Microprocessor Systems

Lecture 6

The carry flag

- The carry flag is used in many arithmetic and logic instructions.
  - The use of the carry flag is best illustrated by looking at addition.
- Consider a 32 bit processor such as the ARM7.
  - If the sum of two numbers is greater than 0xFFFFFFFF (= 4,294,967,29510) then
  - the sum will have more than 32 bits and it cannot be fitted into a 32 bit register.

Setting the carry flag

- The carry flag will be set if the sum is greater than 0xFFFFFFFF using the instruction ADDS.
- Hence the following instructions will set the carry flag:

```
MOV r2, 0xF2000000
MOV r7, 0x11000000
ADDS r9, r2, r7
```

- The sum should be 0x103000000 or $100000011000000000000000000000000_2$ but the register can not hold the most significant bit.

Question

- What happens to the zero and carry flags after each addition in the following?

  MOV r5, #0xFFFFFFFF

  ADDS r4, r5, #0 ;add 0 to r5

  ADDS r4, r5, #1 ;add 1 to r5

  ADDS r4, r5, #2 ;add 2_{10} to r5

Answer

- ADDS r4, r5, #0
  - r4 holds the value 0xFFFFFFFF and both the zero flag, Z, and the carry flag, C, are cleared.
- ADDS r4, r5, #1
  - r4 holds the value 0x00000000 and both the zero flag, Z, and the carry flag, C, are set.
- ADDS r4, r5, #2
  - r4 holds the value 0x00000001, the zero flag, Z, is cleared and the carry flag, C, is set.

Use of the carry flag

- The carry flag can be used in two ways:
  – In common with the other flags it can be used to determine if a conditional instruction is executed or not.
    - E.g.
      – ADDCS r1, r1, #1 will only execute if the carry flag is set and
      – 'BCC label' will only branch if the carry flag is clear.
  – The other use of the carry flag is in the addition instruction ADC (and the subtraction instructions SBC and RSC).

Add with carry

- The instruction ADC ‘add with carry’ adds together the two values and adds another 1 if the carry flag is set.
  - E.g.
    - ADC r0, r1, #3 ;
      - with value in r1 equal to 2
      - if carry flag is clear the sum in r0 is 5 (=3 + 2) but
      - if the carry flag is set the sum in r0 is 6.

Using add with carry

- ADC is used when we add together numbers greater than $(2^{32}-1)$
  - e.g.
    - $12{,}000{,}000{,}000_{10}$ added to $14{,}000{,}000{,}000_{10}$ which in hexadecimal is 0x2CB417800 added to 0x342770C00.
    - Both numbers are 34 bits in length and each one can be stored in two registers.
    - E.g.
      - r0 could hold the value 0xCB417800 and r1 could hold the value 0x00000002 for $12{,}000{,}000{,}000_{10}$

Using add with carry

- Example:
  - r0 holds 0xCB417800 and r1 holds 0x00000002
  - r2 holds 0x42770C00 and r3 holds 0x00000003

    ADDS r4, r2, r0
  - The sum of r2 and r0 is greater than 0xFFFFFFFF
  - so the carry flag is set and r4 holds the lowest 32 bits: 0x0DB88400.

Using add with carry

- Next instruction:

  ADC r5, r3, r1

- Because the carry flag is set, an extra 1 is added into the sum so r5 will hold 0x00000006.
- Taken together r5 and r4 hold the value 0x60DB88400 which is $26,000,000,000_{10}$

Negative numbers

- There are two main methods for representing negative numbers in microprocessors.
- These are:
  1) Sign magnitude.
  2) Two's complement.
- In each case the most significant bit - 'm.s.b.' - indicates the sign (1 for -ve and 0 for +ve).

The sign bit

- If the m.s.b. or ‘sign bit’ is 1 the number is -ve and if the m.s.b. is 0 the number is +ve.
- To find out if a number is -ve or +ve we could use:

  MOVS rx, rx

- The value in register rx remains unchanged but the negative flag is set if the m.s.b. is 1 and it is cleared if the m.s.b. is 0.

