Mid Term Review with Examples:

Keys to doing well on the mid term ------------------------------

Things to remember for the midterm:

1. Remember the 15-21 commands for ARM Assembly
2. Know how to write loops
3. Understand bit shifting using arrays (LSL,LSR,AND,OR,XOR, and so on)
4. Carry Bits
5. Arrays

Words of Wisdom from Kruger:

1. Write notes with exams and could give points
2. Can add the logic statements to the math operations
3. WIll tell us what is going to be on the midterm via Canvas

1) Arm Commands Basics:

What does MOV do?:

1. Destination = Location = Intermediate mode (number)
2. Destination = Location = Register value

Example:

Intermediate Mode Mov:

MOV R0, #1

(R0 = 1)

Register Mode Mov:

MOV R0, R1

(R0 = R1)

What does ADD function do?:

1. Destination = Location + Intermediate mode
2. Destination = Location + Register

Examples:

Intermediate Mod Add:

ADD R0, R0,#1

(R0=R0+1)

Register Mode Add:

ADD R0,R1,R2

(R0=R1+R2)

What does SUB function do?:

1. Destination = Location - Intermediate mode
2. Destination = Location - Register

Examples:

Intermediate Mod Subtraction:

SUB R0, R0,#1

(R0=R0-1)

Register Mode Subtraction:

SUB R0,R1,R2

(R0=R1-R2)

What does MUL function do?:

1. Destination = Location \* Intermediate mode
2. Destination = Location \* Register

Examples:

Intermediate Mod Multiplication:

MUL R0, R0,#1

(R0=R0\*1)

Register Mode Multiplication:

MUl R0,R1,R2

(R0=R1\*R2)

There is no no division operation in C++, so an algorithm had to be made

Comments:

@ single line comment

/\* \*/ multiple line comment

Logical Commands:

\*Command that must be first when comparing values and making decisions

[ (Commands) <value 1> <value 2>]

CMP R0, R1 [“Compares” R0 with R1]

\*Commands that come after CMP, that determines how R0 and R1 are being compared, then calls function to do other processes

[ (Commands) {Call function} ]

BGE [“Branch Greater Than or Equal To” → “>=” ]

BGT [“Branch Greater Than” → “>” ]

BLT [“Branch Less Than” → “<” ]

BLE [“Branch Less Than or Equal To” → “<=” ]

BNZ: !=0 [“Branch Not Equal To 0” → “!= 0” ]

BZ: equal zero [“Branch Equal To 0” → “=0” ]

BEQ: == [“Branch Equal To” → “=” ]

NF: != [“Branch Not Equal To” → “!=” ]

2) Loops:

Things need for a loop in assembly:

1. Variables (Registers)
2. Numerical values
3. Comparison statement (CMP)
4. Logical operation (8 in total)

Example Code:

MOV R0, #3

.loop:

ADD R0,R0,#2

CMP R0, #21

BLT .loop

bx lr

Example of how to add sums from 1 to 100:

MOV R1,#0

MOV R0, #1

.loop1:

ADD R1,R1,R0

ADD R0,R0,#1

CMP R0, #100

BLE .loop1

bx lr

Example of how to multiplying sums from 1 to 100:

MOV R1,#1

MOV R0, #1

.loop2:

MUL R1,R1,R0

ADD R0,R0,#1

CMP R0, #100

BLE .loop2

bx lr

Making a loop: Odd numbers from 3 to 21, including 21

MOV r0, #3

.loop:

ADD r0, r0, #2

CMP r0, #21

BLT .loop

Now let's add all numbers from 1 to 100

MOV r1, #0

MOV r0, #1

.loop

ADD r1, r1, r0

ADD r0, #1

CMP r0, #100

BLE .loop

3) Bit shifting:

Look at this document for more information about AND,OR, XOR, and so on: <https://docs.google.com/document/d/1j4DJwb-qzTxocXmPf5ciSNSrx62GTcMM-hU-dWcsb1Y/edit?usp=sharing>

Examples of AND,OR, XOR:

‘AND' & 'OR' MASKING

LSL --> Logical Shift Left @shifts binary values to left

LSR --> Logical Shift Right @shifts binary values to right

let's say r0 = 000101110000011111000000, and we want to set values to 'and' or 'or'.

@'or' sets things to 1

@'and' sets things to 0

to set values, you need to come up with the corresponding binary

r0 = 0001 0111 0000 0111 1100 0000

r1 = 0000 0000 1000 0000 0000 0000

orr r0, r0, r1

r0 = 0001 0111 1000 0111 1100 0000 @by ‘orr’-ing both registers, r0 has changed

let's say

MOV r0, #0x81 1000 0001

ORR r0, r0, #0x08 1000 1001

Now let's say

MOV r0, 0xEC 1110 1100

MOV r1, 0xFB 1111 1011

-------------

AND r0, r0, r1 1110 1000 @0xE8

Now let’s say

MOV r0, 0xA1 1010 0001

MOV r1, 0xB2 1011 0010

--------------

XOR r0, r0, r1 0001 0011

It is recommended when performing these operation to do them in Hexadecimal

4) Carry Bits:

Because the registers we are working with only go from 0000 to 1111, we will occasionally get values that can’t be expressed normally when adding in assembler. For example, let’s say we added 1111 and 1111 together. How would we express this when limited to only four bits? The only way to do this is with carrying. Look below for an example of this:

*Carry: 111*

1111

* 1111

-------

1110 with the carry bit set to 1

What is happening here is that when we add 1 + 1, we get zero, but we also set the carry bit to 1 in the process. When we move one space to the left, we get 1 + 1 = 1 because we are adding the carry bit to the answer, so we’re really doing 1 + 1 (+ 1) = 1. To do the above example in ARM assembler, we must use the ADC and ADCS\* commands to use the carry bit.

\*The ADC and ADCS commands both perform addition with carrying, but if S is specified, it will update the N, Z, C, and V flags\*\* according to the result. You only need to use ADCS if you will use the updated four flags after you get the results of your addition.

\*\*The four flags are **N**(egative), **Z**(ero), **C**(arry), and (o)**V**(erflow) respectively

Here is an example of addition with carrying in assembler:

MOV r0, 0xFF @1111 1111

MOV r1, 0x01 @0000 0001

ADC r0, r0, r1 @0000 0000 with the carry bit set to 1

The result on a bigger register would be 0x100, or 0001 0000 0000

5) Arrays:

R0 points to {3,4,1,2}

R1 is amount of numbers in array

R0 should be the sum of numbers in the array

Accessing arrays in assembler

MOV R2, R0 @R2 = R0 (memory address)

@This operation allows you to delete R0 if you want since R2 has R0 memory address

LDR R0, #=\_\_\_\_\_\_\_\_\_\_\_\_\_ (Location in memory)

LDR R1, [R0] @ R1 =3 @R0 is a location in memory

ADD R0, R0, #4 @ allows you to go to the next number in the array

@ This takes the memory of R0 and applies

0000 0000 0000 0011 -> 0003

0000 0000 0000 0100 -> 0004

How to add the sum of thing from an array:

R0 points to {3,4,1,2}

R1 is amount of numbers in array

R0 should be the sum of numbers in the array

x = {3,4,1,2}

MOV R2,R0 @ R2 points to the x

MOV R0, #0

.loop:

LDR R3, [R2] @ R1 = current element in Array

ADD R0, R0, R3 @R0=R0+R3

ADD R2,R2,#4 @advance to the next thing in the array

SUB R1,R1,#1 @counts down value of things in array

BNT .loop

bx lr