Embedded Systems with ARM Cortex-M3 Microcontrollers in Assembly Language and C

Chapter 6
Flow Control in Assembly

Dr. Yifeng Zhu
Electrical and Computer Engineering
University of Maine

Spring 2015

Overview

- **`If-then-else`**
- **`While loop`**
- **`For loop`**

Comparison Instructions

Instruction	Operands	Brief description	Flags
CMP	Rn, Op2	Compare	N,Z,C,V
CMN	Rn, Op2	Compare Negative	N,Z,C,V
TEQ	Rn, Op2	Test Equivalence	N,Z,C
TST	Rn, Op2	Test	N,Z,C

- The only effect of the comparisons is to **update the condition flags**.
  - No need to set S bit.
  - No need to specify Rd.
- Operations are:

CMP	operand1 - operand2, but result not written
CMN	operand1 + operand2, but result not written
TST	operand1 & operand2, but result not written
TEQ	operand1 ^ operand2, but result not written

- Examples:

```
CMP     r0, #3; compare r0 with 3
CMN     r0, #3; compare r0 with -3
```

CMP and CMN

CMP{cond} Rn, Operand2

CMN{cond} Rn, Operand2

- The CMP instruction subtracts the value of Operand2 from the value in Rn.
  - This is the same as a SUBS instruction, except that the result is discarded.
- The CMN instruction adds the value of Operand2 to the value in Rn.
  - This is the same as an ADDS instruction, except that the result is discarded.
- These instructions update the N, Z, C and V flags according to the result.

Example of CMP

$$f(x) = |x|$$

```
         AREA absolute, CODE, READONLY
         EXPORT __main
         ENTRY

__main PROC
          MOV     r1, #-10
         CMP     r1, #0
         RSBLT   r1, r1, #0

done     B done      ; deadloop

         ENDP
         END
```

Note: RSB = Reverse SuBtract

TST and TEQ

```
TST{cond} Rn, Operand2    ; Bitwise AND
TEQ{cond} Rn, Operand2    ; Bitwise Exclusive OR
```

- The TST instruction performs a bitwise AND operation on the value in Rn and the value of Operand2.
  - This is the same as a ANDS instruction, except that the result is discarded.
- The TEQ instruction performs a bitwise Exclusive OR operation on the value in Rn and the value of Operand2.
  - This is the same as a EORS instruction, except that the result is discarded.
- Update the N and Z flags according to the result
- Can update the C flag during the calculation of Operand2
- Do not affect the V flag.

Branch Instructions

Instruction	Operands	Brief description	Flags
B	label	Branch	-
BL	label	Branch with Link	-
BLX	Rm	Branch indirect with Link	-
BX	Rm	Branch indirect	-

- *B label*: causes a branch to label.
- *BL label*: instruction copies the address of the next instruction into r14 (lr, the link register), and causes a branch to label.
- *BX Rm*: branch to the address held in Rm
- *BLX Rm*: copies the address of the next instruction into r14 (lr, the link register) and branch to the address held in Rm

Branch With Link

- The "Branch with link (BL)" instruction implements a subroutine call by writing PC-4 into the LR of the current bank.
  - i.e. the address of the next instruction following the branch with link (allowing for the pipeline).
- To return from subroutine, simply need to restore the PC from the LR:
  - `MOV pc, lr`
  - Again, pipeline has to refill before execution continues.
- The "Branch" instruction does not affect LR.

