Chapter 6
Control Flow in Assembly

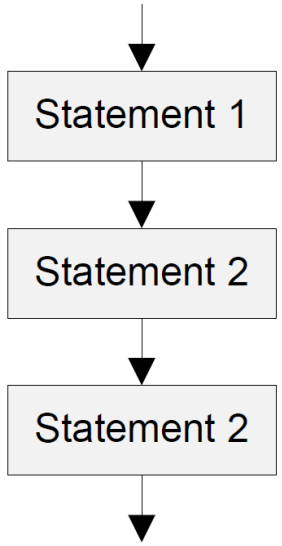
Z. Gu

Fall 2025

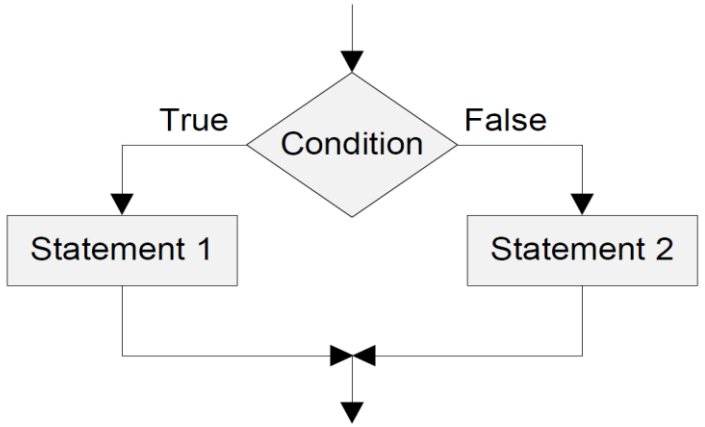
Acknowledgement: Lecture slides based on Embedded Systems with ARM Cortex-M Microcontrollers in Assembly Language and C, University of Maine https://web.eece.maine.edu/~zhu/book/

Three Control Structures

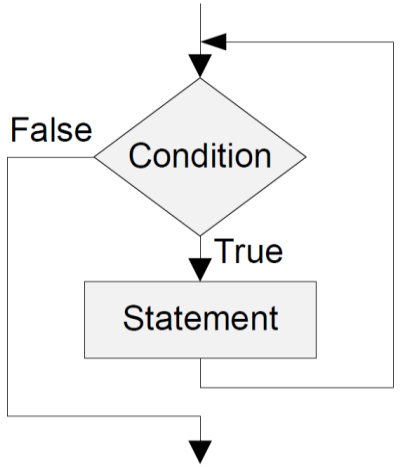
- Sequence Structure
  - Computer executes statements (instructions), one after another, in the order listed in the program
- **`Selection Structure`**
  - **`If-then-else`**
- **`Loop Structure`**
  - **`while loop`**
  - **`for loop`**



