Assembly Programming

Lecture 07

Josh Brake

Harvey Mudd College

Outline

- Compilation process overview
- C to assembly examples
  - Arithmetic
  - Logical
  - Conditional execution
  - Loops
- Design Example

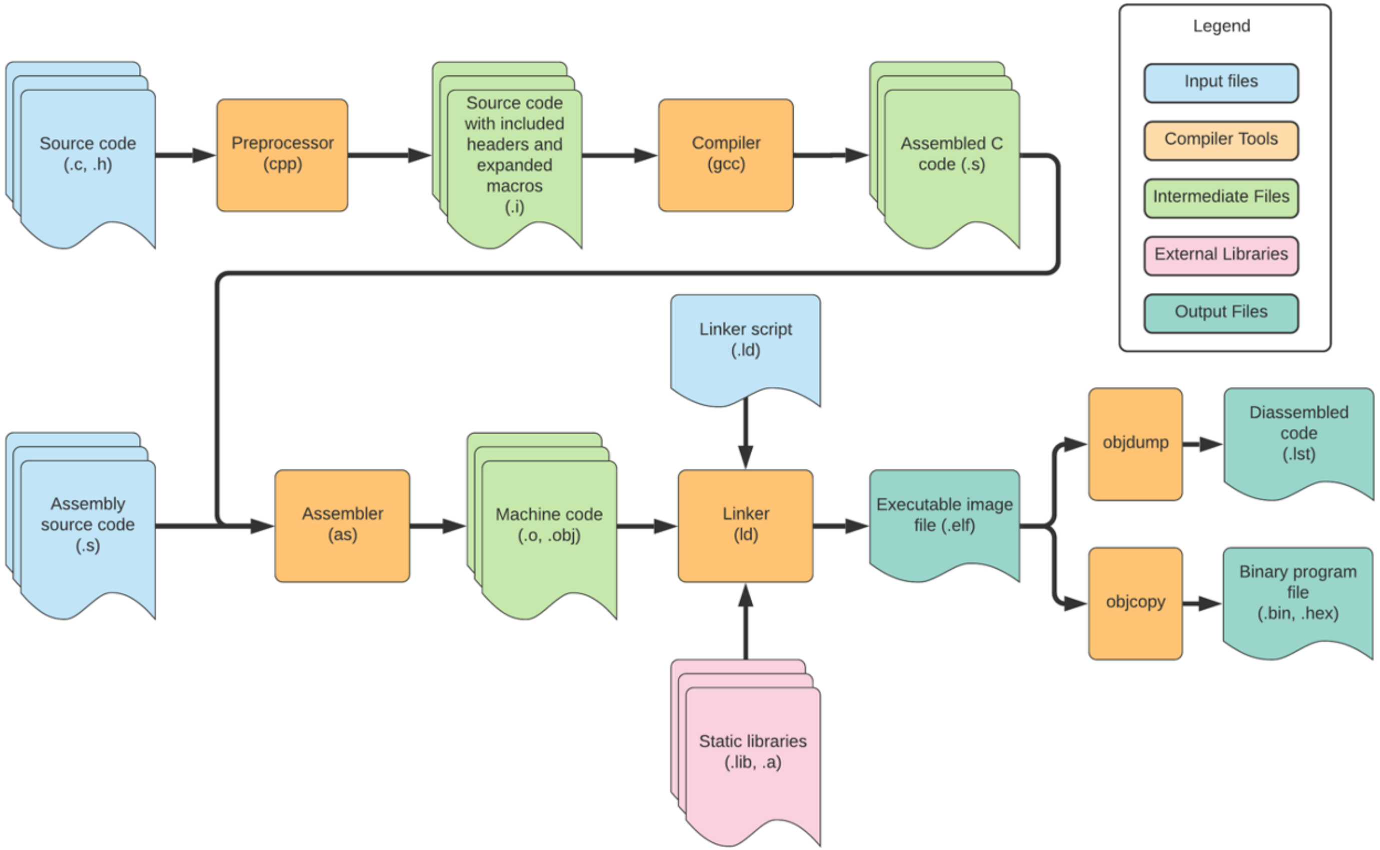
Learning Objectives

By the end of this lecture you should be able to…

- List the steps of the program compilation process
- Recall the assembly idioms for common C programming structures

Compilation Process

This example is for the GNU Compiler Collection (gcc)



C to Assembly Examples

Arithmetic Ex. 1

C

```c
a = b + c;
```

ARM Assembly

```
ADD R0, R1, R2
```

Arithmetic Ex. 2

C

```c
a = b + 2 * c - d;
```

ARM Assembly

```
ADD R0, R1, R2 LSL #1
SUB R0, R0, R3
```

Arithmetic Ex. 3

C

```c
a = d / 4;
```

ARM Assembly

```
ASR R0, R3 #2 ; if d is signed
LSR R0, R3 #2 ; if d is unsigned
```

Logical Ex. 1

C

```c
a = b & c;
```

ARM Assembly

```
AND R0, R1, R2
```

Logical Ex. 2

C

```c
a = b | c;
```

ARM Assembly

```
ORR R0, R1, R2
```

Logical Ex. 3

C

```c
a = b ^ c;
```

ARM Assembly

```
EOR R0, R1, R2
```

Logical Ex. 4

C

```c
a = b << c;
```

ARM Assembly

```asm
LSL R0, R1, #4
```

Logical Ex. 5

C

```c
a = b << c;
```

ARM Assembly

```
ASR R0, R1, R2
```

Conditional Execution Ex. 1

C

```c
if (a) b = 1;
```

ARM Assembly

```
// ARM v7
CMP R0, #0
MOVNE R2, #1

// ARM Thumb-2
CMP R0, #0
IT NE
MOVNE R2, #1
```

