;@============================================================================

;@

;@ Student Name 1: Minh Phat Tran

;@ Student 1 #: 301297286

;@ Student 1 userid (email): phatt (phatt@sfu.ca)

;@

;@ Student Name 2: Linchong Weng

;@ Student 2 #: 301308043

;@ Student 2 userid (email): lwa89 (lwa89@sfu.ca)

;@

;@ Below, edit to list any people who helped you with the code in this file,

;@ or put ï¿½noneï¿½ if nobody helped (the two of) you.

;@

;@ Helpers: TAs

;@

;@ Also, reference resources beyond the course textbooks and the course pages on Canvas

;@ that you used in making your submission.

;@

;@ Resources: lab3 document

;@

;@% Instructions:

;@ \* Put your name(s), student number(s), userid(s) in the above section.

;@ \* Edit the "Helpers" line and "Resources" line.

;@ \* Your group name should be "<userid1>\_<userid2>" (eg. stu1\_stu2)

;@ \* Form groups as described at: https://courses.cs.sfu.ca/docs/students

;@ \* Submit your file to courses.cs.sfu.ca

;@

;@ Name : asuAdd.S

;@ Description : asuAdd subroutine for Assignment 1.

;@============================================================================

;@ For Eclipse...

;@ Set "Displayed tab width" to 8 in Window > Preferences > General > Editors > Text Editors

;@ For Keil MDK...

;@ Highly recommended in Edit > Configuration

;@ Set "Auto Indent" to: "None"

;@ Tabs set for 8 characters for ASM files

;@ include macro definitions including for FUNCTION and ENDFUNC

.include "../src/macros.i"

;@ arbitrary-sized-unsigned-addition subroutine

;@ implements:

;@ typedef unsigned int bigNumN[];

;@ int asuAdd(bigNumN bigN0PC, const bigNumN bigN1PC, unsigned int maxN0Size);

;@ as explained in the assignment handout

;@ returns -1 for error in inputs, 1 if overflow/carry-out and 0 if no overflow/carry-out

;@asuAdd:

FUNCTION asuAdd

push {r1-r11, lr} @ store values r1 to r11 and link register

mov r10, #0 @ store value 0 to r10, r10 will be a counter

ldr r3, [r0, #0] @ load the first element of r0 to r3

ldr r4, [r1, #0] @ load the first element of r1 to r4

CMP r3, #0 @ compare the size of first array with the maximum size

BLEQ Counter @ if yes, counter will increase by 1

CMP r4, #0 @ compare the size of second array with the maximum size

BLEQ Counter @ if yes, counter will increase by 1

CMP r10, #2 @ check if the counter has the value of 2 or not

BEQ bothSizeZero @ if yes, branch to bothSizeZero subroutine

CMP r3, r4 @ compare the size of first array and the size of second array

MOVHI r5, r3 @ and the larger size will be stored to r5

MOVLS r5, r4

@ compare the size with the maximum size

CMP r5, r2

BHI Addition1

CMP r5, r2

BEQ Addition2

CMP r2, r5

BHI Addition3

@ count how many arrays which has the size of 0

Counter:

add r10, r10, #1

mov pc, lr

@ this condition is banched when both size of arrays are zero

bothSizeZero:

mov r0, #0

pop {r1-r11, pc}

@ this condition is branched when the largest size is larger than the maximum size

Addition1:

adds r0, r0, #0

str r5, [r0, #0]

MRS r11, CPSR

mov r6, #4

loop1:

MSR CPSR, r11

ldr r7, [r0, r6]

ldr r8, [r1, r6]

adcs r9, r7, r8

MRS r11, CPSR

str r9, [r0, r6]

add r6, r6, #4

subs r5, r5, #1

BNE loop1

mov r0, #-1

pop {r1-r11, pc}

@ this condition is branched when the largest size is equal the maximum size

Addition2:

adds r0, r0, #0

str r5, [r0, #0]

MRS r11, CPSR

mov r6, #4

loop2:

MSR CPSR, r11

ldr r7, [r0, r6]

ldr r8, [r1, r6]

adcs r9, r7, r8

MRS r11, CPSR

str r9, [r0, r6]

add r6, r6, #4

subs r5, r5, #1

BNE loop2

BCS checkCarry2

mov r0, #0

pop {r1-r11, pc}

@ this conditon is branched when the result of Addition2 is overflowed

checkCarry2:

mov r0, #1

pop {r1-r11, pc}

@ this conditon is branched when the largest size is smaller than the maximum size

Addition3:

adds r0, r0, #0

str r5, [r0, #0]

push {r5}

MRS r11, CPSR

mov r6, #4

loop3:

MSR CPSR, r11

ldr r7, [r0, r6]

ldr r8, [r1, r6]

adcs r9, r7, r8

MRS r11, CPSR

str r9, [r0, r6]

add r6, r6, #4

subs r5, r5, #1

BNE loop3

pop {r5}

MSR CPSR, r11

BCS checkCarry3

mov r0, #0

pop {r1-r11, pc}

@ this conditon is branched when the result of Additon3 is overflowed

checkCarry3:

add r5, r5, #1

str r5, [r0, #0]

mov r9, #1

str r9, [r0, r6]

mov r0, #0

pop {r1-r11, pc}

@ end of functiopn

ENDFUNC asuAdd

;@ .size asuAdd, .-asuAdd