Chapter 6
Control Flow in Assembly
Exercises

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

- Write assembly program for pseudocode.

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

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

C to Assembly

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

- 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: What is wrong?

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

C to Assembly

C Program	Assembly Program

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

C Program	Assembly Program

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

C Program	Assembly Program

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

C to Assembly: What is wrong?

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:`

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

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`
	
	`DONE`

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 = array base address, R1 = i`
	`MOV R0, 0x60000000`
	`MOV R1, #199`
	
	`FOR`

- 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

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

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,d3.

Test for Equal

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

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)

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`	

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 …	