EE 319K
Introduction to Embedded Systems

Lecture 3: Debugging, Arithmetic Operations, Condition Code Bits, Conditionals

Andreas Gerstlauer

Announcements

- Lab lectures
  - Start this week
- Lab
  - First lab due this week (today & tomorrow)
    - Digital lock in TExaS
  - Second lab due next week
    - LED and switch interface in TExaS
    - You may select a different partner
- Homework
  - Second homework due next Monday

Agenda

- Recap
  - 9S12 execution
  - Input/output
  - Logical/shift operations
  - Introduction to C
- Outline
  - Debugging
  - Computer arithmetic
    - Basic arithmetic operations: addition & subtraction
    - Arithmetic condition codes
    - Conditionals: basic if-then-else

Debugging

- Aka: Testing, Diagnostics, Verification
- Debugging Actions
  - Functional debugging, input/output values
  - Performance debugging, input/output values with time
  - Tracing, measure sequence of operations
  - Profiling,
    - measure percentage for tasks,
    - time relationship between tasks
  - Performance measurement, how fast it executes
  - 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, BDM
- Minimally intrusive
  - negligible effect on the system being debugged
  - e.g., dumps(ScanPoint) and monitors
- Highly intrusive
  - print statements, breakpoints and single-stepping

Debugging Aids in TExas

Interface

- uc – ViewBox, BreakPoints
- stk – StackField, MemoryBox
- Modes
  - FollowPC
  - CycleView
  - InstructionView
  - LogRecord
- Single Step, Few, StepOver, StepOut, Run
- Breakpoint versus ScanPoint

Run time errors

- Read from un-programmed ROM
- Write to ROM
- Read from un-initialized RAM
- Read/write unimplemented I/O

... 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)
    - TExaS does not support conditional assembly
    - Easy to remove all instruments
- Visualization: How the debugging information is displayed
- We will look at the following examples*
  - **`Simple_DP512asm.zip`**
  - **`Square_DP512asm.zip`**

*: Downloadable from Jon's site: http://users.ece.utexas.edu/~valvano/Starterfiles/

Ramesh Yerraballi

Arithmetic Operations

- **Question 1.** How many bits does it take to store the result of two unsigned 8-bit numbers added together?
- ***Question 2.*** *How many bits does it take to store the result of two unsigned 8-bit numbers subtracted?*
- **Question 3.** How many bits does it take to store the result of two unsigned 8-bit numbers multiplied together?

… Arithmetic Operations

- **Question 4.** How many bits does it take to store the result of two unsigned 8-bit numbers added together?
  - 0 + 0 = 0, 255+255 = 510   0 to 510 is 9 bits
- ***Question 5.*** *How many bits does it take to store the result of two unsigned 8-bit numbers subtracted?*
  - 0 - 255 = -255, 255-0 = 255   -255 to +255 is 9 bits
- **Question 6.** How many bits does it take to store the result of two unsigned 8-bit numbers multiplied together?
  - 0 * 0 = 0, 255*255 = 65025   0 to 65025 is 16 bits

Add and Subtract Operations

Table 5-4. Addition and Subtraction Instructions

Mnemonic	Function	Operation
Addition Instructions		
ABA	Add B to A	(A) + (B) ⇒ A
ABX	Add B to X	(B) + (X) ⇒ X
ABY	Add B to Y	(B) + (Y) ⇒ Y
ADCA	Add with carry to A	(A) + (M) + C ⇒ A
ADCB	Add with carry to B	(B) + (M) + C ⇒ B
ADDA	Add without carry to A	(A) + (M) ⇒ A
ADDB	Add without carry to B	(B) + (M) ⇒ B
ADDD	Add to D	(A:B) + (M : M + 1) ⇒ A : B
Subtraction Instructions		
SBA	Subtract B from A	(A) – (B) ⇒ A
SBCA	Subtract with borrow from A	(A) – (M) – C ⇒ A
SBCB	Subtract with borrow from B	(B) – (M) – C ⇒ B
SUBA	Subtract memory from A	(A) – (M) ⇒ A
SUBB	Subtract memory from B	(B) – (M) ⇒ B
SUBD	Subtract memory from D (A:B)	(D) – (M : M + 1) ⇒ D

INH

+,-: Condition Code Register

- **C** set after an unsigned add if the answer is wrong
- **V** set after an signed add if the answer is wrong

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

Condition code bits are set after R=X+M, where X is initial register value, R is the final register value.

