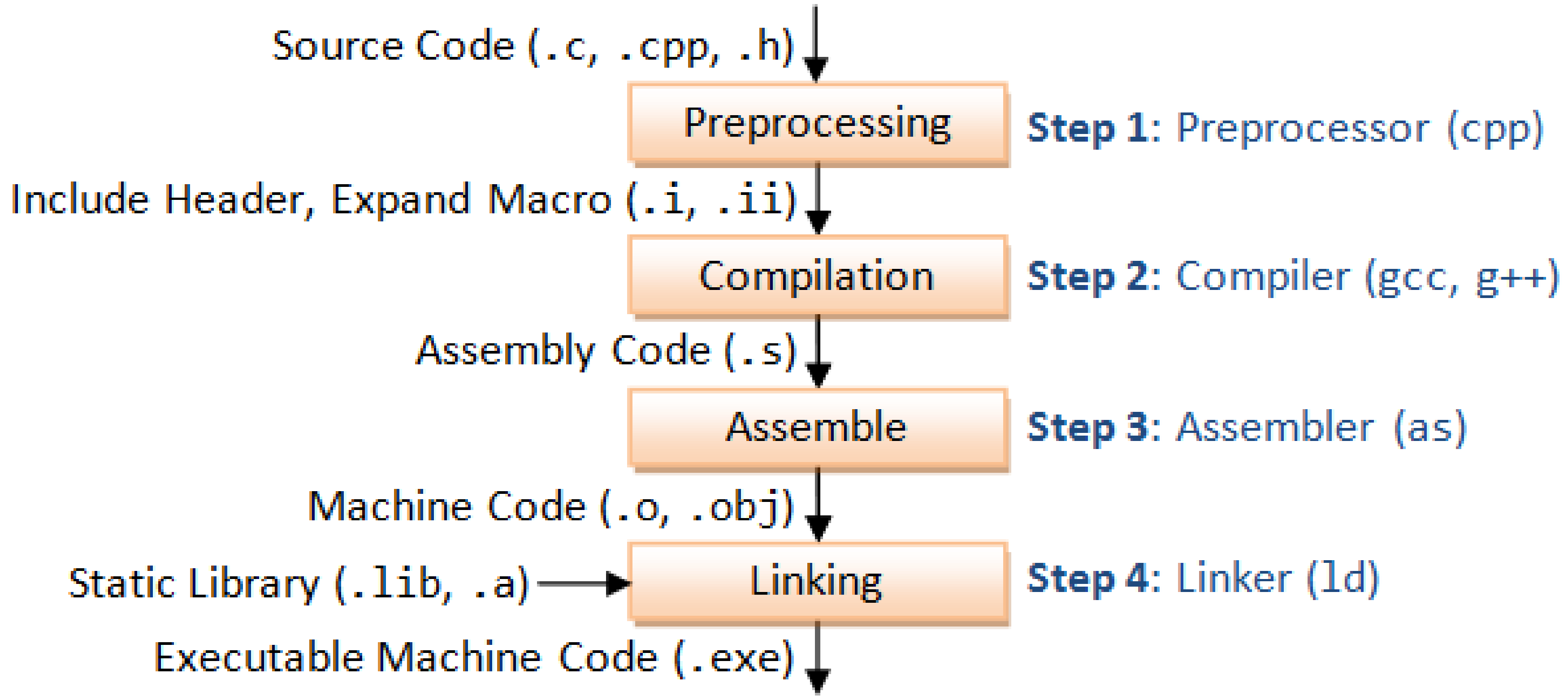
CSE 211: Introduction to Embedded Systems

Section 2

Contact Information

tasneem.awaad@eng.asu.edu.eg

Software Build Process


Source Code (.c, .cpp, .h)
Preprocessing
Step 1: Preprocessor (cpp)
Include Header, Expand Macro (.i, .ii)
Compilation
Step 2: Compiler (gcc, g++)
Assembly Code (.s)
Assemble
Step 3: Assembler (as)
Machine Code (.o, .obj)
Static Library (.lib, .a)
Linking
Step 4: Linker (ld)
Executable Machine Code (.exe)