Sequence Structure



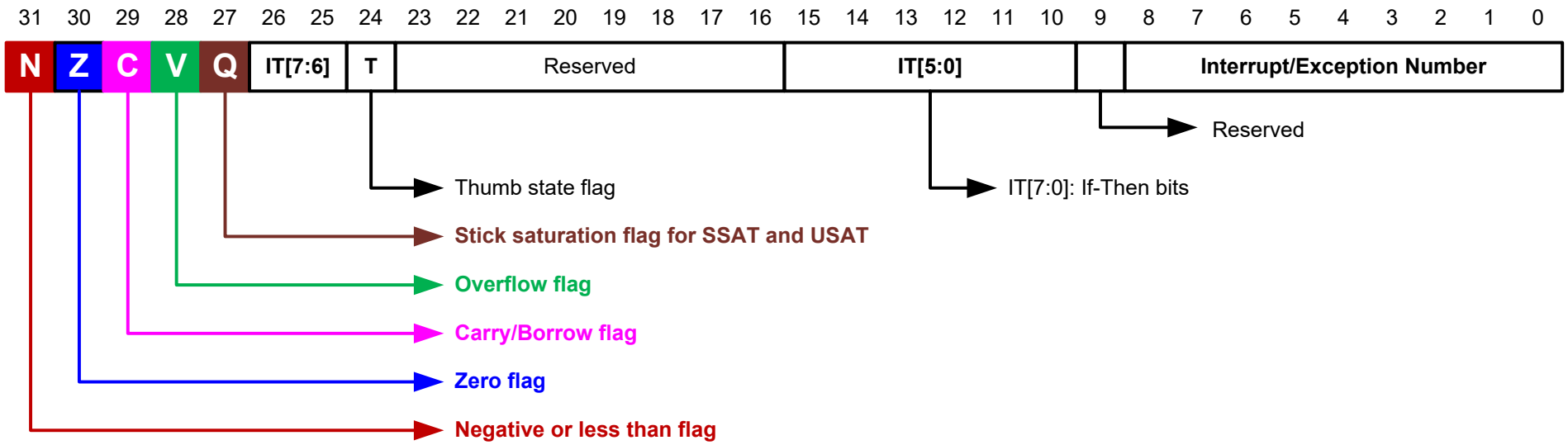
Selection Structure



Loop Structure

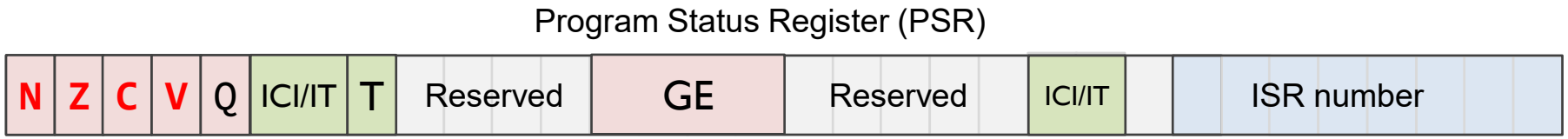
Review

Combined Program Status Registers (xPSR)



Review

Condition Flags



- **Negative** bit
  - `N = 1` if most significant bit of result is 1
- **Zero** bit
  - `Z = 1` if all bits of result are `0`
- **Carry** bit
  - For unsigned addition, `C = 1` if carry takes place
  - For unsigned subtraction, `C = 0` (carry = not borrow) if borrow takes place
  - For shift/rotation, C = last bit shifted out
- **oVerflow** bit
  - `V = 1` if adding 2 same-signed numbers produces a result with the opposite sign
    - Positive + Positive = Negative, or
    - Negative + negative = Positive
  - Non-arithmetic operations does not touch V bit, such as `MOV`, `AND`, `LSL`, `MUL`

Negative ---------- signed result is negative

Zero ---------- result is 0

Carry ---------- add op ➜ overflow
sub op doesn't borrow
last bit shifted out when shifting

oVerflow --- add/sub op ➜ signed overflow

Review

Carry and Overflow Flags w/ Arithmetic Instructions

Carry flag C = 1 (Borrow flag = 0) upon an **unsigned** addition if the answer is wrong (true result > 2^n-1)

Carry flag C = 0 (Borrow flag = 1) upon an **unsigned** subtraction if the answer is wrong (true result < 0)

Overflow flag V =1 upon a **signed** addition or subtraction if the answer is wrong (true result > $2^{n-1}-1$ or true result < -2^{n-1})

Overflow may occur when adding 2 operands with the same sign, or subtracting 2 operands with different signs; Overflow cannot occur when adding 2 operands with different signs or when subtracting 2 operands with the same sign.

	Unsigned Addition	Unsigned Subtraction	Signed Addition or Subtraction
Carry flag	true result > 2^n-1 ➔ Carry flag=1 Borrow flag=0 (Result incorrect)	true result < 0 ➔ Carry flag=0 Borrow flag=1 (Result incorrect)	N/A
Overflow flag	N/A	N/A	true result > $2^{n-1}-1$ or true result < -2^{n-1} ➔ Overflow flag=1 (Result incorrect)

Updating Condition Flags

- Method 1: append "**S**": updates destination register, and sets flags
  - `ADD r0,r1,r2` → `ADDS r0,r1,r2`
  - `SUB r0,r1,r2` → `SUBS r0,r1,r2`
  - Performs operation, writes the result into Rd, and also updates NZCV flags
- Method 2: compare instructions: sets flags only
  - CMP/CMN/TEQ/TST: performs operation to update NZCV flags, but the computation result is not saved and discarded

