Modular Arithmetic

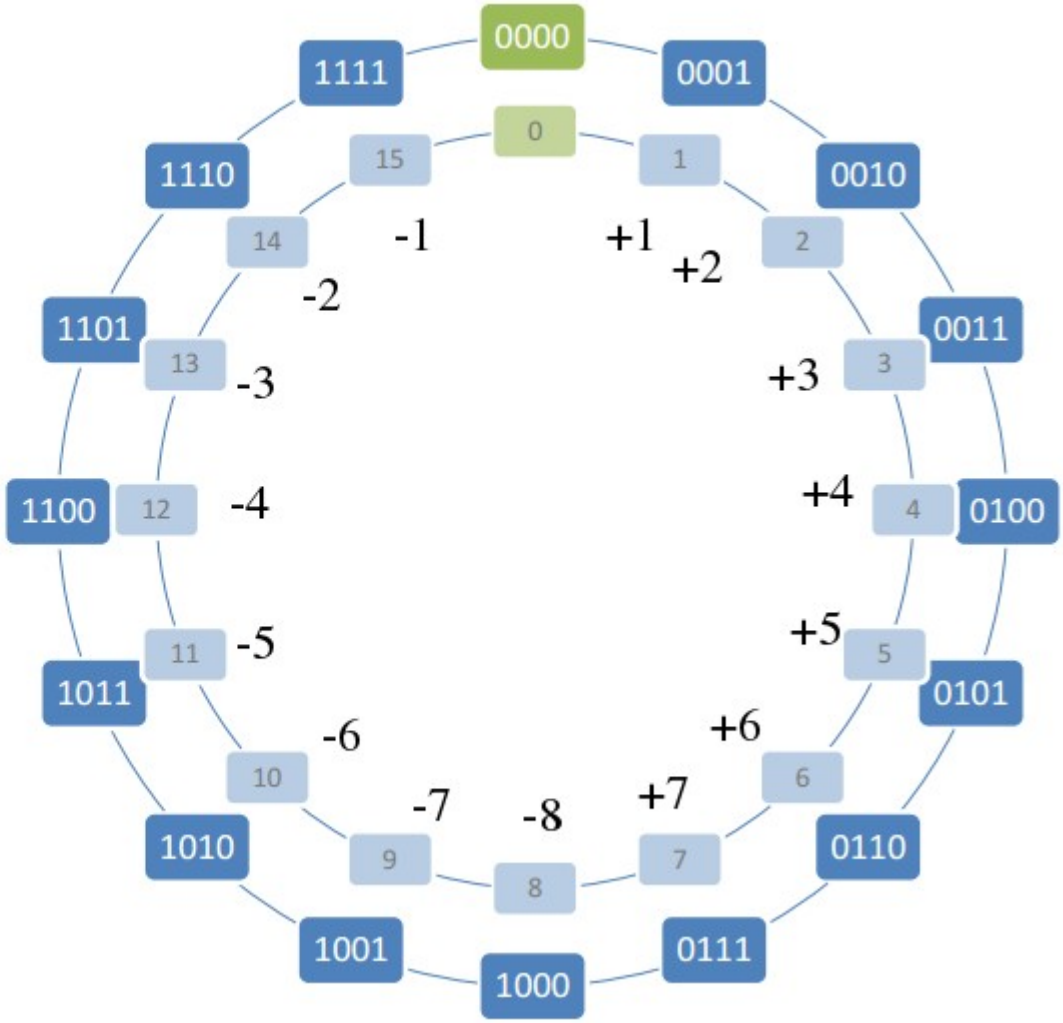
In mathematics, modular arithmetic is a system of arithmetic for integers, where numbers "wrap around" when reaching a certain value, called the modulus.

A familiar use of modular arithmetic is in the 12-hour clock, in which the day is divided into two 12-hour periods. If the time is 7:00 now, then 8 hours later it will be 3:00. Simple addition would result in 7 + 8 = 15, but clocks "wrap around" every 12 hours. Because the hour number starts over at zero when it reaches 12, this is arithmetic modulo 12.