Condition Codes

Suffix	Description	Flags tested
EQ	EQual	
NE	Not Equal	
CS/HS	Unsigned Higher or Same	
CC/LO	Unsigned LOwer	
MI	MInus (Negative)	
PL	PLus (Positive or Zero)	
VS	oVerflow Set	
VC	oVerflow Clear	
HI	Unsigned HIgher	
LS	Unsigned Lower or Same	
GE	Signed Greater or Equal	
LT	Signed Less Than	
GT	Signed Greater Than	
LE	Signed Less than or Equal	
AL	ALways	

Note AL is the default and does not need to be specified

Condition Codes

- The possible condition codes are listed below:

Suffix	Description	Flags tested
EQ	EQual	Z=1
NE	Not Equal	Z=0
CS/HS	Unsigned Higher or Same	C=1
CC/LO	Unsigned LOwer	C=0
MI	MInus (Negative)	N=1
PL	PLus (Positive or Zero)	N=0
VS	oVerflow Set	V=1
VC	oVerflow Clear	V=0
HI	Unsigned HIgher	C=1 & Z=0
LS	Unsigned Lower or Same	C=0 or Z=1
GE	Signed Greater or Equal	N=V
LT	Signed Less Than	N!=V
GT	Signed Greater Than	Z=0 & N=V
LE	Signed Less than or Equal	Z=1 or N!=V
AL	ALways	

Note AL is the default and does not need to be specified

Signed Greater or Equal (N == V)

CMP r0, r1

We in fact perform subtraction r0 – r1, without saving the result.

	N = 0	N = 1
V = 0	• No overflow, implying the result is correct. • The result is non-negative, • Thus r0 – r1 ≥ 0, i.e., r0 ≥ r1	• No overflow, implying the result is correct. • The result is negative. • Thus r0 – r1 < 0, i.e., r0 < r1
V = 1	• Overflow occurs, implying the result is incorrect. • The result is mistakenly reported as non-negative and in fact it should be negative. • Thus r0 – r1 < 0 in reality, i.e., r0 < r1	• Overflow occurs, implying the result is incorrect. • The result is mistakenly reported as negative and in fact it should be non-negative. • Thus r0 – r1 ≥ 0 in reality., i.e. r0 ≥ r1

Conclusions:

- If N == V, then it is signed greater or equal (GE).
- Otherwise, it is signed less than (LT)

Signed vs. Unsigned

Conditional codes applied to branch instructions

Compare	Signed	Unsigned
==	EQ	EQ
≠	NE	NE
>	GT	HI
≥	GE	HS
<	LT	LO
≤	LE	LS

Compare	Signed	Unsigned
==	BEQ	BEQ
!=	BNE	BNE
>	BGT	BHI
>=	BGE	BHS
<	BLT	BLO
<=	BLE	BLS

Branch Instructions

	Instruction	Description	Flags tested
Unconditional Branch	B label	Branch to label	
Conditional Branch	BEQ label	Branch if EQual	Z = 1
	BNE label	Branch if Not Equal	Z = 0
	BCS/BHS label	Branch if unsigned Higher or Same	C = 1
	BCC/BLO label	Branch if unsigned LOwer	C = 0
	BMI label	Branch if MInus (Negative)	N = 1
	BPL label	Branch if PLus (Positive or Zero)	N = 0
	BVS label	Branch if oVerflow Set	V = 1
	BVC label	Branch if oVerflow Clear	V = 0
	BHI label	Branch if unsigned HIgher	C = 1 & Z = 0
	BLS label	Branch if unsigned Lower or Same	C = 0 or Z = 1
	BGE label	Branch if signed Greater or Equal	N = V
	BLT label	Branch if signed Less Than	N != V
	BGT label	Branch if signed Greater Than	Z = 0 & N = V
	BLE label	Branch if signed Less than or Equal	Z = 1 or N = !V

Number Interpretation

Which is greater?

`0xFFFFFFFF` or `0x00000001`

- If they represent signed numbers, the latter is greater.
- If they represent unsigned numbers, the former is greater.

Which is Greater: 0xFFFFFFFF or 0x00000001?