Updating Condition Flags

Instruction	Operands	Brief description	Flags
CMP	R1 - R2	Compare	N,Z,C,V
CMN	R1 + R2	Compare Negative	N,Z,C,V
TST	R1 & R2	Test	N,Z,C
TEQ	R1 ⊕ R2	Test Equivalence	N,Z,C

- **Update flags**
  - No need to add S. No need to specify destination register.
- Operations are:
  - `CMP` `R1 – R2`: Same as SUBS, except result discarded (not written to destination register)
  - `CMN` `R1 + R2`: Same as ADDS, except result discarded
  - `TST` `R1 & R2`: Same as ANDS, except result discarded
  - `TEQ` `R1 ⊕ R2`: Same as EORS, except result discarded
- Examples:
  - `CMP r0, r1`
  - `TST r2, #5`

Example of CMP

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

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

__main  PROC
        CMP     r1, #0          ; r1 = x
        RSBLT   r0, r1, #0

done    B done      ; deadloop, end of program

        ENDP
        END
```

RSBLT r0, r1, #0:: conditional execution of the RSB instruction with condition code LT. If r1 < 0, then set r0 = 0 – r1 = -r1

Updating Condition Flags: TST and TEQ

```
TST R1, R2   ; Bitwise AND
TEQ R1, R2   ; Bitwise Exclusive OR
```

- Update N and Z according to the result
- Can update C during the calculation of R2 (w/ shifting such as LSL, LSR…)
- Do not affect V
- TST performs bitwise AND on R1 and R2.
  - Same as ANDS, except result discarded.
  - Use R2 as a mask; Z=0 implies "some masked bit(s) are set, so result is non-zero" Z=1 implies "none of the masked bit(s) are set, so result is zero." For a single-bit mask, Z=0 means "that bit in R1 is 1," and Z=1 means "that bit is 0."
- TEQ performs bitwise Exclusive OR on R1 and R2.
  - Same as EORS, except result discarded.
  - If R1 and R2 are equal, then R1 $\oplus$ R2 is 0, and Z is set to 1; otherwise Z is set to 0 (cleared).

x	y	x AND y
0	0	0
0	1	0
1	0	0
1	1	1

x	y	x XOR y
0	0	0
0	1	1
1	0	1
1	1	0

Example of TEQ

- Translate C code into assembly:

C Code	Assembly

`if (char=='!'||char=='?')`
`found++;`
	`TEQ r0, #'!'`
	`TEQNE r0, #'?'`
	`ADDEQ r1, r1, #1`

- TEQ r0, #'!' performs a test-equal by computing r0 '!' and setting condition flags; Z=1 when r0 equals '!'.
- TEQNE r0, #'?' executes only if the previous Z=0 (i.e., char was not '!'); it tests r0 against '?' and sets Z accordingly. This achieves the logical OR without branches by conditionally running the second test only when needed.
  - Logical OR operator (||) employs short-circuit evaluation, meaning it evaluates expressions from left to right and stops as soon as the result of the entire expression is determined. For (cond1||cond2): If cond1 evaluates to true (non-zero), the overall result of the || operation is already known to be true, so cond2 is not evaluated. If cond1 evaluates to false (zero), the evaluation proceeds to the next operand cond2.
- ADDEQ r1, r1, #1 executes only if Z=1 after the tests, meaning char matched either '!' or '?'.
  - If r0 == '!', then TEQ sets Z = 1. TEQNE is skipped, and ADDEQ is executed
  - The 2nd TEQNE executes only if the first comparison failed (Z=0). If r0 == '?', then TEQNE is executed and sets Z = 1, and ADDEQ is executed
  - If r0 != '!' && r0 != '?', then TEQ sets Z = 0, TEQNE is executed and sets Z = 0, and ADDEQ is not executed

Unconditional Branch Instructions

Instruction	Operands	Brief description
B, BAL	label	Branch
BL	label	Branch with Link
BLX	Rm	Branch indirect with Link
BX	Rm	Branch indirect

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

Unconditional Branch Instructions: A Simple Example