A number x mod N is the equivalent of asking for the remainder of x when divided by N. Two integers a and b are said to be congruent (or in the same equivalence class) modulo N if they have the same remainder upon division by N

- A 4-bit number system allows us to represent 16 values
- Ignoring carries from addition gives us modulo-16 arithmetic

- (**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



The two's complement of an N-bit number is defined as the result of subtracting the number from 2^N

So two's complement of 3 is 16 - 3 = 13

So

5 + -3

(5 + 16 – 3) Mod 16

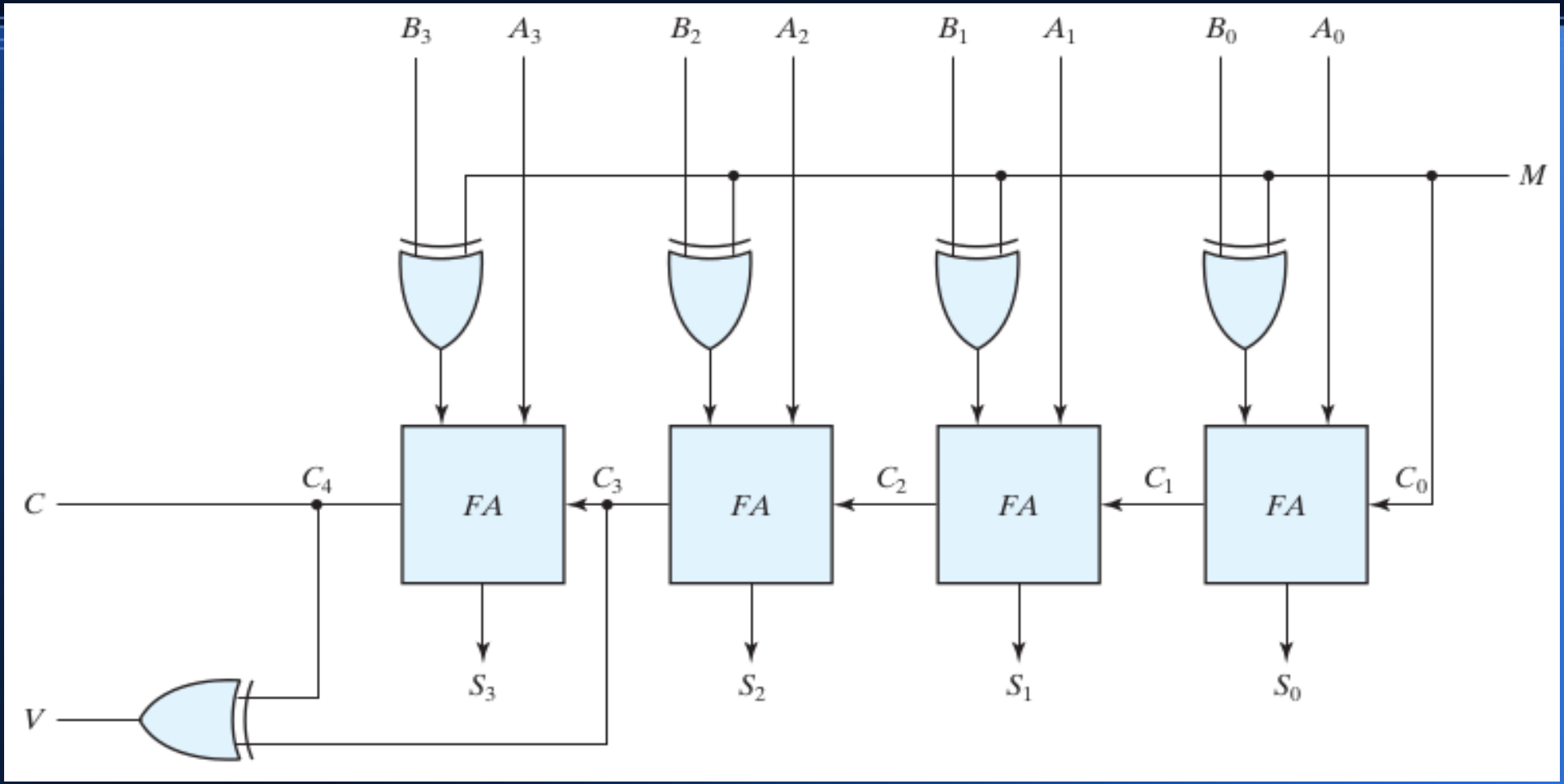
(5 -3) Mod 16

2

The two's complement of an N-bit number is defined as the complement with respect to 2^N, in other words the result of subtracting the number from 2^N.

This is also equivalent to taking the ones' complement and then adding one, since the sum of a number and its ones' complement is all 1 bits.

