CSE 2312 - Exam 1 Notes
March 11, 2021
Luke Sweeney
UT Arlington
Professor Losh

1 Assembly Examples

The following examples may or may not have accompanying `C` code. Most of it is just assembly.

Random Examples

This section just shows some random examples, they're not organized into functions or anything. Most lines are independent.

```
// Add and subtract
ADD R0, R0, R1      // R0 <- R0 + R1
SUB R0, R0, R1      // R0 <- R0 - R1
ADDS R0, R0, R1     // R0 <- R0 + R1 and update flags
SUBS R0, R0, R1     // R0 <- R0 - R1 and update flags
// The flags will be things like the zero flag (z), negative (n), etc. See the manual for more info


// BX LR returns from the assembly function
// It puts the address of the next instruction (from the caller)
// into the correct address.
BX LR


// Form CMP, the result is thrown away and flags are updated
// This is useful for a lot of things. For example, CMP to a number
// and the z flag will be set if they're equal.
CMP R0, R1


// This means R0 - 0, which seems like it does nothing,
// but all the flags are being updated.
// Example, the zero flag Z = 1 if R0 is zero
CMP R0, #0
// This would set Z = 1 if R0 is 5
CMP R0, #5

// This ADDs if the zero flag = 0 (meaning the number is nonzero)
// R0 <- R1 + R2 if ZF = 0
ADDNE R0, R1, R2
//  ^^ NE is called a "condition code"



// Condition codes are in the manual
// Here's an example of using them
// CMP to 0, if R0 is 0 then set Z = 1
CMP R0, #0
// This just adds 5 to R0
// But only if the Z flag is set (which CMP just did)
// So these two lines add 5 only if R0 is 0
ADDEQ R0, R0, #5
```

Reading and writing to a memory location

```c
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>

extern void writeU32(uint32_t* p, uint32_t x);
extern uint32_t readU32(uint32_t* p);

int main(void) {
    // Make an integer on the stack, we just need the mem location
    uint32_t a;
    printf("&a = %p\n", &a);
    // Write this hex number at the address of a
    writeU32(&a, 0x12345678);
    printf("a = 0x%08x\n", readU32(&a));
    return EXIT_SUCCESS;
}
```

```
.global writeU32
.global readU32

.text


// extern void writeU32(uint32_t* p, uint32_t x);
// pointer is in R0, value is in R1
writeU32:
    STR R1, [R0]
    BX LR

// extern uint32_t readU32(uint32_t* p);
// Pointer is in R0
readU32:
    LDR R0, [R0]
    BX LR
```

Is Even

```
is_even:
    // This will bitshift with 1, leaving the result in R0
    // This will leave a 1 in R0 if the number is odd
    AND R0, R0, #1
    // We use this to switch from 0-1 or vice versa
    RSB R0, R0, #1
    // Note: You can also use exclusive or (EOR) to flip the bit
    //       EOR R0, R0, #1
    BX LR
```

is Positive

```
// Return true (1) if positive, false (0) otherwise
// Becuse this accepts an unsigned number, it really just checks if
// the number is zero or nonzero
isPositiveU32:
    // This sets the zero flag = 1 if R0 is a 0
    // if (R0 = 0) set Z = 1
    CMP R0, #0
    // So now, if Z = 1, the argument R0 is 0
    // If Z = 0, the number is not 0, therefore positive
    // becuase it's an unsigned.

    // This moves a 1 (true) into R0 only if Z clear (Z = 0)
    MOVNE R0, #1
    // this moves a 0 (false) into R0 only if Z set (Z = 1)
    MOVEQ R0, #0
    BX LR

isPositiveS32:
```

```
    CMP R0, #0
    // Default case is positive (return true)
    MOV R0, #1
    // Return 0 (false) if the number == 0
    MOVEQ R0, #0
    // return 0 (false) if MI is set
    MOVMI R0, #0
    BX LR
```

Add 64 Bit

```
// uint64_t addU64(uint64_t x, uint64_t y);
// x in R1:R0 (bits 63-32 are in R1, bits 31-0 are in R0)
// Same case for y, in R3, R2
// Results are returned in R1:R0, just how x is passed in.
addU64:
    ADDS R0, R0, R2      // Add with flags
    ADC R1, R1, R3       // Adds with carry
    BX LR
```