It's software's reasonability to tell computer how to interpret data:

- If written in C, declare the signed vs unsigned variable
- If written in Assembly, use signed vs unsigned branch instructions

```
signed int x, y ;            MOV r6, #0xFFFFFFFF
x = 1;                       MOV r5, #0x00000001
y = 2;                       CMP  r5, r6
if (x > y)                   BLE  Then_Clause
...
```

BLE: Branch if less than or equal, signed ≤

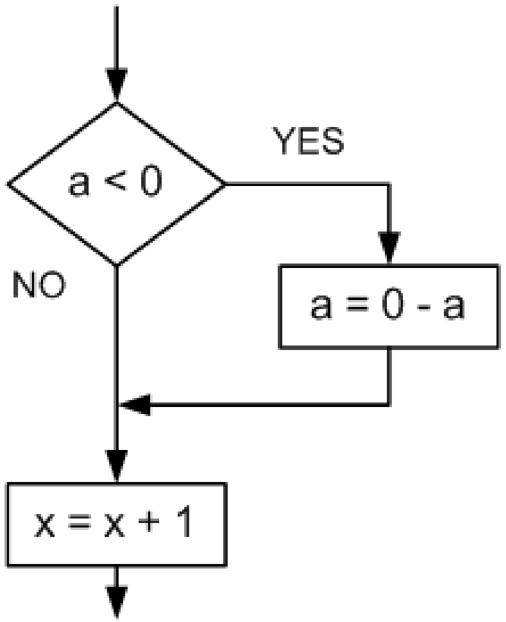
```
unsigned int x, y ;          MOV r6, #0xFFFFFFFF
x = 1;                       MOV r5, #0x00000001
y = 2;                       CMP  r5, r6
if (x > y)                   BLS  Then_Clause
...
```

BLS: Branch if lower or same, unsigned ≤

If-then Statement

C Program

```
if (a < 0 ) {
  a = 0 - a;
}
x = x + 1;
```

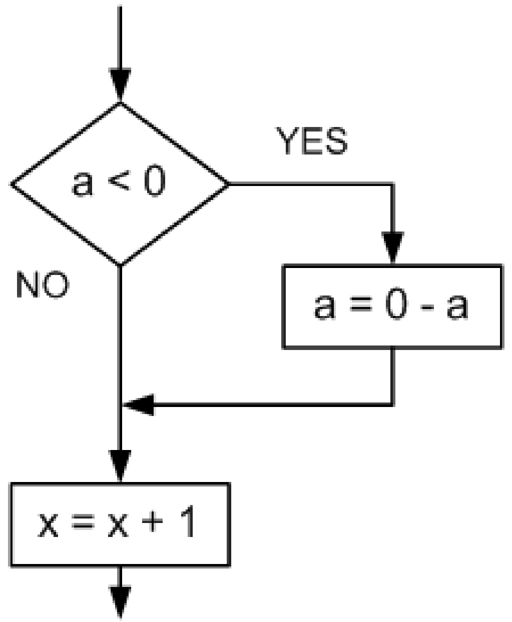


Implementation 1:

```
          ; r1 = a, r2 = x
          CMP r1, #0          ; Compare a with 0
          BGE endif           ; Go to endif if a ≥ 0
then      RSB r1, r1, #0      ; a = - a
endif     ADD r2, r2, #1      ; x = x + 1
```

If-then Statement

```
C Program
if (a < 0 ) {
  a = 0 - a;
}
x = x + 1;
```



Implementation 2:

```
; r1 = a, r2 = x
CMP    r1, #0         ; Compare a with 0
RSBLT  r1, r1, #0     ; a = 0 - a if a < 0
ADD    r2, r2, #1     ; x = x + 1
```