Four-bit adder–subtractor



```
QEMU 4.2.1 monitor - type 'help' for more information
(qemu) info registers
R00=00000000 R01=00000000 R02=00000000 R03=00000000
R04=00000000 R05=00000000 R06=00000000 R07=00000000
R08=00000000 R09=00000000 R10=00000000 R11=00000000
R12=00000000 R13=00000000 R14=00000000 R15=00000000
PSR=400001d3 -Z-- A svc32
FPSCR: 00000000
(qemu)
```

Condition Code Flags

Current Program Status Register

Condition Code Flags

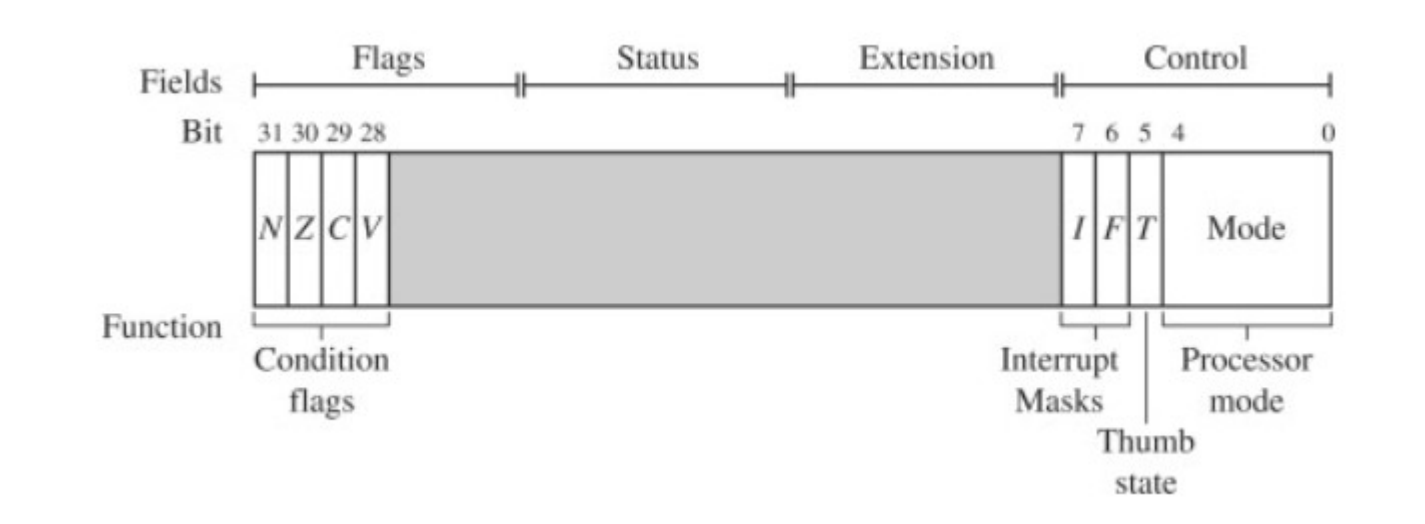
- Some instructions can optionally update the **Condition Code Flags** to provide information about the result of the execution of the instruction
  - e.g. whether the result of an addition was zero, or negative or whether a carry occurred