N: result is negative $N=R_7$

Z: result is zero $Z = \overline{R_7} \bullet \overline{R_6} \bullet \overline{R_5} \bullet \overline{R_4} \bullet \overline{R_3} \bullet \overline{R_2} \bullet \overline{R_1} \bullet \overline{R_0}$

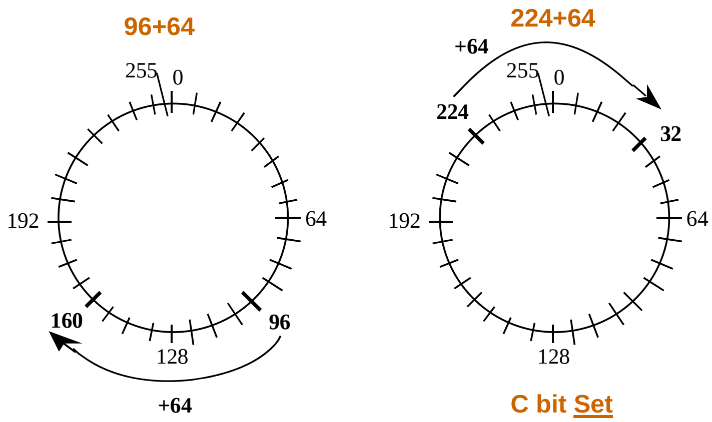
V: signed overflow $V = X_7 \bullet M_7 \bullet \overline{R_7} + \overline{X_7} \bullet \overline{M_7} \bullet R_7$

C: unsigned overflow $C = X_7 \bullet M_7 + M_7 \bullet \overline{R_7} + \overline{R_7} \bullet X_7$

C bit could also be set on a signed add but we usually do not look at it if our intent was to do a unsigned add

Similarly, the **V** bit could also be set on an unsigned add but we usually do not look at it if our intent was to do a signed add

Unsigned number wheel



C bit Cleared

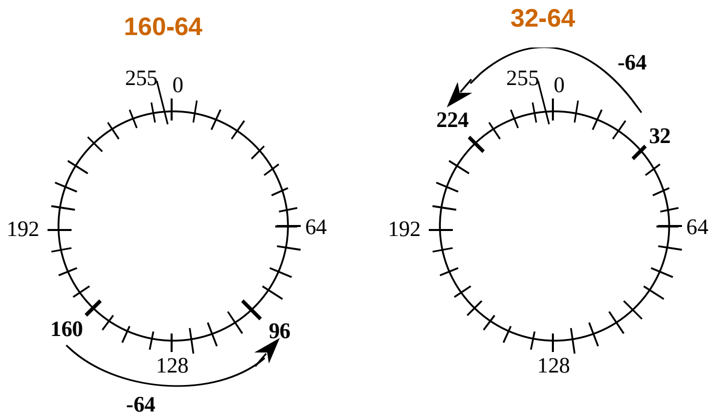
The carry bit, C, is set after an unsigned addition or subtraction when the result is incorrect.

Trick

- Whenever you directly add 2 unsigned 8-bit numbers and the answer is more than 255, C bit is SET. The final answer is obtained by subtracting 256 from the direct addition.

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

Unsigned Number Wheel



C bit <u>Cleared</u>

C bit <u>Set</u>

<u>The carry bit, C, is normally set when we cross over from 255 to 0 while adding, or cross over from 0 to 255 while subtracting.</u>

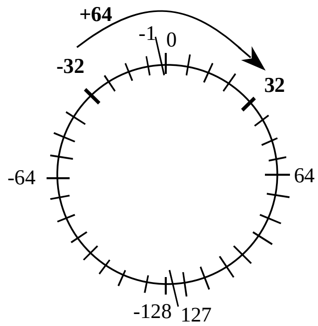
Trick

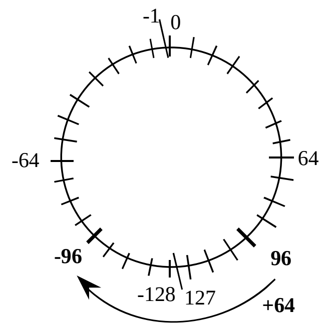
- Whenever you directly subtract 2 unsigned 8-bit numbers and the answer is negative, C bit is SET. The final answer is obtained by adding 256 to the direct subtraction.