General Layout of Assembly

- {label} {instruction|directive|pseudo-instruction} {;comment}

Addressing Modes (Immediate Addressing)

- MOV
  - Format: **MOV Rn, Op2**
  - **Op2** can be register or #immediate
  - MOV R0,#0x25
- LDR Rn, =const to move any 32-bit value into a register
  -

Addressing Modes

- Load/Store memory
  - Register indirect Addressing Mode
  - PC relative Addressing Mode
  - PUSH and POP Register Addressing Mode

Addressing Modes

- Load/Store memory
  - Register indirect Addressing Mode
    - **Regular Register indirect**
      - LDR R7, [R5]
      - R5 unchanged, R7 = Mem[R5]
    - **With Immediate Offset:**
      - LDR R7, [R5, #4]
      - R5 unchanged, R7 = Mem[R5 + 4]
    - **With Register Offset:**
      - LDR R7, [R5, R4]
      - R5 Unchanged, R7 = Mem[R5 + R4]

Addressing Modes

- Load/Store memory
  - Register indirect Addressing Mode
    - **With Pre indexed Immediate Offset:**
      - LDR R7, [R5, #4]!
      - R5 = R5 + 4
      - R7 = Mem[R5]
    - **With Post indexed Immediate Offset:**
      - LDR R7, [R5], #4
      - R7 = Mem[R5]
      - R5 = R5 + 4
    - **With Shifted Register Offset:**
      - LDR R7, [R5, R4, LSL #2]
      - R5 Unchanged, R7 = Mem[R5 + R4<<2]

Sheet 2

- What are the differences between the following three instructions?

```
LDRB R0, [R1]          LDRH R0, [R1]          LDR R0, [R1]
```

Answer

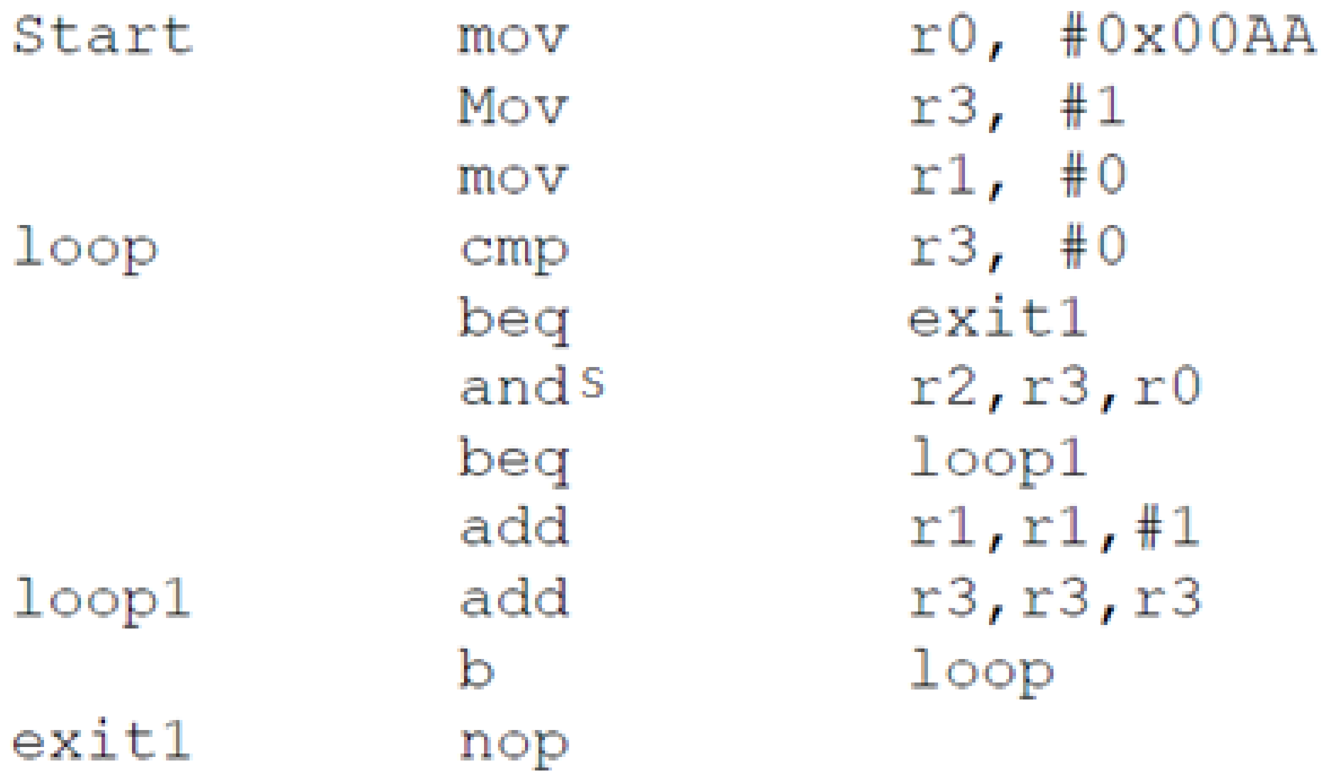
- LDR Load 32-bit word
- LDRH Load 16-bit unsigned halfword
- LDRB Load 8-bit unsigned byte