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

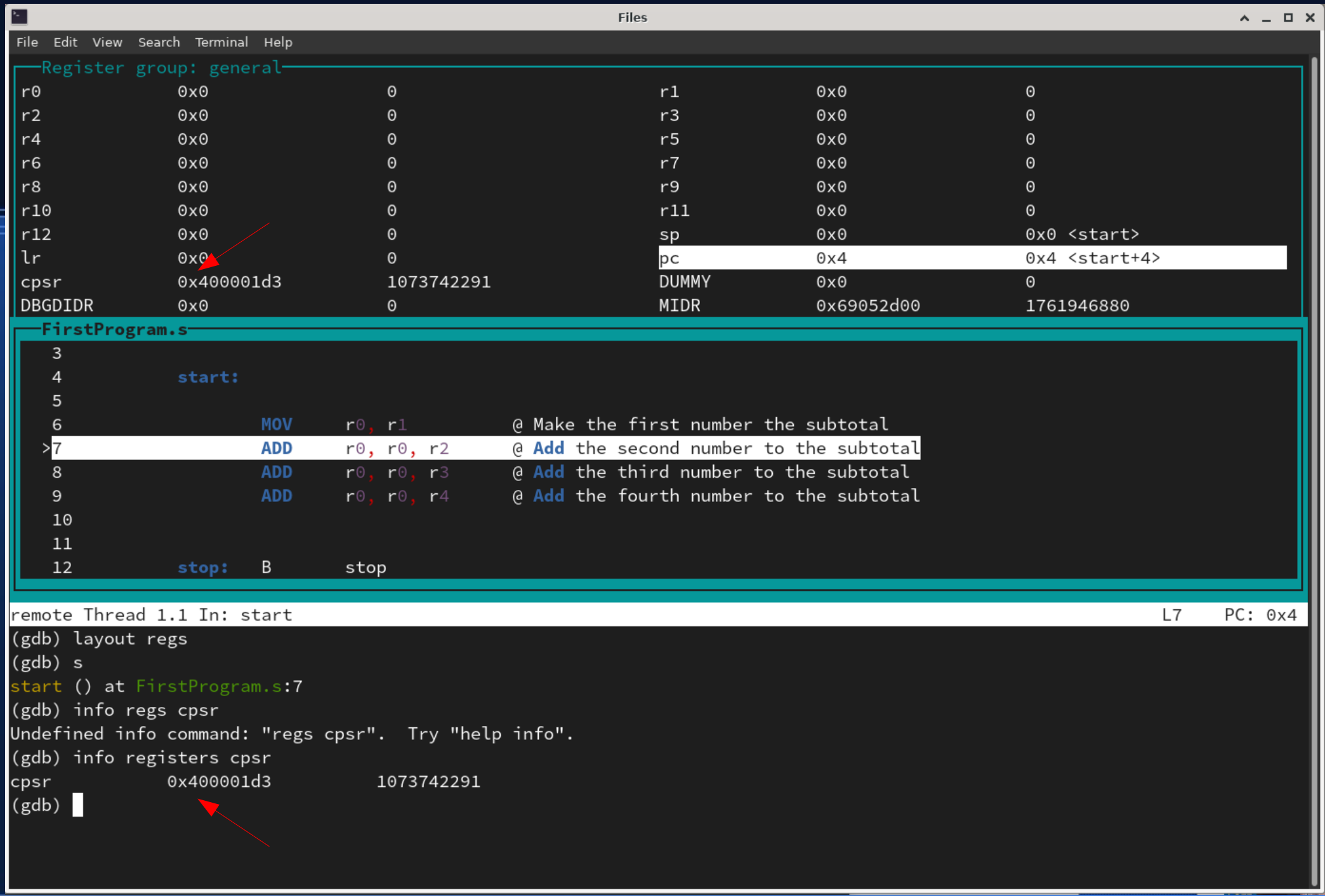
```
(gdb) info registers cpsr
cpsr           0x400001d3       1073742291
```

CPUlator flags display

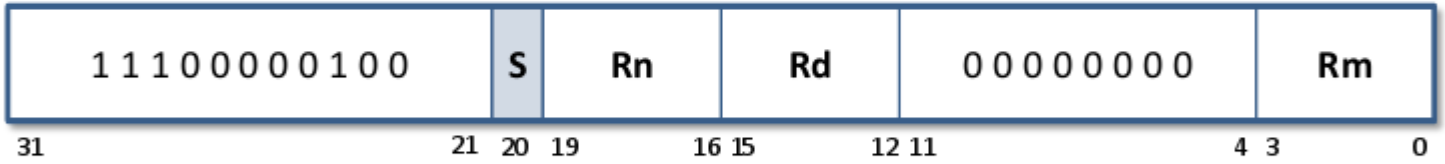
sp	00000000	
lr	00000000	
pc	00000004	
cpsr	400001d3	NZCVI SVC
spsr	00000000	NZCVI ?
s0	00000000	



There are seven processor modes in total: six privileged modes (abort, fast interrupt request, interrupt request, supervisor, system, and undefined) and one nonprivileged mode (user).


Files
File Edit View Search Terminal Help
Register group: general
r0 0x0 0 r1 0x0 0
r2 0x0 0 r3 0x0 0
r4 0x0 0 r5 0x0 0
r6 0x0 0 r7 0x0 0
r8 0x0 0 r9 0x0 0
r10 0x0 0 r11 0x0 0
r12 0x0 0 sp 0x0 0x0 <start>
lr 0x0 0 pc 0x4 0x4 <start+4>
cpsr 0x400001d3 1073742291 DUMMY 0x0 0
DBGDIDR 0x0 0 MIDR 0x69052d00 1761946880
FirstProgram.s
3
4 start:
5
6 MOV r0, r1 @ Make the first number the subtotal
>7 ADD r0, r0, r2 @ Add the second number to the subtotal
8 ADD r0, r0, r3 @ Add the third number to the subtotal
9 ADD r0, r0, r4 @ Add the fourth number to the subtotal
10
11
12 stop: B stop
remote Thread 1.1 In: start L7 PC: 0x4
(gdb) layout regs
(gdb) s
start () at FirstProgram.s:7
(gdb) info regs cpsr
Undefined info command: "regs cpsr". Try "help info".
(gdb) info registers cpsr
cpsr 0x400001d3 1073742291
(gdb)

- 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. **ADD** instruction
  - Condition Code Flags only updated if **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**)

A Simple Formula for Overflow

Let the carry out of the full adder adding the least significant bit be called c0. Then, the carry out of the full adder adding the next least significant bit is c1. Thus, the carry out of the full adder adding the most significant bits is c(k - 1). This assumes that we are adding two k bit numbers. We can write the formula as:

V = c(k-1) XOR c(k-2)

This is effectively XORing the carry-in and the carry-out of the leftmost full adder. Why does this work? The XOR of the carry-in and carry-out differ if there's either a 1 being carried in, and a 0 being carried out, or if there's a 0 being carried in, and a 1 being carried out.