Compound Boolean Expression

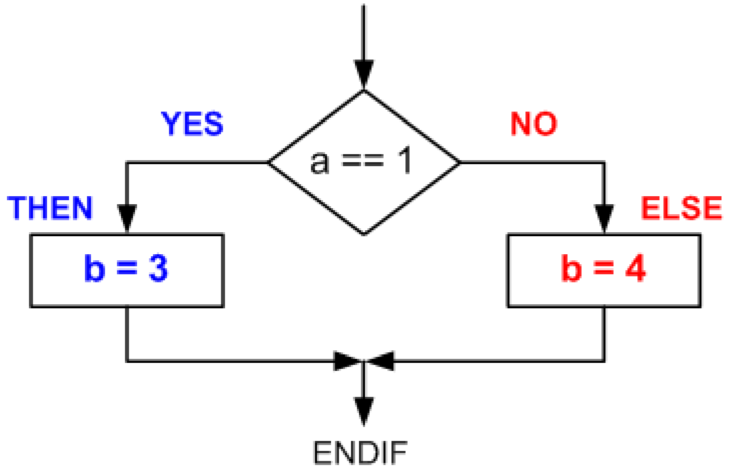
```
 x > 20 && x < 25
x == 20 || x == 25
!(x == 20 || x == 25)
```

C Program	Assembly Program

`// x is a signed integer`
`if(x <= 20 || x >= 25){`
`   a = 1`
`}`
	`       ; r0 = x`
	`       CMP  r0, #20   ; compare x and 20`
	`       BLE  then      ; go to then if x ≤ 20`
	`       CMP  r0, #25   ; compare x and 25`
	`       BLT  endif     ; go to endif if x < 25`
	`then   MOV  r1, #1    ; a = 1`
	`endif`

If-then-else

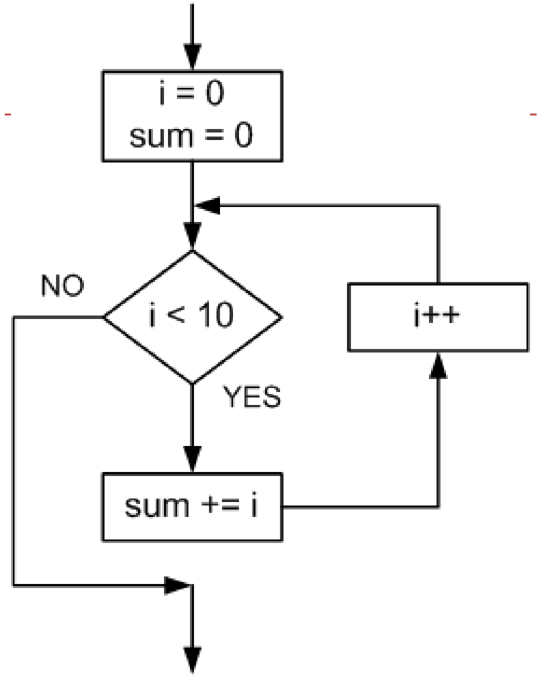
```
C Program
if (a == 1)
    b = 3;
else
    b = 4;
```



```
         ; r1 = a, r2 = b
         CMP r1, #1    ; compare a and 1
         BNE else      ; go to else if a ≠ 1
then     MOV r2, #3    ; b = 3
         B   endif     ; go to endif
else     MOV r2, #4    ; b = 4
endif
```

For Loop

```
C Program
int i;
int sum = 0;
for(i = 0; i < 10; i++){
   sum += i;
}
```



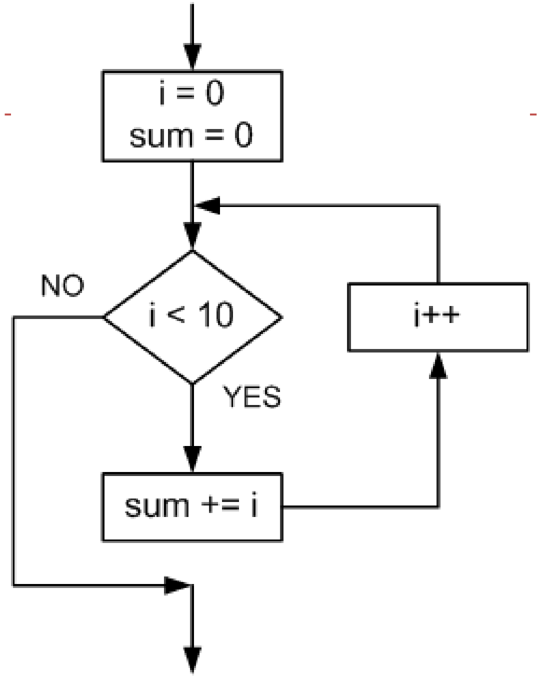
Implementation 1:

```
         MOV r0, #0   ; i
         MOV r1, #0   ; sum

         B   check
loop     ADD r1, r1, r0
         ADD r0, r0, #1
check    CMP r0, #10
         BLT loop
endloop
```