Sheet 2

- Translate the below C code for counting the number of occurrence of ones in R0 into ARM assembly code assuming that the initial value of r0=0x00AA, using the registers indicated by the variable names.

```
r3 = 1;
r1 = 0;
while (r3 != 0) {
    if ((r0 & r3) != 0) {
        r1 = r1 + 1;
    }
    r3 = r3 + r3;
}
```

Answer



```
Start       mov      r0, #0x00AA
            Mov      r3, #1
            mov      r1, #0
loop        cmp      r3, #0
            beq      exit1
            ands     r2,r3,r0
            beq      loop1
            add      r1,r1,#1
loop1       add      r3,r3,r3
            b        loop
exit1       nop
```

Sheet 2

- Explain what an ARM processor accomplishes in terms of accessing and changing its registers when it executes a BEQ instruction

Answer

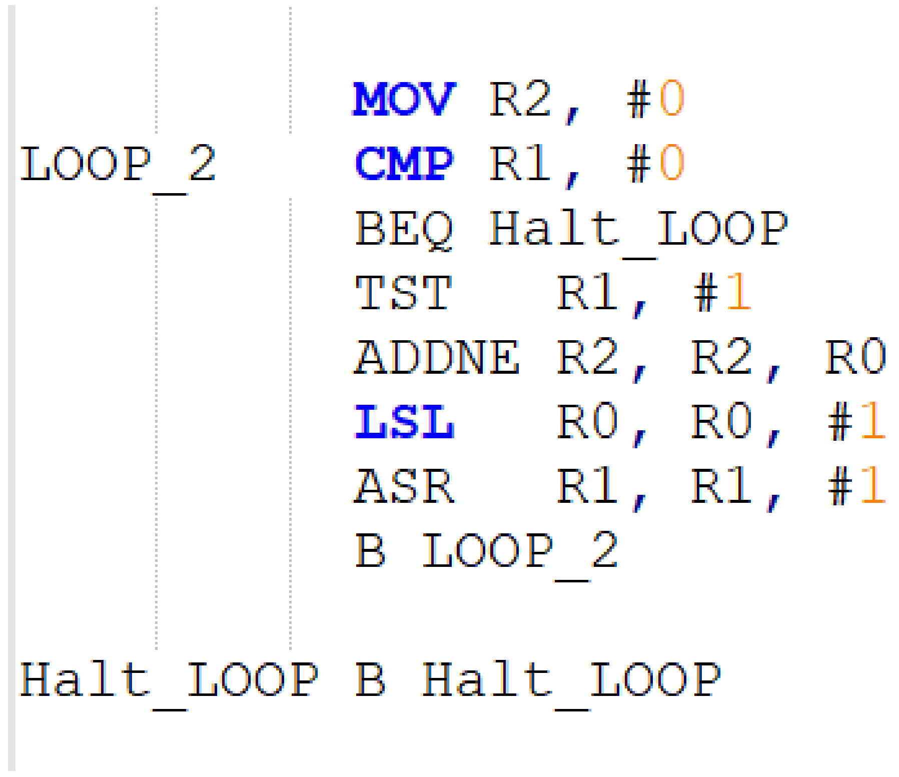
- It looks at the Z flag to see whether the Z flag is 0 or 1. If the Z flag is 1, then it changes R15 (the program counter) to the address named within the instruction. If the Z flag is 0, then R15 will be increased by 4 (so that the next instruction executed is the next instruction after the BEQ instruction).

