Tutorial 4

Tuesday, April 20, 2021 11:00 AM

function calling convention — Parameter R0:R3, R12

Q1. Write a complete ARM assembly program for the procedure func2. The procedure func2 calculates this C expression ((X+Y)>>3) – Z and stores its value in R0. Assume X, Y, Z are 32-bit signed numbers. X, Y, Z are defined in the memory as shown

```
        AREA  mydata, DATA, READONLY
X       DCD   -20
Y       DCD   -60
Z       DCD   -20
```

```
func2   LDR  R0, =X
        LDR  R0, [R0] ; X
        LDR  R1, =Y
        LDR  R1, [R1] ; Y
        ADD  R0, R0, R1
        ASR  R0, #3
        LDR  R2, =Z
        LDR  R2, [R2] ; Z
        SUB  R0, R0, R2
        BX   LR
```

Q2. In a digital clock embedded system, you need to implement a function that lets a user pressing a button to display the day of the year. Write an Embedded C function that takes 3 parameters of the day, month and year, performs proper checks on all inputs and returns the day of year (1 – 366). Leap year is that divisible by 4 and 400 but not divisible by 100.

```
static int days_per_month[2][13] = { {0,31,28,31,30,31,30,31,31,30,31,30,31},
                                     {0,31,29,31,30,31,30,31,31,30,31,30,31} };

int day_of_year(int year, int month, int day) {
    int leap = year%4==0 && (year%100!=0 || year%400==0) ? 1 : 0;
    if(month<1 || month>12) return -1;
    if(day<1 || day > days_per_month[leap][month]) return -1;
    for(int i=1; i<month; i++) day += days_per_month[leap][i];
    return day;
}
```

getDayOfYear(20, 4, 2021);

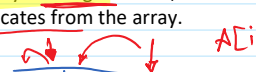
31 + 28 + 31 + 20

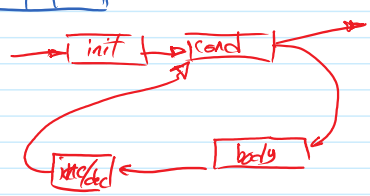
400 ✓ divisible by 4 ✓ 2000 ✓ leap
not divisible by 100 ✓ divisible by 100 ~ 1900 ✗ not leap

Q3. 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;
```

```
        MOV  R3, #1
        MOV  R2, #1
Loop    CMP  R2, R1
        BGE  exit
```





```
LDR  R4, [R0, R2, LSL #2]
sub  R5, R2, #1
LDR  R5, [R0, R5, LSL #2]
CMP  R4, R5
STR NE R4, [R0, R3, LSL #2]
Add NE R3, #1
Add    R2, #1
b      loop
exit mov  R1, R3
```

Q4. Translate the below C fragment into an equivalent ARM assembly language program, using registers corresponding to the variable names.

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

```
loop   mov R2, #0
       cmp R1, #0
       beq done
loop2  TST R1, #1
       Addne R2, R0
       mov R0, R0, LSL #1
       mov R1, R1, ASR #1
       b   loop
done   b   done
```

```
CMP A, B
(A-B) ⇒ set flags
CMN
TST A, B  (A&B)
```

```
And R1, R1, #1
CMP R1, #0
```

And S R1, R1, #1

```
movs R1, R1, ASR #1
bne  loop2
```