Sign magnitude

- Using the sign magnitude method a negative number is the same as a positive number but with the m.s.b. or ‘sign bit’ equal to 1.
- E.g. in 16 bits:

Decimal	Binary	Hexadecimal
+160	0000000010100000	0x00A0
-160	1000000010100000	0x80A0
+20640	0101000010100000	0x50A0
-20640	1101000010100000	0xD0A0

Sign magnitude

- In general any hexadecimal number is negative if it starts with 8 or greater and it is positive if it starts with 7 or less.
- So in 32 bits the number 0x800050A0 is negative and 0x000050A0 is positive using the sign magnitude method.
- The magnitude of a 'sign magnitude' number is easy to find:
  - simply AND with 0x7FFFFFFF.

Sign magnitude

- However sign magnitude numbers cannot be used for arithmetic.
- For example: 3 + (-3) should be 0.

Decimal	Hexadecimal
3	0x00000003
+(-3)	+ 0x80000003
?	0x80000006

- The answer is -6 rather than 0 !

2's complement

- In the two's complement method a negative number, -x, is given by the value (2^n - x) for an n bit processor.
- For example -3 in a 32 bit processor is:

$$\begin{array}{r} \text{0x100000000} \\ -3 \\ \hline \text{0xFFFFFFFD} \end{array}$$

- So 0xFFFFFFFD is the 2's complement representation of -3 in a 32 bit processor.

2's complement

- The two's complement method automatically sets the m.s.b. or 'sign bit' to 1.
- The following method can be used to find a 2's complement representation of a negative number,
  - e.g.
    - -20640
    - First find the positive value: 0x000050A0 or $0000\ 0000\ 0000\ 0000\ 0101\ 0000\ 1010\ 0000_2$
    - Next invert all bits ($0 \rightarrow 1$, $1 \rightarrow 0$).
      - $1111\ 1111\ 1111\ 1111\ 1010\ 1111\ 0101\ 1111_2$ or
      - 0xFFFFAF5F.
    - And then add 1 ➔0xFFFFAF60

2's complement

- So 0xFFFFAF60 is the 2's complement representation of -2064010.
- The inversion of bits can be implemented in hexadecimal rather than binary as follows:
  - Inverted No: F E D C B A 9 8 7 6 5 4 3 2 1 0
  - Original No: 0 1 2 3 4 5 6 7 8 9 A B C D E F

Question

- What is the two's complement of the following numbers in 32 bits?

  $-1{,}500{,}000{,}000_{10}$

  $(1{,}500{,}000{,}000_{10} = \text{0x59682F00})$

  -114_{10} $(114_{10} = \text{0x00000072})$

  -2006_{10} $(2004_{10} = \text{0x000007D6})$

  – Inverted No: F E D C B A 9 8 7 6 5 4 3 2 1 0
  – Original No: 0 1 2 3 4 5 6 7 8 9 A B C D E F

Answer

- First invert 0x59682F00 to find 0xA697D0FF and
  - then add 1
  - so -1,500,000,000$_{10}$ = 0xA697D100
- Invert 0x00000072 to find 0xFFFFFF8D and
  - then add 1
  - so -114$_{10}$ = 0xFFFFFF8E
- Invert 0x000007D6 to find 0xFFFFF829 and
  - then add 1
  - so -2006$_{10}$ = 0xFFFFF82A

2’s complement

- This method also works in reverse so if you have a two’s complement number
  - e.g.
    - 0xFFFFFF60 and you want to know it’s value in decimal.
    - First the sign bit is 1
    - so it is a negative number.
    - Therefore invert and add 1:
    - 0x0000009F + 0x00000001 = 0x000000A0
    - which is 160 in decimal.
    - Hence 0xFFFFFF60 is the 2’s complement representation for -160_{10}

2's complement

- Unlike sign magnitude, arithmetic is simple in two's complement e.g. in 8 bits

Decimal	Binary
3	00000011
+(-3)	+ 11111101
0	1 00000000

- Note that if the carry bit (9th bit) is ignored the answer is 0 which is correct.

3C 41 52 4D 2E 50 4F 57 45 52 45 44 3E 0D 0A