```
        MOVS r1, #1
        B    target  ; Branch to target
        MOVS r2, #2  ; Not executed
        MOVS r3, #3  ; Not executed
        MOVS r4, #4  ; Not executed
target  MOVS r5, #5
```

- A label marks the location of an instruction
- Labels help human to read the code
- In machine program, labels are converted to numeric offsets by assembler
- Here MOVS can be replaced by MOV since the flags are not used

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 and 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 and 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 vs. Unsigned Comparison

Op	Cond (Signed)	Flags	Explanation	Cond (Unsigned)	Flags	Explanation
R1 > R2	GT (Greater Than)	Z=0 & N=V	Non-zero result and signs agree	HI (Higher)	C=1 & Z=0	No borrow and not equal
R1 ≥ R2	GE (Greater or Equal)	N=V	See next page	HS (Higher or Same)	C=1	No borrow (R1 ≥ R2)
R1 < R2	LT (Less Than)	N≠V	See next page	LO (Lower)	C=0	Borrow occurred (R1 < R2)
R1 ≤ R2	LE (Less or Equal)	Z=1 or N≠V	Zero or overflow mismatch	LS (Lower or Same)	C=0 or Z=1	Borrow or equal
R1==R2	EQ (Equal)				Z=1	Zero
R1 ≠ R2	NE (Not Equal)				Z=0	Non-Zero

CMP R1, R2
perform subtraction R1 – R2, set flags without saving result

Signed Comparison Explanations

Condition (signed)	N	V	CMP R1, R2 returns	Meaning
GE (Greater or Equal)	0	0	1	Result non-negative (R1 – R2 ≥ 0), no overflow → R1 ≥ R2
GE (Greater or Equal)	1	1	1	Result negative (R1 – R2 < 0), but overflowed so sign is flipped → true result ≥ 0 → R1 ≥ R2
LT (Less Than)	1	0	0	Result negative (R1 – R2 < 0), no overflow → R1 < R2
LT (Less Than)	0	1	0	Result non-negative (R1 – R2 ≥ 0), but overflowed so sign is flipped → true result < 0 → R1 < R2

- If N = V, then GE (CMP R1, R2 returns 1)
- If N ≠ V, then LT (CMP R1, R2 returns 0)

Signed Comparison Examples (5-bit system)

	N = 0	N = 1
V = 0	• R1 = +7 (00111) • R2 = +3 (00011) • R1 − R2 = +4 (00100); • result non-negative and no signed overflow, so N=0,V=0 ⇒ GE holds	• R1 = +3 (00011) • R2 = +7 (00111) • R1 − R2 = −4 (11100) • result negative with no overflow, so N=1,V=0 ⇒ LT holds
V = 1	• R1 = −10 (10110) • R2 = +7 (00111) • R1 − R2 = −17, outside range [−16,+15]; result is 00111 (decimal 7), whose sign bit is 0 so N=0, but signed overflow occurs so V=1 ⇒ LT holds	• R1 = +10 (01010) • R2 = −7 (11001) • R1 − R2 = +17, outside range [−16,+15]; result is 10001 (decimal −15), whose sign bit is 1 so N=1, but signed overflow occurs so V=1 ⇒ GE holds

- If N = V, then GE (CMP R1, R2 returns 1)
- If N ≠ V, then LT (CMP R1, R2 returns 0)

Number Interpretation

Which is greater?

`0xFFFFFFFF` or `0x00000001`

- If they represent signed numbers, the latter is greater.
  (**1 > -1**).

- If they represent unsigned numbers, the former is greater
  (2^{32-1} **> 1**).

Which is Greater: 0xFFFFFFFF or 0x00000001?

It's **software's responsibility** 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

```
int32_t x, y;
x = -1;
y = 1;
if (x > y)
   ...
```

```
    MOV r5, #0xFFFFFFFF
    MOV r6, #0x00000001
    CMP  r5, r6
    BLE  Then_Clause
    ...
```

BLE: Branch if less than or equal, signed ≤

