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

Chapter 6
Flow Control in Assembly
Exercises ANS

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/

Pseudocode to Assembly

- Write assembly program for pseudocode, one version without Conditional Execution instructions, one version with Conditional Execution.

Pseudocode	Assembly Program
`if (r0 != r1) r2 = r2 + r2`	

Pseudocode	Assembly Program w/Conditional Execution
`if (r0 != r1) r2 = r2 + r2`	

Pseudocode to Assembly ANS

- Write assembly program for pseudocode, one version without Conditional Execution instructions, one version with Conditional Execution.
  - if (r0 != r1) r2 = r2 + r2

Pseudocode	Assembly Program

`if (r0 != r1) r2 = r2 + r2`	`CMP r0, r1`
	`BEQ skip`
	`ADD r2, r2, r2`
	`skip:`

Pseudocode	Assembly Program w/Conditional Execution

`if (r0 != r1) r2 = r2 + r2`	`CMP r0, r1`
	`ADDNE r2, r2, r2`

Pseudocode to Assembly

- Write assembly program for pseudocode.

Pseudocode	Assembly Program 1
`r1 = (r0 >> 4) & 15`	

Pseudocode to Assembly ANS

- Write assembly program for pseudocode.

Pseudocode	Assembly Program 1

`r1 = (r0 >> 4) & 15`	`MOV r1, r0, LSR #4`
	`AND r1, r1, 0x0F`
	`%AND r1, r1, #15 is also OK`

Pseudocode	Assembly Program 2

`r1 = (r0 >> 4) & 15`	`AND r1, r0, #0xF0`
	`MOV r1, r1, LSR #4`

- Assembly Program 1: It computes r1 = (r0 >> 4) & 15 shifting it down by 4 bits, then masking the low nibble (4 bits) with 0x0F (or 0xF).
- Assembly Program 2: It computes r1 = (r0 >> 4) & 15 by first masking the high nibble with 0xF0 and then shifting it down by 4 bits.
  - AND r1, r0, #0xF0 keeps only bits [7:4] of r0, clearing all others.
  - MOV r1, r1, LSR #4 performs a logical right shift by 4 on the masked value, moving those bits into positions [3:0].

Assembly Program Understanding

- What is r0 equal to after running this program:
  - MOV r0, #10
  - MOV r1, #7
  - MOV r2, #5
  - CMP r1, r2
  - ADDGT r0, r0, #100

Assembly Program Understanding ANS

- What is r0 equal to after running this program:
  - MOV r0, #10 ; Load immediate value 10 into r0
  - MOV r1, #7 ; Load immediate value 7 into r1
  - MOV r2, #5 ; Load immediate value 5 into r2
  - CMP r1, r2 ; Compare r1 and r2 (compute r1-r2, set flags)
  - ADDGT r0, r0,100 ;Add 100 to r0 only if r1 > r2 (Greater Than)
- This code sets r0 to 10, compares 7 and 5. Since 7 > 5, adds 100 to the base value of 10 to get 110
- What about:
  - MOV r0, #10
  - MOV r1, #7
  - MOV r2, #5
  - SUBS r1, r2
  - ADDGE r0, r0, #100
- Here r0 = 110, since SUBS sets flags and ADDGE is executed

C to Assembly

- save[] is an array of 32-bit integers. Assume that i and k correspond to registers r1 and r2, and the base of the array save is in r0. Write assembly code corresponding to this C code.

C code	Assembly Program

`while (save[i] == k)`
`i += 1;`
	`; r0 = &save[0]      (base address of array)`
	`; r1 = i             (index)`
	`; r2 = k             (value to compare)`
	`...`

C to Assembly ANS

- save[] is an array of 32-bit integers. Assume that i and k correspond to registers r1 and r2, and the base of the array save is in r0. Write assembly code corresponding to this C code. (This program is not perfect, since the array size of save[] is not considered, so the while loop may never terminate)

C code	Assembly Program
`i = 0;`	`; r0 = &save[0]    (base address of array)`
`while (save[i] == k)`	`; r1 = i           (index)`
`i += 1;`	`; r2 = k           (value to compare)`
	`loop:`
	`    LDR r3, [r0, r1, LSL #2] ; r3 = save[i], 32-bit loads with index scaled by 4, so r3 is loaded from mem address r0+4*r1`
	`    CMP r3, r2      ; compare save[i] with k`
	`    BNE done       ; exit loop if save[i] != k`
	`    ADD r1, r1, #1 ; i += 1`
	`    B loop         ; repeat`
	`done:`

C to Assembly ANS 2

- This version removes r1, and uses post-index addressing to increment the array index r0, adding 4 to r0 in every loop iteration

C code	Assembly Program
i = 0;	; r0 = &save[0] ; pointer to current
while (save[i] == k)	element
i += 1;	; r2 = k ; value to compare
	loop:
	LDR r3, [r0], #4 ; load current
	save[i], advance pointer
	CMP r3, r2
	BNE done
	B loop
	done:

C to Assembly

- Write the equivalent assembly program for this piece of code in C.