For example: 5 - 255 = -250, C = 1 and the actual answer is -250+256 = 6

Signed Number Wheel

-32+64 **96+64**

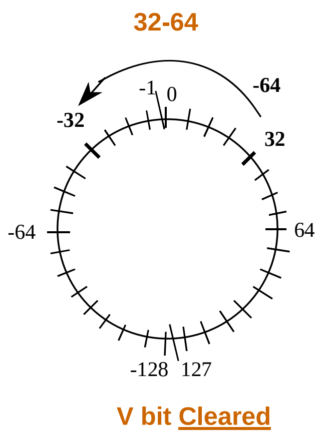


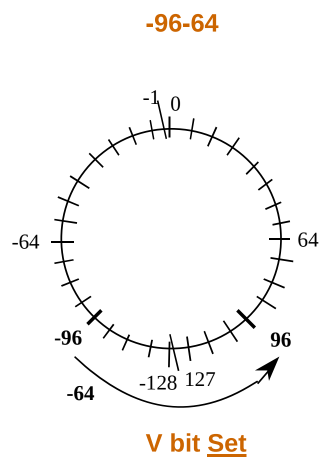


V bit Cleared **V bit Set**

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

Signed Number Wheel





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.

Addition Summary

Let the result R be the result of the addition A+B.

- **N bit** is set
  - if unsigned result is above 127 or
  - if signed result is negative.
  - N = R7
- **Z bit** is set if result is zero
- **V bit** is set after a signed addition if result is incorrect
  - V = $A7 \,\&\, B7 \,\&\, \overline{R7} + \overline{A7} \,\&\, \overline{B7} \,\&\, R7$
- **C bit** is set after an unsigned addition if result is incorrect
  - C = $A7 \,\&\, B7 + A7 \,\&\, \overline{R7} + B7 \,\&\, \overline{R7}$

Ramesh Yerraballi

3-18

On signed addition: V is set if we add two positive numbers and we get a negative result OR if we add two negative numbers and we get a positive result

On unsigned addition: C bit is set if we add two numbers both larger than or equal to 128 OR if the result is less than 128 when one of the operands is greater than 128.

E.g., of the second case (129 + 127 = 256) => 10000001 + 011111111 = 00000000 - In general it applies for sums of A and B which exceed 255 and A or B is >128

Subtraction Summary

Let the result R be the result of the addition A-B.

- **N bit** is set
  - if unsigned result is above 127 or
  - if signed result is negative.
  - N = R7
- **Z bit** is set if result is zero
- **V bit** is set after a signed subtraction if result is incorrect
  - V = $A7 \,\&\, \overline{B7} \,\&\, \overline{R7} + \overline{A7} \,\&\, B7 \,\&\, R7$
- **C bit** is set after an unsigned subtraction if result is incorrect
  - C = $\overline{A7} \,\&\, B7 + B7 \,\&\, R7 + \overline{A7} \,\&\, R7$

Ramesh Yerraballi

3-19

On signed subtraction: V is set if we subtract positive number from a negative number and we get a positive result OR if we subtract a negative number from a positive number and we get a negative result

On unsigned subtraction [R= A-B]: we have an overflow if A is less than B. That is A is less than 128 and B is greater than 128 (`A7' & B7`) OR B is greater than 128 and the result is still greater than 128 (`B7 & R7`) OR A is less than 128 and the result is greater than 128 **(**`A7' & R7`**)** .

E.g., Case 1: (A(32) - B(129) = R(159) --> A7=0; B7=1

Case 2: (A(150) - B(159) = R(247) --> B7=1; R7=1

Case 3: (A(32) - B(42)) = R(246) --> A7=0; R7=1

Problem

When you perform (32 – 129) in an 8-bit system what is the status of the NZVC bits?

