2023

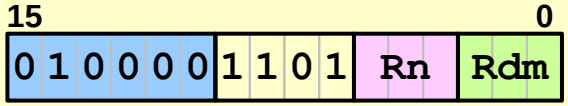
School of Engineering
InES Institute of Embedded Systems

Multiplication

School of Engineering
InES Institute of Embedded Systems

- **MULS (register)**
  - Flags
    - N and Z updated
    - C and V unchanged
  - Operands
    - Rn multiplicand
    - Rdm multiplier
    - only low registers
  - Result
    - **Rdm contains only lowest 32 bits of product**

```
MULS <Rdm>,<Rn>,<Rdm>
```




```
Rdm = Rn * Rdm
```

```
0000002E 4351   MULS      R1,R2,R1

00000030        ;MULS     R1,R1,R2 ; not possible: destination and
00000030                            ; 2nd source must be same
00000030        ;MULS     R1,R8,R1 ; not possible: high reg
```

Multiplication

- **Result requires twice as many binary digits**
  - Example 4-bit * 4-bit → 8-bit result
- **Signed and unsigned multiplication are different**

```
0101 * 0011
     0011
    0000
   0011
  0000
--------
 00001111
```

unsigned

```
0101 * 1101
   00001101
  00000000
 00001101
00000000
--------
000001000001
```

zero extension of multiplier

signed

```
0101 * 1101
   11111101
  00000000
 11111101
00000000
--------
100111110001
```

sign extension of multiplier

Interpretation unsigned
5d * 3d = 15d → correct

Interpretation signed
5d * 3d = 15d → correct

Interpretation unsigned
5d * 13d = 65d → correct

Interpretation signed
5d * – 3d = 65d → wrong

Interpretation unsigned
5d * 13d = 241d → wrong

Interpretation signed
5d * – 3d = – 15d → correct

Multiplication

School of Engineering
InES Institute of Embedded Systems

- **MULS on Cortex-M0**
  - Rn (32-bit) * Rdm (32-bit) → Rdm (32-bit)
    - Upper 32-bit of result are lost!
    - Lower 32-bit are the same for unsigned and signed
  - unsigned → watch out if result requires more than 32 bits
  - signed → part with sign bit is missing

Conclusion

School of Engineering
InES Institute of Embedded Systems

- **Processor Arithmetic**
  - Odometer effect because of finite word length
- **Processors do not distinguish signed and unsigned**
  - User (resp. compiler) has to know, whether he is working with signed or unsigned numbers
  - Processor always calculates flags for both cases
    - carry (C) unsigned
    - overflow (V) signed
      negative (N)

Conclusion

School of Engineering
InES Institute of Embedded Systems

- **unsigned**
  - Addition → C = 1 → carry
    result too large for available bits
  - Subtraction → C = 0 → borrow
    result less than zero → no negative numbers
- **signed**
  - Addition → potential overflow in case of operands with equal signs
  - Subtraction → potential overflow in case of operands with opposite signs
- **Arithmetic Instructions**
  - **`ADD/ADDS`** **`ADCS`** **`ADR`**
    `SUB/SUBS` **`SBCS`** **`RSBS`**
    `MULS`