C code	Assembly Program

for (i=0; i<8; i++){
a[i] = b[7-i];
}
	; Assume r0 = base address of a, r1 = base address of b

C to Assembly ANS

- Write the equivalent assembly program for this piece of code in C.

C code	Assembly Program

for (i=0; i<8; i++){
a[i] = b[7-i];
}
	; Assume r0 = base address of a, r1 = base address of b
	MOV r2, #0 ; i = 0
	loop_start:
	    CMP r2, #8
	    BGE loop_end
	    RSB r3, r2, #7 ; r3 = 7 – i
	    ;Note that SUB r3, #7, r2 is incorrect, since operand 1 cannot be an immediate value
	    LDR r4, [r1, r3, LSL#2] ; Load b[7-i] from mem address r1 + 4*r3
	    STR r4, [r0, r2, LSL#2] ; Store to a[i] to mem address r0 + 4*r2
	    ADD r2, r2, #1 ; i++
	    B loop_start
	loop_end:

C to Assembly: What is wrong?

C Program	Assembly Program
int cnt = 1;	MOV r0, #1 ; cnt = 1
while (cnt <= 10)	loop:
{	CMP r0, #10 ; Compare r0 with 10 while r0 <=10
// loop body	BEQ end ; If cnt == 10, branch to end
cnt++;	ADD r0, r0, #1 ; cnt = cnt + 1
}	B loop ; Repeat the loop
	done:

C Program	Assembly Program
int cnt = 10;	MOV r0, #10 ; remaining iterations
while (cnt != 0)	loop:
{	; loop body
// loop body	SUB r0, r0, #1 ; cnt--; sets flags
cnt--;	BNE loop ; repeat until cnt == 0 (10 times)
}	done:

C to Assembly: What is wrong? ANS: Loop runs for 9 iterations, not 10

C Program	Assembly Program

`int cnt = 1;`
`while (cnt <= 10)`
`{`
`    // loop body`
`    cnt++;`
`}`
	`MOV r0, #1        ; cnt = 1`
	`loop:`
	`   CMP r0, #11       ; Compare x0 with 10 while x0 <=10`
	`   BEQ done          ; If cnt == 10, branch to end`
	`   ADD r0, r0, #1    ; cnt = cnt + 1`
	`   B loop            ; Repeat the loop`
	`done:`

C Program	Assembly Program

`int cnt = 1;`
`while (cnt <= 10)`
`{`
`    // loop body`
`    cnt++;`
`}`
	`MOV r0, #1        ; cnt = 1`
	`loop:`
	`   CMP r0, #10       ; Compare x0 with 10 while x0 <=10`
	`   BGT done          ; If cnt > 10, branch to end`
	`   ADD r0, r0, #1    ; cnt = cnt + 1`
	`   B loop            ; Repeat the loop`
	`done:`

C to Assembly: What is wrong? ANS 2

C Program	Assembly Program
`uint cnt = 1;`	`MOV r0, #1        ; cnt = 1`
`while (cnt <= 10)`	`loop:`
`{`	`   CMP r0, #11      ; Compare x0 with 10 while x0 <=10`
`    // loop body`	`   BEQ done         ; If cnt == 10, branch to end`
`    cnt++;`	`   ADD r0, r0, #1   ; cnt = cnt + 1`
`}`	`   B loop           ; Repeat the loop`
	`done:`

C Program	Assembly Programu
`uint cnt = 1;`	`MOV r0, #1        ; cnt = 1`
`while (cnt <= 10)`	`loop:`
`{`	`   CMP r0, #10      ; Compare x0 with 10 while x0 <=10`
`    // loop body`	`   BHI done         ; If cnt > 10, branch to end`
`    cnt++;`	`   ADD r0, r0, #1   ; cnt = cnt + 1`
`}`	`   B loop           ; Repeat the loop`
	`done:`

C to Assembly: What is wrong? ANS

C Program	Assembly Program
`int cnt = 10;`	`MOV r0, #10          ; remaining iterations`
`while (cnt != 0)`	`B check`
`{`	`loop:  SUBS r0, r0, #1   ; cnt--; sets flags`
`    // loop body`	`check: BNE  loop    ; repeat until cnt == 0 (10 times)`
`    cnt--;`	`done:`
`}`	

SUB should be SUBS to set flags
Add a B check before loop body
(program behavior the same if cnt = r0 = 10 initially with or without "B check", but different if cnt = r0 = 0 initially. c.f. pp 22-25 in Ch6 lecture on while loop and do-while loop.)