Answer = 159

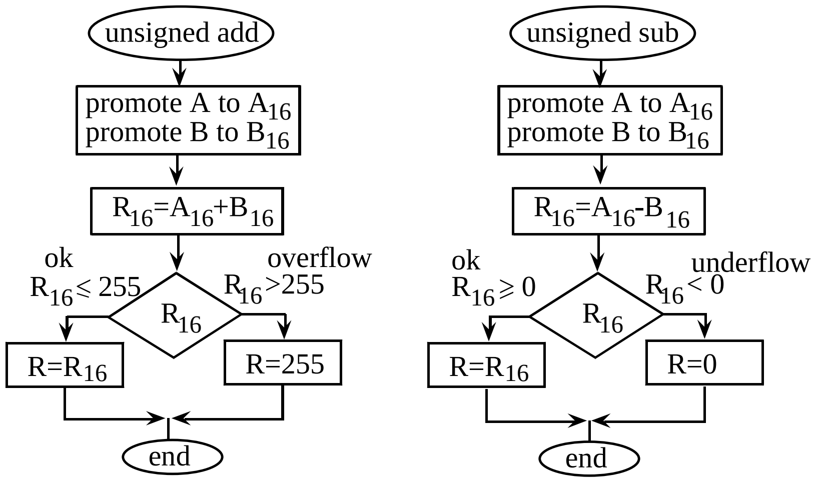
NZVC = $\boxed{1011}$

Unsigned Promotion

- Promotion involves increasing the precision of the input numbers, and performing the operation at that higher precision

decimal	8-bit	16-bit
224	1110,0000	**0000,0000**,1110,0000
+ 64	+0100,0000	+**0000,0000**,0100,0000
288	0010,0000	0000,0001,0010,0000

Unsigned Ceiling and Floor



Ramesh Yerraballi

Signed Promotion

- To promote a signed number, we duplicate the sign bit

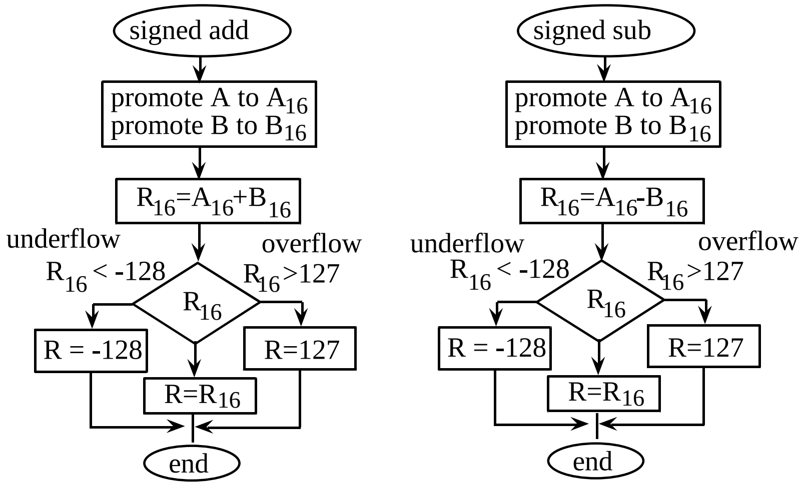
decimal	8-bit	16-bit
-96	1010,0000	**1111,1111**,1010,0000
- 64	-0100,0000	-**0000,0000**,0100,0000
-160	0110,0000	1111,1111,0110,0000

Ramesh Yerraballi

3-23

Otherwise known as sign extension (SEXT opcode)

Signed Ceiling and Floor



Conditionals

> < ≥ ≤ conditional branch instructions must follow a subtract compare or test instruction, such as

- **suba subb sbca sbcb subd cba cmpa cmpb cpd cpx cpy tsta tstb tst**

C code	assembly code

`if(G2 == G1){`
`  isEqual();`
`}`
	`     ldaa G2`
	`     cmpa G1`
	`     bne  next`
	`     jsr  isEqual`
	`next`

`if(G2 != G1){`
`  isNotEqual();`
`}`
	`     ldaa G2`
	`     cmpa G1`
	`     beq  next`
	`     jsr  isNotEqual`
	`next`