For Loop

```
C Program
int i;
int sum = 0;
for(i = 0; i < 10; i++){
   sum += i;
}
```

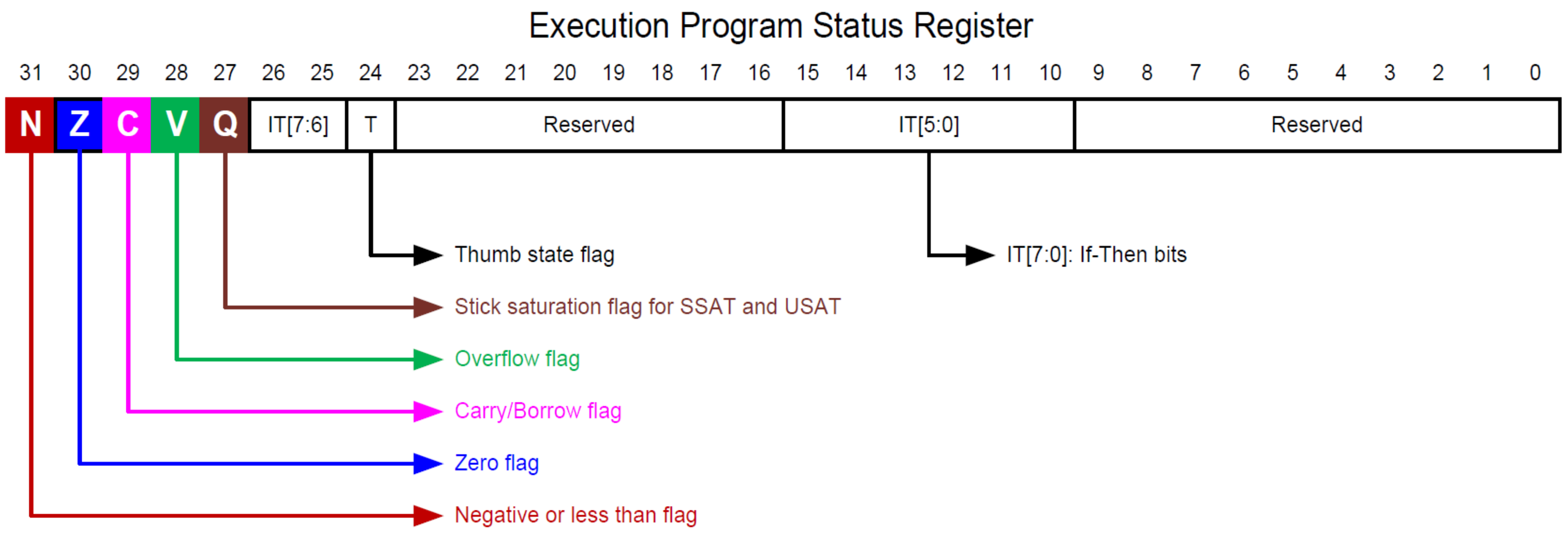


Implementation 2:

```
loop        MOV r0, #0   ; i
            MOV r1, #0   ; sum

loop        CMP r0, #10
            BGE endloop
            ADD r1, r1, r0
            ADD r0, r0, #1
            B   loop
endloop
```

Current Program Status Registers (CPSR)



Condition Codes

- The possible condition codes are listed below:

Suffix	Description	Flags tested
EQ	Equal	Z=1
NE	Not equal	Z=0
CS/HS	Unsigned higher or same	C=1
CC/LO	Unsigned lower	C=0
MI	Negative	N=1
PL	Positive or Zero	N=0
VS	Overflow	V=1
VC	No overflow	V=0
HI	Unsigned higher	C=1 & Z=0
LS	Unsigned lower or same	C=0 or Z=1
GE	Signed Greater or equal	N=V
LT	Signed Less than	N!=V
GT	Signed Greater than	Z=0 & N=V
LE	Signed Less than or equal	Z=1 or N!=V
AL	Always	

Note AL is the default and does not need to be specified

Conditional Execution

Add instruction	Condition	Flag tested
ADDEQ r3, r2, r1	Add if EQual	Add if Z = 1
ADDNE r3, r2, r1	Add if Not Equal	Add if Z = 0
ADDHS r3, r2, r1	Add if Unsigned Higher or Same	Add if C = 1
ADDLO r3, r2, r1	Add if Unsigned LOwer	Add if C = 0
ADDMI r3, r2, r1	Add if Minus (Negative)	Add if N = 1
ADDPL r3, r2, r1	Add if PLus (Positive or Zero)	Add if N = 0
ADDVS r3, r2, r1	Add if oVerflow Set	Add if V = 1
ADDVC r3, r2, r1	Add if oVerflow Clear	Add if V = 0
ADDHI r3, r2, r1	Add if Unsigned HIgher	Add if C = 1 & Z = 0
ADDLS r3, r2, r1	Add if Unsigned Lower or Same	Add if C = 0 or Z = 1
ADDGE r3, r2, r1	Add if Signed Greater or Equal	Add if N = V
ADDLT r3, r2, r1	Add if Signed Less Than	Add if N != V
ADDGT r3, r2, r1	Add if Signed Greater Than	Add if Z = 0 & N = V
ADDLE r3, r2, r1	Add if Signed Less than or Equal	Add if Z = 1 or N = !V

Example of Conditional Execution

```
if (a <= 0)
  y = -1;
else
  y = 1;
```

$$a \longrightarrow r0$$
$$y \longrightarrow r1$$

```
CMP    r0, #0
MOVLE  r1, #-1
MOVGT  r1, #1
```

LE: Signed Less than or Equal
GT: Signed Greater Than

Example of Conditional Execution

```
a ⟶ r0
y ⟶ r1
```

```
if (a==1 || a==7 || a==11)
   y = 1;
else
   y = -1;
```

$\Rightarrow$

```
CMP   r0, #1
CMPNE r0, #7
CMPNE r0, #11
MOVEQ r1, #1
MOVNE r1, #-1
```

NE: Not Equal
EQ: Equal

Compound Boolean Expression

C Program	Assembly Program

`// x is a signed integer`
`if(x <= 20 || x >= 25){`
`    a = 1;`
`}`
	`       ; r0 = x, r1 = a`
	`       CMP   r0, #20  ; compare x and 20`
	`       MOVLE r1, #1   ; a=1 if less or equal`
	`       CMP   r0, #25  ; CMP if greater than`
	`       MOVGE r1, #1   ; a=1 if greater or equal`
	`endif`

Example 1: Greatest Common Divider (GCD)

Euclid's Algorithm

```
While (a != b ) {
    if (a > b) a = a – b;
    else b = b – a;
}
```

```
gcd   CMP r0, r1
      SUBGT r0, r0, r1
      SUBLT r1, r1, r0
      BNE gcd
```

```
         ; suppose r0 = a and r1 = b
gcd      CMP r0, r1           ; a > b?
         BEQ end              ; if a = b, done

         BLT less             ; a < b
         SUB  r0, r0, r1      ; a = a – b
         B gcd

less     SUB r1, r1, r0       ; b = b – a
         B gcd
```