Sheet 2

- Translate the below C fragment into an equivalent ARM assembly language program, using registers corresponding to the variable names. Assume r0 and r1 hold signed values.

```
r2 = 0;
while (r1 != 0) {
    if ((r1 & 1) != 0) {
        r2 += r0;
    }
    r0 <<= 1;
    r1 >>= 1;
}
while (1); // halting loop
```

Answer



```
            MOV R2, #0
LOOP_2      CMP R1, #0
            BEQ Halt_LOOP
            TST    R1, #1
            ADDNE R2, R2, R0
            LSL    R0, R0, #1
            ASR    R1, R1, #1
            B LOOP_2

Halt_LOOP B Halt_LOOP
```

Sheet 2

- Translate the below C code into ARM assembly code, using the registers indicated by the variable names. The C code presumes that r0 holds the address of the first entry of an array of integer values, and r1 indicates how many elements the array holds; the code removes all adjacent duplicates from the array.

```
r3 = 1;
for (r2 = 1; r2 < r1; r2++) {
    if (r0[r2] != r0[r2 - 1]) {
         r0[r3] = r0[r2];
         r3 += 1;
    }
}
r1 = r3;
```

Answer

```
        MOV R3, #1                     ; counter for nonduplicated items
        MOV R2, #1                     ; loop iterator
        MOV R1, #Array_Size
LOOP1   CMP R2, R1                     ; R2- R1
        BGE Done                       ; R2> R1 jump to Done
        LDR R4, [R0,R2, LSL #2 ]       ; R4= MEM[R0+R2*4]
        SUB R5, R2, #1                 ; R5=R2-1
        LDR R5, [R0, R5, LSL #2]       ; R5= MEM[R0 + (R2-1)*4]
        CMP R4, R5                     ; R0[R2]!= R0[R2-1] --> R0[i]!= R0[i-1]
        STRNE R4, [R0,R3, LSL #2 ]     ; R0[R3]= R0[R2], Z flag=0 N flag
        ADDNE R3, R3, #1               ; R3=R3+1
        ADD   R2, R2, #1               ; R2=R2+1
        B LOOP1

Done   MOV R1, R3
```

Syntax of Arithmetic Instructions

- op{cond}{S} Rd, Rn, Operand2
  - op -> operation such as ADD
  - cond (EQ,NE, GT,..etc.)-> is an optional condition code
  - S -> is an optional suffix. If S is specified, the condition code flags are updated on the result of the operation

Startup File

- A startup file is a piece of code written in assembly or C language that executes before the main() function of our embedded application. It performs various initialization steps by setting up the hardware of the microcontroller so that the user application can run. Therefore, a startup file always runs before the main() code of our embedded application.

- The startup file performs various initializations and contains code for interrupt vector routines.