`if(H2 == H1){`
`  isEqual();`
`}`
	`     ldd  H2`
	`     cpd  H1`
	`     bne  next`
	`     jsr  isEqual`
	`next`

`if(H2 != H1){`
`  isNotEqual();`
`}`
	`     ldd  H2`
	`     cpd  H1`
	`     beq  next`
	`     jsr  isNotEqual`
	`next`

Equality/Inequality check

Ramesh Yerraballi 3-25

suba, subb: Do not use a carry; sbca and sbcb use the carry bit

```
sbca Mem    ; A <— A-[Mem]-C
```

tsta, tstb and tst check respectively whether RegA, RegB or the specicified memory location is zero.

Signed Conditional

bge target

;Branch if signed greater than or equal to (≥),

;if (N^V)=0, or (~N.V+N.~V)=0

bgt target

;Branch if signed greater than (>),

;if (Z+N^V)=0, or (Z+~N.V+N.~V)=0

ble target

;Branch if signed less than or equal to (≤),

;if (Z+N^V)=1, or (Z+~N.V+N.~V)=1

blt target

;Branch if signed less than (<),

;if (N^V)=1, or (~N.V+N.~V)=1

Ramesh Yerraballi

3-26

blt: is preceded by a comparison (cmpa, cmpb, cpd, cps, cpy, cpx, cba)

or subtraction (sbca, sbcb, suba, subb, subd) operation;

Checks whether the contents of Reg{A/B/D} is less than the Memory contents specified:

```
        cmpa $3600
        blt        next              ; if RegA < [$3600]
        jsr        gtrEqual
next
```

Say A has X1 and $3600 has X2. The cmpa operation above performs the subtraction (X1-X2
counter-clock-wise journey of X2 steps from X1. So if X1 is less than X2 then the difference

•The difference does not cause an underflow (Examples: X1=30, X2= 60; X1=-64, X2=10; X1

•The difference causes an underflow (Examples: X1=-64, X2 = 65; X1=-2, X2=127) => N bit i

So, the check for blt is: (N.~V + ~N.V) == 1

… Signed Conditional

C code	assembly code

`if(G2 > G1){`
`  isGreater();`
`}`
	`    ldaa G2`
	`    cmpa G1`
	`    ble  next`
	`    jsr  isGreater`
	`next`

`if(G2 >= G1){`
`  isGreaterEq();`
`}`
	`    ldaa G2`
	`    cmpa G1`
	`    blt  next`
	`    jsr  isGreaterEq`
	`next`

`if(G2 < G1){`
`  isLess();`
`}`
	`    ldaa G2`
	`    cmpa G1`
	`    bge  next`
	`    jsr  isLess`
	`next`

`if(G2 <= G1){`
`  isLessEq();`
`}`
	`    ldaa G2`
	`    cmpa G1`
	`    bgt  next`
	`    jsr  isLessEq`
	`next`

≤, <, ≥, > checks

Unsigned Conditional

```
bhs target
```
;Branch if unsigned greater than or equal to (≥),
;if C=0, same as bcc

```
bhi target
```
;Branch if unsigned greater than (>),
;if C+Z=0

```
bls target
```
;Branch if unsigned less than or equal to (≤),
;if C+Z=1

```
blo target
```
;Branch if unsigned less than (<),
;if C=1, same as bcs

… Unsigned Conditional

C code	assembly code

`if(G2 > G1){`
`  isGreater();`
`}`
	`    ldaa G2`
	`    cmpa G1`
	`    bls  next`
	`    jsr  isGreater`
	`next`

`if(G2 >= G1){`
`  isGreaterEq();`
`}`
	`    ldaa G2`
	`    cmpa G1`
	`    blo  next`
	`    jsr  isGreaterEq`
	`next`

`if(G2 < G1){`
`  isLess();`
`}`
	`    ldaa G2`
	`    cmpa G1`
	`    bhs  next`
	`    jsr  isLess`
	`next`

`if(G2 <= G1){`
`  isLessEq();`
`}`
	`    ldaa G2`
	`    cmpa G1`
	`    bhi  next`
	`    jsr  isLessEq`
	`next`

≤, <, ≥, > checks