Example 2

```
int foo(int x, int y) {
  if ( x + y < 0 )
    return 0;
  else
    return 1;
}
```

```
foo     ADDS    r0, r0, r1
        BPL     PosOrZ
done    MOV     r0, #0
        MOV     pc, lr
PosOrZ  MOV     r0, r1
        B       done
```

```
foo     ADDS    r0, r0, r1      ;  r1 = x + y,  setting CCs
        MOVPL   r0, #1          ;  return 1 if n bit = 0
        MOVMI   r0, #0          ;  return 0 if n bit = 1
        MOV     pc, lr          ;  exit foo function
```

Combination

Instruction	Operands	Brief description	Flags
CBZ	Rn, label	Compare and Branch if Zero	-
CBNZ	Rn, label	Compare and Branch if Non Zero	-

- Except that it does not change the condition code flags, CBZ Rn, label is equivalent to:

```
CMP     Rn, #0
BEQ     label
```

- Except that it does not change the condition code flags, CBNZ Rn, label is equivalent to:

```
CMP     Rn, #0
BNE     label
```

Break and Continue

Example code for break	Example code for continue

`for(int i = 0; i < 5; i++){`
`  if (i == 2) break;`
`  printf(“%d, ”, i)`
`}`
	`for(int i = 0; i < 5; i++){`
	`  if (i == 2) continue;`
	`  printf(“%d, ”, i)`
	`}`

Output: ??	Output: ??

Break and Continue

Example code for break	Example code for continue

`for(int i = 0; i < 5; i++){`
`  if (i == 2) break;`
`  printf(“%d, ”, i)`
`}`
	`for(int i = 0; i < 5; i++){`
	`  if (i == 2) continue;`
	`  printf(“%d, ”, i)`
	`}`

`Output: 0, 1,`	`Output: 0, 1, 3, 4`

Break and Continue

C Program

```c
// Find string length
char str[] = "hello";
int len = 0;

for( ; ; ) {
    if (*str == '\0')
        break;
    str++;
    len++;
}
```

Assembly Program

```
          ; r0 = string memory address
          ; r1 = string length
          MOV  r1, #0          ; len = 0

loop      LDRB r2, [r0]
          CBNZ r2, notZero
          B    endloop
notZero   ADD  r0, r0, #1      ; str++
          ADD  r1, r1, #1      ; len++
          B    loop
endloop
```

IT (If-Then) instruction

IT{x{y{z}}} {cond}

- where the x, y, and z specify the existence of the optional second, third, and fourth conditional instruction respectively.
- x, y, and z are either T (Then) or E (Else)

Examples:

```
ITTE    NE          ; IT can be omitted
ANDNE  r0,r0,r1    ; 16-bit AND, not ANDS
ADDNE  r2,r2,#1    ; 32-bit ADDS
MOVEQ  r2,r3       ; 16-bit MOV
```

```
ITT    EQ
MOVEQ  r0,r1
BEQ    dloop     ; branch at end of IT block is permitted
```

```
ITT    EQ
MOVEQ  r0,r1
ADDEQ  r0,r0,#1
```

```
ITT    AL             ; emit 2 non-flag setting 16-bit instructions
ADDAL  r0,r0,r1    ; 16-bit ADD, not ADDS
SUBAL  r2,r2,#1    ; 16-bit SUB, not SUB
ADD    r0,r0,r1    ; expands into 32-bit ADD, and is not in  IT block
```

IT (If-Then) instruction

IT{x{y{z}}} {cond}

- where the x, y, and z specify the existence of the optional second, third, and fourth conditional instruction respectively.
- x, y, and z are either T (Then) or E (Else)
- You do not need to write IT instructions in your code.
- The assembler generates them for you automatically according to the conditions specified.

Branch Instructions

Instruction	Operands	Brief description	Flags
B	label	Branch	-
BL	label	Branch with Link	-
BLX	Rm	Branch indirect with Link	-
BX	Rm	Branch indirect	-

- *B label*: causes a branch to label.
- *BL label*: instruction copies the address of the next instruction into r14 (lr, the link register), and causes a branch to label.
- *BX Rm*: branch to the address held in Rm
- *BLX Rm*: copies the address of the next instruction into r14 (lr, the link register) and branch to the address held in Rm