For addition, the microprocessor follows the basic rules of binary addition, for subtraction complement subtrahend (invert bits and add 1) and perform an addition.

Carry Flag

addition – easy to perform addition
subtraction – complement subtrahend and add

Overflow Flag when adding two signed numbers of different sign, (or subtracting two of same sign) overflow cannot occur V bit 0. The sign bit is the most significant bit.

Adding two signed numbers of same sign (or subtracting two or different sign) and getting a sign change in the result V bit 1.

A3	B3	C3	S3	C4	V
0	0	0	0	0	0
0	0	1	1	0	1
0	1	0	1	0	0
0	1	1	0	1	0
1	0	0	1	0	0
1	0	1	0	1	0
1	1	0	0	1	1
1	1	1	1	1	0

When does that happen? Let's look at each case:

Case 1: 0 carried in, and 1 carried out
If a 0 is carried in, then the only way that 1 can be carried out is if x(k-1) = 1 and y(k-1) = 1. That way, the sum is 0, and the carry out is 1.
This is the case when you add two negative numbers, but the result is non-negative.

Case 1: 1 carried in, and 0 carried out
The only way 0 can be carried out if there's a 1 carried in is if x(k-1) = 0 and y(k-1) = 0. In that case, 0 is carried out, and the sum is 1.
This is the case when you add two non-negative numbers and get a negative result.

(0+1=0 and 0+1=1 carry in = carry out)

BHI	$C\bar{Z}$	unsigned >
BLS	$\bar{C} + Z$	unsigned ≤

5 0101
4 0100

5-4
0101
1011
C 1111
0001

4-5
0100
1010
1
1111

5-5
0101
1010
1111
0000

BCC / BLO	$\bar{C}$	unsigned <

```
Question 2.3
;;
;; After execution of the following instructions
;; what value will be in the condition code flags?
;;
00 0201A0E3          MOV   r0, #0x80000000
04 0111A0E3          MOV   r1, #0x40000000
08 007051E0          SUBS  r7, r1, r0

(A) 0x1 (B) 0x8 (C) 0x0
(D) 0x9 (E) 0xB (F) OTHER
```

4 - (-8) = -4 !!

$$\begin{array}{r} 0100 \\ 1000 \\ \hline 1100 \end{array} \qquad \begin{array}{r} 0100 \\ 0111 \\ \hline 1 \end{array}$$

N Z C V
1 0 0 1

is 0100 > 1000 unsigned no, no carry is unsigned <
signed yes, V and N is signed >=

(Un-) Conditional Branch Instructions

Branch Instruction	Condition Code Flag Evaluation	Description
B (or BAL)	don't care	unconditional (branch always)
BEQ	Z	equal
BNE	$\bar{Z}$	not equal
BCS / BHS	C	unsigned ≥
BCC / BLO	$\bar{C}$	unsigned <
BMI	N	negative
BPL	$\bar{N}$	positive or zero
BVS	V	overflow
BVC	$\bar{V}$	no overflow
BHI	$C\bar{Z}$	unsigned >
BLS	$\bar{C} + Z$	unsigned ≤
BGE	$NV + \bar{N}\bar{V}$	signed ≥
BLT	$N\bar{V} + \bar{N}V$	signed <
BGT	$\bar{Z}(NV + \bar{N}\bar{V})$	signed >
BLE	$Z + N\bar{V} + \bar{N}V$	signed ≤

Consider a – b, if the result is not a positive number and there was no overflow then a is greater than or equal to b. (for greater than include zero flag)
i.e. -2 - (-3) = 1 but -3 - (-2) = -1

If the overflow is set the N bit is “wrong”, ie
the opposite of what is should be because we don’t have enough bits

BGE	$NV + \bar{N}\bar{V}$	signed ≥

This is also sometimes written as N==V

BLT	$N\bar{V} + \bar{N}V$	signed <

Is sometimes written as N!=V. Confusingly there was also a BNV instruction which stood for branch never or NOP, nothing to do with the flags and now deprecated.