C to Assembly

C Program	Assembly Program
`int cnt = 1;`	`MOV r0, #1          ; cnt = 1`
`while (cnt <= 10)`	`loop:`
`{`	`    ; loop body`
`    // loop body`	`    ADD  ??? ; cnt++`
`    cnt++;`	`    CMP  ???`
`}`	`    BLE  ???  ; while (cnt <= 10) continue`
	`done:`

C Program	Assembly Program
`int cnt = 10;`	`MOV r0, #10          ; remaining iterations`
`while (cnt != 0)`	`loop:`
`{`	`    ; loop body`
`    // loop body`	`    SUBS ???  ; cnt--; sets Z flag if r0=0`
`    cnt--;`	`    BNE  ???     ; repeat until cnt == 0 (10 times)`
`}`	`done:`

C Program	Assembly Program
`int cnt = 10;`	`MOV r0, #10          ; remaining iterations`
`while (cnt > 0) {`	`loop:`
`    // loop body`	`    ; loop body`
`    cnt--;`	`    SUBS ???; cnt--; sets Z flag if r0=0`
`}`	`    BGT  ???     ; repeat until cnt == 0 (10 times)`
	`done:`

C to Assembly ANS (assuming cnt is signed int)

C Program	Assembly Program
`int cnt = 1;`	`MOV r0, #1           ; cnt = 1`
`while (cnt <= 10)`	`loop:`
`{`	`    ; loop body`
`    // loop body`	`    ADD r0, r0, #1      ; cnt++`
`    cnt++;`	`    CMP r0, #10`
`}`	`    BLE loop              ; while (cnt <= 10) continue`
	`done:`

C Program	Assembly Program
`int cnt = 10;`	`MOV r0, #10          ; remaining iterations`
`while (cnt != 0)`	`loop:`
`{`	`    ; loop body`
`    // loop body`	`    SUBS r0, r0, #1   ; cnt--; sets Z flag if r0=0`
`    cnt--;`	`    BNE  loop      ; repeat until cnt == 0 (10 times)`
`}`	`done:`

C Program	Assembly Program
`int cnt = 10;`	`MOV r0, #10          ; remaining iterations`
`while (cnt > 0) {`	`loop:`
`    // loop body`	`    ; loop body`
`    cnt--;`	`    SUBS r0, r0, #1   ; cnt--; sets Z flag if r0=0`
`}`	`    BGT  loop      ; repeat until cnt == 0 (10 times)`
	`done:`

C to Assembly ANS (assuming cnt is unsigned int)

C Program	Assembly Program
`uint cnt = 1;`	`MOV r0, #1            ; cnt = 1`
`while (cnt <= 10)`	`loop:`
`{`	`    ; loop body`
`    // loop body`	`    ADD r0, r0, #1      ; cnt++`
`    cnt++;`	`    CMP r0, #10`
`}`	`    BLS loop              ; while (cnt <= 10) continue`
	`done:`

C Program	Assembly Program
`uint cnt = 10;`	`MOV r0, #10           ; remaining iterations`
`while (cnt != 0)`	`loop:`
`{`	`    ; loop body`
`    // loop body`	`    SUBS r0, r0, #1   ; cnt--; sets Z flag if r0=0`
`    cnt--;`	`    BNE  loop     ; repeat until cnt == 0 (10 times)`
`}`	`done:`

C Program	Assembly Program
`uint cnt = 10;`	`MOV r0, #10           ; remaining iterations`
`while (cnt > 0) {`	`loop:`
`    // loop body`	`    ; loop body`
`    cnt--;`	`    SUBS r0, r0, #1   ; cnt--; sets Z flag if r0=0`
`}`	`    BHI  loop     ; repeat until cnt == 0 (10 times)`
	`done:`

C to Assembly: What is wrong?