```
uint32_t x, y;
x = 4294967295;
y = 1;
if (x > y)
   ...
```

```
    MOV r5, #0xFFFFFFFF
    MOV r6, #0x00000001
    CMP  r5, r6
    BLS  Then_Clause
    ...
```

BLS: Branch if lower or same, unsigned ≤

Conditional Branch Instructions

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

Conditional codes applied to branch instructions

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

If-then Statement

C Program	Assembly Program 1	Assembly Program 2

`// a is signed integer`
`if (a < 0 ) {`
`  a = 0 – a;`
`}`
`x = x + 1;`
	`; r1 = a, r2 = x`
	`CMP r1, #0  ; Compare a with 0`
	`BGE endif ; Go to endif if a ≥ 0`
	`RSB r1, r1, #0  ; a = - a`
	`endif: ADD r2, r2, #1 ; x = x + 1`
		`; r1 = a, r2 = x`
		`CMP  r1, #0`
		`RSBLT r1, r1, #0  ; a = - a if a < 0`
		`ADD  r2, r2, #1  ; x = x + 1`

C Program	Assembly Program 1	Assembly Program 2

`// a is signed integer`
`if(a <= 20 || a >= 25){`
`  x = 1`
`}`
	`; r1 = a, r2 = x`
	`CMP  r1, #20  ; compare a and 20`
	`BLE  then  ; go to then if a ≤ 20`
	`CMP  r1, #25  ; compare a and 25`
	`BLT  endif  ; go to endif if a < 25`
	`then:  MOV  r2, #1  ; x = 1`
	`Endif`
	`; implements short circuit evaluation of || condition (if 1st condition is true, 2nd condition checking is skipped)`
		`; r1 = a, r2 = x`
		`CMP  r1, #20  ; compare a and 20`
		`MOVLE  r2, #1  ; a <= 20 → x = 1`
		`CMP  r1, #25  ; compare a and 25`
		`MOVGE  r2, #1  ; a >= 25 → x = 1`
		`; else (21 <= a <= 24) → no MOV executed. Does not implement short circuit evaluation. Both conditions will always be evaluated, and r2 is possibly assigned twice.`

If-then-else

C Program	Assembly Program 1

`// a is signed integer`
`if (a == 1)`
`  x = 3;`
`else`
`  x = 4;`
	`; r1 = a, r2 = b`
	`  CMP r1, #1  ; compare a and 1`
	`  BNE else  ; go to else if a ≠ 1`
	`then: MOV r2, #3  ; x = 3`
	`B  endif  ; go to endif`
	`else: MOV r2, #4  ; x = 4`
	`endif:`

For Loop

C Program

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

C Program (equivalent)

```
int i = 0;
int sum = 0;

while (i < 10) {
    sum += i;
    i++;
}
```

Implementation 1 (Classic compare-and-branch):

```
    MOV     r0, #0          % sum = 0
    MOV     r1, #0          % i = 0
loop:
    CMP     r1, #10         % i < 10 ?
    BGE     done            % exit if i >= 10
    ADD     r0, r0, r1      % sum += i
    ADD     r1, r1, #1      % i++
    B       loop
done:           % : is optional after a label
```

For Loop

C Program

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

C Program (equivalent)

```
int i = 0;
int sum = 0;

while (i < 10) {
    sum += i;
    i++;
}
```

Implementation 2a:

```
    MOV     r0, #0   % sum = 0
    MOV     r1, #0   % i = 0
    B   check
loop:
    ADD r0, r0, r1   % sum += i
    ADD r1, r1, #1   % i++
Check: CMP r1, #10   % check whether i < 10
    BLT loop         % loop if i less than 10.
```

For Loop

C Program

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

C Program (equivalent)

```
int i = 0;
int sum = 0;

do {
    sum += i;
    i++;
} while (i < 10);
```

Implementation 2b:

```
    MOV     r0, #0          % sum = 0
    MOV     r1, #0          % i = 0
    %B   check deleted
Loop:
    ADD r0, r0, r1  % sum += i
    ADD r1, r1, #1  % i++
    CMP r1, #10     % check whether i < 10
    BLT loop        % loop if i less than 10.
```

Explanations for 2a and 2b

- 2a and 2b implement two different loop structures:
  - Version with "B check" is a pre-test loop (while/for): it tests i < 10 before the first iteration, so the body may execute zero times if the condition is false initially
  - Version without "B check" is a post-test loop (do-while): it executes the body once before testing, then repeats while i < 10
- Because i starts at 0 and the condition is i < 10, and loop iterates from i=0 to i=9, even though the control-flow order differs.
  - If the while condition is initially true (i < 10), this program has same behavior as previous version. But if the while condition is initially false (i >= 10), this program executes for 1 iteration while the previous program executes for 0 iteration.

For Loop

C Program

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

C Program (equivalent)

