ESET 349 – Microcontroller Architecture

Loops

Dr. Muhammad Faeyz Karim

More on Loops

- We have encountered loops before. Now we will cover looping in more details.
- Looping actually interferes with the 3-stage pipeline in ARM architecture. This reduces the efficiency of the pipeline.
- The reason is simple, since it involves a conditional execution of a branching instruction, it is not possible to fetch the next instruction in advance.
- So unnecessary branching should be avoided for efficiency sake.

WHILE LOOPS

- While loops evaluate the loop condition before the loop body.

```
MOV r3, #0x64
B Test
Loop …
      ….    ;      instructions
Test        ;     evaluate condition
      BNE  Loop
```

For Loops

- Example in C language

  ```
  for (j=0; j<10; j++) {instructions}
  ```

- In assembly

```
        MOV     r1, #0          ;j=0
Loop    CMP     r1, #10         ;j<10?
        BGE     Done            ;if j >= 10, finish
        …                       ;instructions
        ADD     r1, r1, #1      ;j++
        B       Loop
Done
```

Count down loops

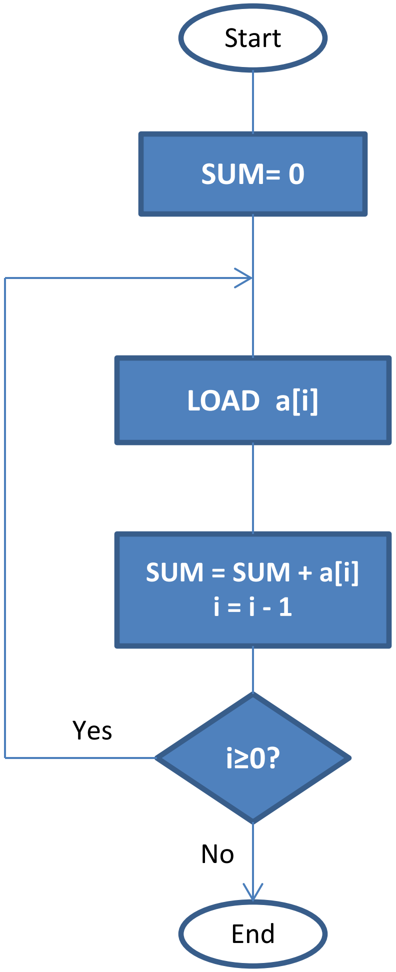
- In cases when a count down loop can be used instead of a count up loop, it should be used.
- A CMP instruction can be saved.

```
        MOV         r1, #10         ;j = 10
Loop    …
        …                           ;instructions
        SUBS        r1, r1, #1      ;j = j-1
        BNE         Loop            ;if j= 0, finish
Done
```

```
            AREA      Prog8b, CODE, READONLY
SRAM_BASE   EQU   0x40000000
            ENTRY
            MOV     r0, #0              ;i
            ADR     r1, arrayb          ;load address of array b
            MOV     r2, #SRAM_BASE ; a[i] starts here
Loop        CMP     r0, #8              ;i = 8?
            BGE   done
            RSB     r3, r0, #7          ;index = 7-i
            LDRB    r5, [r1, r3]        ;load b[7-i]
            STRB    r5, [r2, r0]        ;store into a[i]
            ADD     r0,r0, #1           ;i++
            B       Loop
done        B       done
            ALIGN
arrayb      DCB     0xA, 0x9, 0x8, 0x7, 0x6, 0x5, 0x4, 0x3
            END
```

Summation example

- Lets look at a program that sum six 32-bit integers.
- The flow chart of the program is given on the right.



```
     AREA Prog8c, CODE, READONLY
     ENTRY
        MOV     r0, #0            ;sum =0
        MOV     r1, #5            ;# of elements -1
        ADR     r2, arraya        ;load start of array
Loop  LDR r3, [ r2,r1, LSL #2] ;load value from memory
        ADD     r0, r3, r0        ;sum += a[i]
        SUBS    r1, r1, #1        ;i=i-1
        BGE     Loop              ;loop only if i>= 0
done  B         done
        ALIGN
arraya  DCD   -1, -2, -3, -4, -5, -6
        END
```

DO… WHILE LOOPs

- Structure as follows:

```
LOOP  ….            ; loop body
      ….            ; evaluate condition
    BNE  LOOP
EXIT  ….
```

More on Flags

- Flags are based on the results of comparisons or ALU operations if the S suffix is added.
- Flags can be used to control loops.
- Flags can also be used to control execution of instructions !!!

Condition codes

Field Mnemonic	Condition Code Flags	Meaning
EQ	Z set	Equal
NE	Z clear	Not equal
CS/HS	C set	Unsigned ≥
CC/LO	C clear	Unsigned <
MI	N set	Negative
PL	N clear	Positive or zero
VS	V set	Overflow
VC	V clear	No overflow
HI	C set and Z clear	Unsigned >
LS	C clear and Z set	Unsigned ≤
GE	$N \geq V$	Signed ≥
LT	$N \neq V$	Signed <
GT	Z clear, $N = V$	Signed >
LE	Z set, $N \neq V$	Signed ≤
AL	Always	Default

Conditional Execution

- Branches should be reduced for efficiency sake.
- Removing a branch operation will not only improve execution time but also reduces code size.
- Conditional execution provides this capability.

FINAL EXAMPLE(GCD)

- The Greatest Common Divisor algorithm by Euclid is presented as follows.

```
while (a != b) {   /* a and b positive nos */
    if (a>b) a = a – b;
    else b = b – a;
    }
```

- E.g. a = 18, b = 6

  first pass : a = 12, b = 6

  second pass : a = 6, b = 6 (answer = 6)

Assembly program 1

- Assume r0 contain ***a*** and r1 contain ***b***

```
gcd     CMP  r0, r1          ; a>b?
        BEQ  end             ; if a = b we’re done
        BLT  less            ; a<b branches
        SUB  r0, r0, r1      ; a = a-b
        B    gcd             ; loop again
less
        SUB  r1, r1, r0      ;b = b –a
        B    gcd
```

Better Assembly Program

```
gcd   CMP         r0, r1
      SUBGT       r0, r0, r1
      SUBLT       r1, r1, r0
      BNE         gcd
; no of branches reduced from 4 to 1!!!
```