C Program	Assembly Program
`int cnt = 10;`	`MOV r0, #10          ; remaining iterations`
`while (cnt > 0) {`	`loop:`
`    // loop body`	`    ; loop body`
`    cnt--;`	`    SUBS r0, r0, #1    ; cnt--; sets Z flag if r0=0`
`}`	`    BPL  loop      ; repeat until cnt == 0 (10 times)`
	`done:`

C to Assembly: What is wrong? ANS

C Program	Assembly Program

```
int cnt = 10;
while (cnt > 0) {
    // loop body
    cnt--;
}
```
	```
	MOV r0, #10          ; remaining iterations
	loop:
	    ; loop body
	    SUBS r0, r0, #1   ; cnt--; sets Z flag if r0=0
	    BPL  loop      ; repeat until cnt == 0 (10 times)
	done:
	```

BPL, Branch if PLus (Positive or Zero, N = 0), tests the Negative flag N=0 (i.e., r0 >=0). Starting from 10 and decrementing, N stays 0 for 10 down to 0, so BPL would still branch at r0 = 0. Then SUBS makes r0 = −1, which sets N = 1, so BPL does not branch and the loop exits. Hence BPL allows the loop to run at 0 and overshoot by one iteration.

What's wrong: BPL tests for ≥ 0, not > 0 → extra iteration.

How to fix: Replace BPL loop with BGT loop.

If the loop variable were unsigned, you'd instead use BHI ("branch if higher") for the same effect.

Quiz: is BPL same as BGE?

ANS: No. PL checks N == 0; GE checks N == V. In this program BGE and BPL have the same behavior, since you never have signed overflow. But in general not, in case of signed overflow V = 1, c.f. p. 16 in Ch6 lecture "Signed Comparison Examples"

C to Assembly

C Program	Assembly Program

int array[200];
int i;
for (i = 199; i >= 0; i
= i - 1) {
    array[i] = array[i]
* 8;
}
	% R0 = base address of array, R1 = i
	MOV R0, 0x60000000 ; base address where array
	resides
	MOV R1, #199 ; i = 199
	…

C to Assembly ANS

C Program	Assembly Program
`int array[200];`	`; R0 = array base address, R1 = i`
`int i;`	`MOV R0, 0x60000000`
	`MOV R1, #199`
`for (i = 199; i >= 0; i = i -`	
`1)`	`FOR`
`array[i] = array[i] * 8;`	`  LDR R2, [R0, R1, LSL #2] ;R2 = array(i)`
	`  LSL R2, R2, #3 ; R2 = R2<<3 = R2*8`
	`  STR R2, [R0, R1, LSL #2] ;array(i) = R2`
	`  SUBS R1, R1, #1 ; i=i-1 and set flags`
	`  BPL FOR ; if (i >= 0) repeat loop`

- For an array of 32-bit ints, each element is 4 bytes, so element i lives at base + i*4, which is implemented as base + (i << 2) via LSL #2 in the addressing mode
- BPL "Branch if PLus" branches when the N (negative) flag N == 0, meaning the prior result was positive or zero

C to Assembly

C Program	Assembly Program
`//Calculate x such that 2^x=128`	`; r0 = pow, r1 = x`
`int pow = 1;`	`  MOV r0, #1 ; pow = 1`
`int x = 0;`	`  MOV r1, #0 ; x = 0`
	`WHILE`
`while (pow != 128) {`	
`pow = pow * 2;`	
`x = x + 1;`	
`}`	`DONE`

C to Assembly ANS

C Program	Assembly Program
`//Calculate x such that 2^x =`	`; r0 = pow, r1 = x`
`128`	`  MOV r0, #1 ; pow = 1`
`int pow = 1;`	`  MOV r1, #0 ; x = 0`
`int x = 0;`	`WHILE`
	`  CMP r0, #128 ; r0-128`
`while (pow != 128) {`	`  BEQ DONE ; if (pow==128) exit loop`
`  pow = pow * 2;`	`  LSL r0, r0, #1 ; pow = pow * 2`
`  x = x + 1;`	`  ADD r1, r1, #1 ; x = x + 1`
`}`	`  B WHILE ; repeat loop`
	`DONE`