```
int sum = 0;      // r0
int count = 10;   // r1
int i = 0;        // r2

while (count != 0) {
    sum += i;
    i += 1;
    count -= 1;
}
```

Implementation 3 (Count-down with SUBS/BNE):

```
    MOV     r0, #0          % sum = 0
    MOV     r1, #10         % loop count = 10
    MOV     r2, #0          % i = 0
loop:
    ADD     r0, r0, r2      % sum += i
    ADD     r2, r2, #1      % i++
    SUBS    r1, r1, #1      % --count, set flags
    BNE     loop            % repeat until loop
count==0
```

```
SUBS r1, r1, #1
Is equivalent to:
SUB r1, r1, #1
CMP r1, #0
```

For Loop

C Program

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

C Program (equivalent)

```
int i = 0;
int sum = 0;

while (i < 10) {
    sum += i;
    i++;
}
```

Implementation 4 (Use conditional execution):

```
    MOV     r0, #0          % sum = 0
    MOV     r1, #0          % i = 0
loop:
    CMP     r1, #10         % set flags from i-10
    ADDLT   r0, r0, r1      % if i<10: sum += i
    ADDLT   r1, r1, #1      % if i<10: i++
    BLT     loop            % if i<10: loop
```

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

Conditional Execution Examples

C Program	Assembly Program

`// a, x are signed integers`
`int32_t a, y`
`if (a <= 0)`
`  y = -1;`
`else`
`  y = 1;`
	`; r0 = a, r1 = y`
	`CMP   r0, #0`
	`MOVLE r1, #-1 ; executed if LE`
	`MOVGT r1, #1  ; executed if GT`

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

`if (a==1 || a==7 || a==11)`
`   y = 1;`
`else`
`   y = -1;`
	`CMP   r0, #1`
	`CMPNE r0, #7  ; executed if r0 != 1`
	`CMPNE r0, #11 ; executed if r0 != 1`
	`                ; and r0 != 7`
	`MOVEQ r1, #1`
	`MOVNE r1, #-1`

LE : Signed Less than or Equal
GT : Signed Greater Than
NE: Not Equal
EQ: Equal

Thought Experiments

- This version does not work, since we cannot use any condition code ?? in MOV??

```
CMP   r0, #20  ; compare a and 20
CMPGT r0, #25  ; CMP if greater
than
MOV?? r1, #1   ; Does not work.
cannot use any condition code here
```

- This version is incorrect, since only the last "CMP r0, #11" sets the Z flag, overwriting flags set by previous two CMP instructions.

```
CMP   r0, #1
CMP   r0, #7
CMP   r0, #11

MOVEQ r1, #1
MOVNE r1, #-1
```

Explanations for Compound Boolean Expression

- Compound condition (a==1 || a==7 || a==11):
  - CMP r0, #1
  - CMPNE r0, #7 ; executed if r0 != 1
  - CMPNE r0, #11 ; executed if r0 != 7
  - MOVEQ r1, #1
  - MOVNE r1, #-1
- CMP r0, #1 compares a with 1 and sets Z=1 if equal, else Z=0.
- CMPNE r0, #7 runs only if the previous compare was not equal, and if it runs, it refreshes the flags by comparing a with 7.
- CMPNE r0, #11 runs only if a was not 1 and not 7, and if it runs, it compares a with 11 to set Z accordingly.
- MOVEQ r1, #1 executes only when Z=1 from any of the comparisons, so y becomes 1 if a matched 1, 7, or 11.
- MOVNE r1, #-1 executes only when Z=0 after all relevant compares, so y becomes -1 when none of the values matched.

Conditional Execution Examples Con't

C Program	Assembly Program w/ Branching	Assembly Program w/ Conditional Execution

`int32_t x, y;`
`……`
`if (x + y < 0)`
`  x = 0;`
`else`
`  x = 1;`
	`% r0 = x, r1 = y`
	`  ADDS r0, r0, r1`
	`  BPL  PosOrZ`
	`  MOV  r0, #0`
	`  B  done`
	`PosOrZ MOV  r0, #1`
	`done`
		`ADDS  r0, r0, r1`
		`MOVMI  r0, #0  ;return 0 if N = 1`
		`MOVPL  r0, #1  ;return 1 if N = 0`

`uint32_t x, y;`
`……`
`while (x != y) {`
`  if (x > y)`
`  x = x - y;`
`  else`
`  y = y - x;`
`}`
	`gcd  CMP r0, r1`
	`  BEQ end  ; if x = y, done`
	`  BLO less  ; x < y`
	`  SUB r0, r0, r1  ; x = x - y`
	`  B  gcd`
	`less  SUB r1, r1, r0  ; y = y - x`
	`  B  gcd`
	`End`
		`gcd  CMP r0, r1`
		`  SUBHI r0, r0, r1`
		`  SUBLO r1, r1, r0`
		`  BNE gcd`

Combination

Instruction	Operands	Brief description
CBZ	R1, label	Compare and Branch if Zero
CBNZ	R1, label	Compare and Branch if Non Zero

- Except that it does not change the status flags, `CBZ R1,label` is equivalent to:

