**EXAM 4 CHEATSHEET – CAMERON STARK  
Binary Numbers**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| 16 | 8 | 4 | 2 | 1 |
| 0 | 1 | 1 | 0 | 0 |

= 12

**Binary Operations**

A = 0011 1100

B = 0000 1101

A & B : 1 if both numbers have are a 1, 0 else

A & B = 0000 1100

A | B : 1 if it exists in either, 0 else

A | B = 0011 1101

A ^ B: 1 if it is 1 in either, 0 else or set to 1 in both

A ^ B = 0011 0001

A ~ B: flips the bits

A ~ B = -0111 101

A << #: everything moved left by #

A << 2 = 1111 0000

A >> #: everything moved right by #

A >> 2 = 0000 1111

A[4] = { 0x3333, 0x5555, 0x7777, 0x9999}

B[4] = {0x6666, 0x8888, 0xAAAA, 0xCCCC}

Mask = 0x7

N\_Shift = 8

to\_assign = 5

\*pInt = B

A[0] & A[0] = 0x3333

!A[1] = 0x0

~A[2] = 0x8888

0x000F ^ A[3] = 0x9996

32 / 6 = 0x5

15 – ((15>>2)<<2) = 0x3

~(Mask << N\_shift) = 0xF8FF

B[0] & ~(Mask << N\_shift) = 0x6066

\*(pInt + 1) = 0x8888

\*pInt++ = 0x6666

(\*pInt)++ = 0xAAAA

++(\*pInt) = 0xAAAC

**Pointer Arrays**

int nums[2][3] = { {16, 18, 20}, { 25, 26, 27} }

\*(\*nums) == nums[0][0] = 16

\*(\*nums + 1) == nums[0][1] = 18

\*(\*nums + 2) == nums[0][2] = 20

\*(\*(nums + 1)) == nums[1][0] = 25

\*(\*(nums + 1) + 1) == nums[1][1] = 26

\*(\*(nums + 1) + 2) == nums[1][2] = 27

**Assembly**

functionName PROC

PUSH {r4, lr } ;preserve environment in stack

LDR r4, [r0] ;r4 = \*r0

LSL r2, r2, r3 ;r2 = r2 << r3

MVN r2, r2 ;r2 = ~r2

AND r4, r4, r2 ;r4 = r4 & r2

LSL r1, r1, r3 ;r1 = r1 << r3

ORR r4, r4, r1 ;r4 = r4 | r1

STR r4, [r0] ;\*r0 = r4

POP {r4, pc} ;return

ENDP

//set\_a\_fpio\_pin(&FPIOA -> ODR, (1 << 2));

IMPORT pin

set\_a\_fpio\_pin PROC

PUSH {r4, pin }

LSL r4, r4,2

STR r4, [r0]

POP {r4, pc}

ENDP

**Data Operations**

C-Flag

Adding two unsigned numbers

if result > max(number) then 1

else 0

Subtracting two unsigned numbers

if result < min(number) then 0

else 1

V-Flag

Adding two signed numbers

if two positive number result < 0

or

if two negative number result > 0

then 1

else 0

Subtracting two signed numbers

if positive from negative result > 0

or

if negative from positive result < 0

then 1

else 0

**LDR Functions**

\*(pInt32 + 2) -> LDR r#, [r#, #8]

\*pInt32++ -> LDR r#, [r#], #4

\*++pInt32 -> LDR r#, [r#, #4]!

\*(pInt32 + i) -> LDR r#, [r#, ri, LSL #2]

**STR Functions**

STR r3, [r2] (stores the value of r3, in r2)

**Conditional Execution**

CMP r0, r1 (compares r0 to r1)

BLE <=

BLT <

BGE >=

BLE >=

BEQ ==

CBZ = 0

MOVLE r1, #1 (r1 = 1, if CMP is less than or equal)

MOVGE

MOVLT

MOVGT

Loops

For Loop

task6 PROC

PUSH {r4, lr}

MOV r2, #0 ;sum6

MOV r3, #0 ;i

task6\_loop

CMP r3, r1

BGE task6\_end

LDR r4, [r0], #4

AND r4, #0x0F

ADD r2, r4

ADD r3, #1

B task6\_loop

task6\_end

MOV r0, r2

POP {r4, pc}

BX lr

ENDP

While Loop

task7 PROC

PUSH {r4, lr}

MOV r2, #0 ;sum7

MOV r3, #0 ;i

MOV r4, #0 ;temp i

task7\_while

CMP r3, r0

BGE task7\_end

MOV r4, r3

LSL r4, #2

ORR r4, #0x0F

ADD r2, r4

ADD r3, #1

B task7\_while

task7\_end

MOV r0, r2

POP {r4, pc}

BX lr

ENDP

Do-While

task9 PROC

PUSH {r4, lr}

MOV r2, #0 ;sum9

MOV r3, #0 ;i

MOV r4, #0 ;temp i

task9\_loop

LSL r4, #2

ORR r4, #0x0F

ADD r2, r4

ADD r3, #1

MOV r4, r3

CMP r3, r0

BLT task9\_loop

task9\_end

MOV r0, r2

POP {r4, pc}

BX lr

ENDP

**Floating Point Expressions**

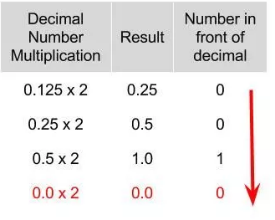
Bias for single precision = 127

Bias for double precision = 1023

Steps for decimal to floating point

85.125

1. Split the whole and decimal portion

85

0.125

2. convert whole to binary

85 -> 1010101

3. convert decimal to binary

.125 -> 0.001

4. combine the two binary values

1010101.001

5. convert to base 2 (shift decimal point until right of first bit)

1.010101001 x 2^6

6. record sign bit (if original value is + then 0 else 1)

Sign = 0

7. get the exponent (bias + 6)

127 + 6 = 133

8. convert the exponent to binary

133 -> 10000101

Exponent = 10000101

9. find the mantissa (which is everything right of the decimal from step 5)

Mantissa = 010101001

10. combine the sign, exponent and mantissa

0 10000101 010101001

Floating point to Decimal

1. Convert to binary

0xC2100000 -> 0b1100\_0010\_0001\_0000\_0000

->0b1\_1000\_0100\_0000\_0000\_0000

2. Get exponent value

0b1000\_0100 = 132

132-127 = 5

3. Get Fraction value

0010 -> 0\*0.5 + 0\*0.25 + 1\*0.125 + 0\*0.0625 +….

= 0.125

4. Get number value

-(1.125)\*2^5

=-36.0