C to Assembly ANS

C Program	Assembly Program

`//Calculate x such that 2^x = 128`
`int pow = 1;`
`int x = 0;`
 	
`while (pow != 128) {`
`  pow = pow * 2;`
`  x = x + 1;`
`}`
	`; r0 = pow, r1 = x`
	`  MOV r0, #1 ; pow = 1`
	`  MOV r1, #0 ; x = 0`
	`WHILE`
	`  LSL r0, r0, #1 ; pow = pow * 2`
	`  ADD r1, r1, #1 ; x = x + 1`
	`  CMP r0, #128 ; r0-128`
	`  BNE WHILE ; repeat loop`
	`DONE`

- **Edge case:** If r0 starts as 128 (zero-iteration case), this version will execute the body once (doubling to 256 and incrementing r1) before checking — i.e., wrong for that initial condition, c.f. pp 22-25 in Ch6 lecture on while loop and do-while loop. To handle the edge case, add a branch to "CMP r0, #128" before the WHILE (adding a label to the CMP instruction)

C to Assembly

C Program	Assembly Program

`int i;`
`int sum = 0;`
`for (i = 1; i <= 22;`
`i++)`
`  sum += i;`	

C to Assembly ANS

C Program	Assembly Program 1
int i;	mov r1, #0 /* r1 ← 0 */
int sum = 0;	mov r2, #1 /* r2 ← 1 */
for (i = 1; i <= 22;	loop:
i++)	cmp r2, #22 /* compare r2 and 22 */
sum += i;	bgt end /* branch if r2 > 22 to end */
	add r1, r1, r2 /* r1 ← r1 + r2 */
	add r2, r2, #1 /* r2 ← r2 + 1 */
	b loop
	end:

C Program	Assembly Program 2
int i;	mov r1, #0 /* r1 ← 0 */
int sum = 0;	mov r2, #1 /* r2 ← 1 */
for (i = 1; i <= 22;	b check_loop /* unconditionally jump at the
i++)	end of the loop */
sum += i;	loop:
	add r1, r1, r2 /* r1 ← r1 + r2 */
	add r2, r2, #1 /* r2 ← r2 + 1 */
	check_loop:
	cmp r2, #22 /* compare r2 and 22 */
	ble loop /* branch if r2 <= 22 to the
	beginning of the loop */
	end:

C to Assembly ANS

- Assembly Program 1:
  - Two branches per iteration in the steady state: a conditional (cmp + bgt) and an unconditional b loop. That increases dynamic branch count.
- Assembly Program 2:
  - Only one conditional branch per iteration (the ble), after the body — fewer dynamic branches overall. In steady-state this is usually faster.
  - Better steady-state throughput and fewer branch misprediction opportunities in hot loops (one backward taken conditional is a canonical, well-predicted pattern).

Assembly Programming

- Write a program that reverses the bits in a register, such that the register containing d31,d30,d29…d1,d0 now contains d0,d1,…d29,d30,d31.

Assembly Programming ANS

- ANS:
  - ; Reverse bits in r0, result in r1
  - MOV r1, #0 ; Initialize result
  - MOV r2, #32 ; Bit counter
  - reverse_loop:
  - CMP r2, #0
  - BEQ reverse_end
  - LSL r1, r1, #1 ; Shift result left
  - AND r3, r0, #1 ; Get LSB of r0
  - ORR r1, r1, r3 ;Add to result
  - LSR r0, r0, #1 ; Shift r0 right
  - SUB r2, r2, #1 ; Decrement counter
  - B reverse_loop
  - reverse_end:

```
; Reverse bits in r0, result in r1
MOV r1, #0          ; Initialize result
MOV r2, #32         ; Bit counter

reverse_loop:
    CMP r2, #0
    BEQ reverse_end
    LSL r1, r1, #1   ; Shift result left
    AND r3, r0, #1   ; Get LSB of r0
    ORR r1, r1, r3   ;Add to result
    LSR r0, r0, #1   ; Shift r0 right
    SUB r2, r2, #1   ; Decrement counter
    B reverse_loop

reverse_end:
```