```
CMP R1, #0
BEQ label
```

- Except that it does not change the status flags, `CBNZ R1,label` is equivalent to:

```
CMP  R1, #0
BNE  label
```

Break vs. 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 Example

C Program	Assembly Program 1	Assembly Program 2

`// Find string length`
`char str[] = "hello";`
`int len = 0;`
 	
`for( ; ; ) {`
`  if (*str == '\0')`
`  break;`
`  str++;`
`  len++;`
`}`
	`;r0 = string memory address`
	`;r1 = string length`
	`  ADR  r0, str`
	`  MOV  r1, #0`
	`Loop:  LDRB r2, [r0]`
	`  CBNZ r2, notZero`
	`  B  endloop`
	`notZero: ADD  r0, r0, #1  ; str++`
	`; to let r0 point to the next char. This works since each char is 1 byte (need to increment by 4 if it was an integer array)`
	`  ADD  r1, r1, #1  ; len++`
	`  B  loop`
	`endloop:`
		`;r0 = string memory address`
		`;r1 = string length`
		`  ADR  r0, str`
		`  MOV  r1, #0`
		`Loop:  LDRB r2, [r0]`
		`  CBZ r2, endloop`
		`  ADD  r0, r0, #1`
		`  ADD  r1, r1, #1`
		`  B  loop`
		`endloop:`

Break Example

C Program	Assembly Program

`// Count characters that are not 'l' until the null terminator`
`char str[] = "hello";`
`int count = 0;`
 	
`for (; ; ) {`
`  if (*str == '\0')`
`  break;`
`  if (*str == 'l')`
`  continue;`
 	
`  count++;`
`  str++;`
`}`
	`; r0 = string address (str)`
	`; r1 = count = 0`
	`  ADR  r0, str`
	`  MOV  r1, #0`
	`Loop:  LDRB r2, [r0]`
	`  CBZ  r2, endloop  ; if '\0' => break`
	
	`  CMP  r2, #'l'  ; if char == 'l'`
	
	`  ADD  r1, r1, #1  ; count++`
	
	`  ADD  r0, r0, #1  ; str++`
	`  B  loop`
	`endloop:`

Break and Continue Example

C Program	Assembly Program 3

`// Count characters that are not 'l' until the null terminator`
`char str[] = "hello";`
`int count = 0;`
 	
`for (; ; ) {`
`  if (*str == '\0')`
`  break;`
 	
`  count++;`
`  str++;`
`}`
	`; r0 = string address (str)`
	`; r1 = count = 0`
	`  ADR  r0, str`
	`  MOV  r1, #0`
	`Loop:  LDRB r2, [r0]`
	`  CBZ  r2, endloop  ; if '\0' => break`
	
	`  CMP  r2, #'l'  ; if char == 'l'`
	`  BEQ  contLoop  ; continue and skip count++`
	
	`  ADD  r1, r1, #1  ; count++`
	
	`contLoop: ADD  r0, r0, #1  ; str++`
	`  B  loop`
	`endloop:`

IT (If-Then) instruction

You do not need to write IT instructions in your code. The assembler generates them for you automatically according to the conditions specified.

- On smaller ARM cores (Cortex-M0), not all data instructions support condition suffixes directly; instead you must use an IT instruction (Thumb-2) or branches.
- "IT" (If-Then) instruction in the ARM Thumb-2 instruction set (16 bits) allows conditional execution of up to four instructions based on a condition flag (like EQ, NE, etc.).
- **IT{x{y{z}}} {cond},** where 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). T = execute the following instruction if condition is True; E = execute the following instruction if condition is False
  - IT — 1 following instruction (If)
  - ITT — 2 following instructions (If-Then)
  - ITE — 2 following instructions (If-Else)
  - ITTE, ITEEE, etc. — up to 4 instructions total

