COL216 Lecture 2

Rishabh Dhiman

10 January 2022

1 Control Flow

pc is a program counter

```
func:
        mov pc, lr
```

```
void func() {
        return;
}
```

```
X: func();

Y: func();
```

```
bl func

bl func
```

Special registers

1. r15 – pc – program counter
2. r14 – lr – link register
3. r13 – sp – stack pointer

How to have nested or recursive calls?

1.1 Passing Parameters

1. via registers

```
caller:
        move parameters into registers
        bl callee
        take result from register
```

```
callee:
        access parameters in registers

        mov pc, lr
```

2. via stack

```
caller:
        move parameters into stack
        bl callee
        pop result from stack

callee:
        access parameters in stack

        mov pc, lr
```

1.1.1 Convention

- First 4 parameters in r0, r1, r2, r3.
- Result in r0.
- Beyond this, use stack.
- Callee can destroy r0, r1, r2, r3, r12.
- It should preserve other registers, except pc.
- Caller should preserver r0, r1, r2, r3, r12.

So the "fixed" code is

```
caller:
        save registers
        move parameters into registers
        bl callee
        take result from register
        restore registers

callee:
        save registers
        access parameters in registers
        restore registers
        mov pc, lr
```

Example:

```
g = A[0];
for (i = 1; i < n; ++i)
        g = gcd(g, a[i]);
```

Sum array program:

```
.equ SWI_Exit 0x11
.text
mov r1, #0 // load first element here instead
ldr r3, =AA // address location
add r6, r3, #400
L: ldr r5, [r3, #0]
add r1, r1, r5 // want to replace this with r1 = gcd(r1, r5)
add r3, r3, #4
cmp r3, r6 @ r6 = q
blt L
swi SWI_Exit
.data
AA: .space 400
.end
```

gcd:

```
ldr r4, =AA // start address location
add r6, r4, #400 // end location
mov r0, [r4, #0]
add r4, r4, #4
L: ldr r1, [r4, #0]
bl gcd
add r4, r4, #4
cmp r4, r6 @ r6 = q
blt L
```

Note, we replace r0, r1, r2, r3 here with other things since they may not be preserved by gcd call.

```
gcd: cmp r0, r1
        beq ret
        blt sub10
sub01: sub r0, r0, r1
        b gcd
sub10: sub r1, r1, r0
        b gcd
ret: mov pc, lr
```

2 DP (data processing) instructions

- Arithmetic –

Fill this in
In an arithmetic operation, the second argument could be a constant too.

2.1 Arithmetic

- add
- sub
- rsb – reverse subtract
- adc
- sbc
- rsc

c stands for with carry

2.2 Logical

- and – bit by bit logical AND
- orr – bit by bit logical OR
- eor – bit by bit logical XOR
- bic – bit clear, op1 and not op2

2.3 Move

- mov – dest ← src
- mvn – dest ← not src

2.4 Compare

- cmp – op1 - op2
- cmn – op1 - (-op2)

2.5 Test

Modifies status flags.

- tst – op1 and op2
- teq – op1 eor op2

3 DT (Data Transfer) Instructions

- ldr / str – load / store word
- ldrb / strb – load / store byte
- ldrh / strh – load / store half word
- ldrsb / ldrsh – load signed byte / half word
- ldm / stm – load / store multiple (any subset of registers can be specified)

4 Comparison in ARM

Signed comparison

- equal – beq
- not equal – bne
- greater or equal – bgeq
- less than – blt
- greater than – bgt
- less or equal – ble

Unsigned comparison

- equal – beq
- not equal – bne
- higher or same – bhs
- lower – blo
- higher – bhi
- lower or same – bls

5 Condition Codes and Flags

1. eq – $Z = 1$
2. ne – $Z = 0$
3. hs / cs (C set) $C = 1$ – for unsigned values $(x - y)$ is implemented as $x + (2^n - y) = x + \tilde{y}$.

4. lo / cc (C clear) $C = 0$
5. mi (minus) $N = 1$
6. pl (plus) $N = 0$
7. vs (V set) $V = 1$
8. vc (V clear) $V = 0$
9. hi $C = 1$ and $Z = 0$
10. ls $C = 0$ or $Z = 1$
11. ge $N = V$
12. ge $N \neq V$
13. gt $N = V$ and $Z = 0$
14. le $N \neq V$ or $Z = 1$
15. al ignore all flags – b = bal

6 Status Flags

- N – Negative
- Z – Zero
- C – Carry
- V – Overflow

7 Assembler Directives

.text .data .end

- .space – reserve some space
- .word – reserve + initialize space
- .byte
- .ascii
- .asciz

.equ – equates a symbol to a value

8 I/O in Arm

SWI Instruction – Instruction to invoke some service provided by system software.
Use in ARMSim:

- I/O from/to stdin/stdout
- input/output from/to files
- opening/closing of files
- Halt execution

Example in ARMSim# 1.91

```
ldr r0, =message
swi 0x02 @ write on stdout

message: .asciz "Welcome\n"
```

I/O Example in ARMSim# 2.01

```
ldr r1, =param
mov r4, #1 @ file #1 is stdout
str r4, [r1]
ldr r4, =message
str r4, [r1, #4]
mov r4, #8 @ number of bytes
str r4, [r1, #8]
mov r0, #5 @ code for write
swi 0x123456

param: .word 0, 0, 0
message: .ascii "Welcome\n"
```

9 ISR vs User Mode