- **Example with 8 bits:**
- Let input r0 = 0b1011_0010
- Initialize r1 = 0, r2 = 8 (bit counter), and r3 is the scratch for the extracted bit; the loop body runs 8 times.
- **Iteration 1**
- LSL r1, r1, #1: r1 = 0 << 1 = 0.
- AND r3, r0, #1: r3 = 0b1011_0010 & 1 = 0 (LSB of r0 is 0).
- ORR r1, r1, r3: r1 = 0 | 0 = 0.
- LSR r0, r0, #1: r0 = 0b0101_1001 (shift right, next LSB becomes current).
- SUB r2, r2, #1: r2 = 7.
- Effect: The first output bit appears at the LSB of r1 and equals original d0.
- **Iteration 2**
- LSL r1: r1 = 0 << 1 = 0.
- AND r3, r0, #1: r3 = 0b0101_1001 & 1 = 1 (new LSB is 1).
- ORR r1, r1, r3: r1 = 0 | 1 = 1 (now r1 = 0b0000_0001).
- LSR r0: r0 = 0b0010_1100.
- SUB r2: r2 = 6.
- Effect: r1 now holds 01, which are the bits "d0 d1".
- On each subsequent iteration, r1 is shifted left before inserting the next bit, so earlier bits move toward the MSB while the newly inserted bit always starts at the LSB; over the whole loop, this builds the reversed word with the earliest extracted bits ending up toward the MSB by the end.

Test for Equal

- Give the different methods to test if two values held in registers r0 and r1 are equal.

Test for Equal ANS

- Give the different methods to test if two values held in registers r0 and r1 are equal.
- ANS:
  - ; Method 1: Compare instruction
  - CMP r0, r1
  - ; Method 2: Exclusive OR
  - EORS r2, r0, r1   ; If equal, result is zero and Z flag set
  - ; Method 3: Subtract and test
  - SUBS r2, r0, r1   ; If equal, result is zero and Z flag set
  - The effect on flags are the same for CMP and SUBS, except CMP discards the result of the subtraction

Review

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

Conditional Instructions

- Write the following ARMv7 instructions:
  - Add registers r3 and r6 only if N is clear (from a previous instruction). Store the result in register r7.
  - Multiply registers r7 and r12, put the results in register r3 only if C is set and Z is clear (from a previous instruction)
  - Compare registers r6 and r8 only if Z is clear (from a previous instruction)

Conditional Instructions ANS

- Write the following ARMv7 instructions:
  - Add registers r3 and r6 only if N is clear. Store the result in register r7.
  - Multiply registers r7 and r12, put the results in register r3 only if C is set and Z is clear
  - Compare registers r6 and r8 only if Z is clear
- ANS:
  - ; Plus, Positive or Zero, (N=0). Add r3 and r6 only if N is clear, store in r7
  - ADDPL r7, r3, r6
  - ; HI, Unsigned Higher, (C=1 & Z=0). Multiply r7 and r12, put result in r3 only if C is set and Z is clear
  - MULHI r3, r7, r12
  - ; NE, Not equal (Z=0). Compare r6 and r8 only if Z is clear
  - CMPNE r6, r8

Assembly to C

- Write the equivalent C program for the following assembly code, assuming registers and C variables are related as (x=r0, y=r1). (Variables in C are in memory, and load/store assembly instructions are omitted here for brevity.) For example:

Assembly Program	C Program

`CMP   r0, #5`
`MOVEQ r0, #10`
`BLEQ  fn`
	`if (x == 5) {`
	`  x = 10;`
	`  fn(x);`
	`}`

Assembly Program	C Program

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

Assembly Program	C rogram

`CMP   r0, #'A’`
`CMPNE r0, #'B'`
`MOVEQ r1, #1`	

Assembly to C ANS

- Write the equivalent C program for the following assembly code, assuming registers and C variables are related as (x=r0, y=r1). (Variables in C are in memory, and load/store assembly instructions are omitted here for brevity.)

Assembly Program	C Program
CMP r0, #5	if (x == 5) {
MOVEQ r0, #10	x = 10;
BLEQ fn	fn(x);
	}

Assembly Program	C Program
CMP r0, #0	if (x <= 0)
MOVLE r0, #0	x = 0;
MOVGT r0, #1	else
	x = 1;

Assembly Program	C Program
CMP r0, #'A’	if (c == ‘A’ || c == 'B')
CMPNE r0, #'**B**'	y = 1;
MOVEQ r1, #1	

Conditional Execution

- Rewrite the assembly program to use conditional execution statements.

Assembly Program	Assembly Program with Cond. Exec
CMP r3, #0 **BEQ** next **ADD** r0, r0, r1 **SUB** r0, r0, r2 next …	

Conditional Execution ANS

- Rewrite the assembly program to use conditional execution statements.

Assembly Program	Assembly Program with Cond. Exec

`CMP   r3, #0`
`BEQ   next`
`ADD   r0, r0, r1`
`SUB   r0, r0, r2`
`next`
`…`
	`CMP   r3, #0`
	`ADDNE r0, r0, r1`
	`SUBNE r0, r0, r2`
	`...`