```
ITTE NE      ; If-Then-Then-Else
ANDNE r0,r0,r1  ; executed if Not Equal
ADDNE r2,r2,#1  ; executed if Not Equal
MOVEQ r2,r3     ; executed if Equal
```

```
ITT    EQ    ; Executes both instructions only if Equal condition is true
   MOVEQ  r0,r1
   BEQ    dloop     ; branch at end of IT block is permitted
```

```
ITT    EQ ; Executes both instructions only if Equal condition is true
   MOVEQ  r0,r1
   ADDEQ  r0,r0,#1
```

```
ITT    AL          ;AL (Always) condition executes two 16-bit instructions unconditionally; the last ADD is outside the IT block.
   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
```

Summary: Condition Codes

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 Cleared	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

Summary: Branch Instructions

	Instruction	Description	Flags tested
Unconditional Branch	`B label`	Branch to label	
Conditional Branch	`BEQ label`	Branch if **EQ**ual	`Z = 1`
	`BNE label`	Branch if **N**ot **E**qual	`Z = 0`
	`BCS/BHS label`	Branch if unsigned **H**igher or **S**ame	`C = 1`
	`BCC/BLO label`	Branch if unsigned **LO**wer	`C = 0`
	`BMI label`	Branch if **MI**nus (Negative)	`N = 1`
	`BPL label`	Branch if **PL**us (Positive or Zero)	`N = 0`
	`BVS label`	Branch if o**V**erflow **S**et	`V = 1`
	`BVC label`	Branch if o**V**erflow **C**lear	`V = 0`
	`BHI label`	Branch if unsigned **HI**gher	`C = 1 & Z = 0`
	`BLS label`	Branch if unsigned **L**ower or **S**ame	`C = 0 or Z = 1`
	`BGE label`	Branch if signed **G**reater or **E**qual	`N = V`
	`BLT label`	Branch if signed **L**ess **T**han	`N != V`
	`BGT label`	Branch if signed **G**reater **T**han	`Z = 0 & N = V`
	`BLE label`	Branch if signed **L**ess than or **E**qual	`Z = 1 or N = !V`

Summary: Conditionally Executed

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

Summary: Condition Codes

- Condition Codes:
  - EQ/NE: Z=1 / Z=0 (Equal/Not Equal)
  - LT/GE: N≠V / N=V (Signed Less Than/Greater Equal)
  - GT/LE: Z=0 & N=V / Z=1 or N≠V (Signed Greater/Less Equal)
  - LO/HS: C=0 / C=1 (Unsigned Lower/Higher Same)
  - HI/LS: C=1 & Z=0 / C=0 or Z=1 (Unsigned Higher/Lower Same)
- Flag Setting Instructions:
  - CMP: R1 - R2 (result discarded)
  - TST: R1 & R2 (result discarded)
  - TEQ: R1 $\oplus$ R2 (result discarded)
  - CMN: R1 + R2 (result discarded)

References

- Lecture 27. Branch instructions
  - https://www.youtube.com/watch?v=_QKD7f1cmRI&list=PLRJhV4hUhlymmp5CCeIFPyxbknsdcXCc8&index=27
- Lecture 28. Conditional Execution
  - https://www.youtube.com/watch?v=9hlxG8L5-G4&list=PLRJhV4hUhlymmp5CCeIFPyxbknsdcXCc8&index=28