Conditional Execution Ex. 2

C

```c
if (a != b) c = d;
```

ARM Assembly

```
// ARM v7
TEQ R0, R1
MOVNE R3, R4

// ARM Thumb-2
CMP R0, R1
IT NE
MOVNE R3, R4
```

Conditional Execution Ex. 3

C

```c
if (a) c = 3;
```

ARM Assembly

```
// ARM v7
CMP R0, #0
MOVNE R3, #3

// ARM Thumb-2
CMP R0, #0
IT NE
MOVNE R3, #3
```

Conditional Execution Ex. 4

C

```
if (a > b) {
  // do stuff 1
  }
else {
  // do stuff 2
  }
```

ARM Assembly

```
 CMP R0, R1
 BLE else
 // stuff1 goes here
 B done
else:
  // stuff2 goes here
done:
```

Conditional Execution Ex. 5

C

```c
if (a > b) c = 1;
else c = 0;
```

ARM Assembly

```
// ARM v7
 CMP R0, R1
 MOVGT R2, #1
 MOVLE R2, #0
// ARM Thumb-2
 CMP R0, R1
 ITE GT
 MOVGT R2, #1
 MOVLE R2, #0
```

Loops Ex. 1

C

```c
int sum = 0, i = 0;
// sum in R0, i in R1

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

ARM Assembly

```
  MOV R0, #0
  MOV R1, #0
loop:
  CMP R1, #10
  BGE done
  ADD R0, R0, R1
  ADD R1, R1, #1
  B loop
done:
```

Loops Ex. 2

C

```c
int i, j; // in R1, R2
int q; // in R3

for (i = 2; i < 8; i++)
  for (j = 1; j < i; j++)
    q = q + i - j;
```

ARM Assembly

```
  MOV R1, #2
loop_i:
  CMP R1, #8
  BGE done_i
loop_j:
  CMP R2, R1
  BGE done_j
  ADD R3, R3, R1
  SUB R3, R3, R2
  ADD R2, R2, #1
  B loop_j
done_j:
  ADD R1, R1, #1
  B loop_i
done_i:
```

Loops Ex. 3

C

```c
int i = 0; // in R1
unsigned int a1[20], a2[20];
// in R4, R5
for (i = 0; i < 20; i++){
  a1[i] = a2[i]/2;
}
```

ARM Assembly

```
  MOV R1, #0
loop:
  CMP R1, #20
  BGE done
  LDR R6, [R4, R1, LSL #2]
  LSR R6, R6, #1
  STR R6, [R5, R1, LSL #2]
  ADD R1, R1, #1
  B loop
done:
```

Loops Ex. 4

C

```c
i = 1;
j = 0;
while (i <= 2048){
  a1[j++] = i;
  i = i * 2;
}
```

ARM Assembly

```
  MOV R1, #1
  MOV R2, #0
while:
  CMP R1, #2048
  BGT done
  STR R1, [R4, R2]
  ADD R2, R2, #1
  LSL R1, R1, #1
  B while
```

Loops Ex. 5

C

```c
char * str1, str2;
// R4, R5
int i = 0;

do {
  str2[i] = str1[i];
}
while (str1[i++]);
```

ARM Assembly

```
  MOV R1, #0
do:
  LDRB R6, [R4, R1]
  STRB R6, [R5, R1]
  CMP R6, #0
  ADD R1, R1, #1
  BNE do
```

Design Example: Low-pass Filter

Problem Statement

Design a 4-sample running average filter for the following data

```
x = [42,54,60,72,78,86,100,112,124,130]
```

Steps

1. Write C code
2. Translate to assembly

C Code

```c
// Algorithm
// i: R0, j: R1, sum: R2, a: R3, size: R4

size = 10
for (i=0; i<size-4; i++) {
  sum = 0;
  for (j=0; j<4; j++)
      sum += a[i+j];
  a[i] = sum / 4;
}
```

Assembly Code

```
// Directives
.syntax unified // Specify the syntax for the file
.cpu cortex-m4  // Target CPU is Cortex-M4
.fpu softvfp    // Use software libraries for floating-point operations
.thumb          // Instructions should be encoded as Thumb instructions

// Define main globally for other files to call
.global main

// Create test array of bytes. Change this for different test cases.
// This will get loaded to the RAM by the startup code
.data
src:
    .int 42, 54, 60, 72, 78, 86, 100, 112, 124, 130
.size src, .-src

dst:
    .fill 128, 4, 0
.size dst, .-dst
```

Assembly Code

```
.text
// The main function
.type main, %function
main:
    ldr r3, =src // load base address of src into R3
    ldr r6, =dst // load base address of dst into R6
    mov r4, #6 // compute size - 4 for comparison
    mov r0, #0 // i = 0
loop_i:
    cmp r0, r4 // i < size -4
    bge done_i // no: finish for i loop
    mov r2, #0 // sum = 0
    mov r1, #0 // j = 0
loop_j:
    cmp r1, #4 // j < 4?
    bge done_j // no: finish for j loop
    add r5, r0, r1 // i + j
    ldr r5, [r3, r5, lsl #2] // a[i+j]
    add r2, r2, r5 // sum += a[i+j]
    add r1, r1, #1 // j++
```

Wrap up

- Assembly programming is most straightforward when you have a particular construct in a higher-level language like C in mind.
- Pay special attention to details like variable types (signed vs. unsigned), sizes, and the addressing modes (e.g., byte vs. word).
- Basic flow is load data into registers from memory, do something with the loaded data, store the result back in memory.