1. Information representation question: The PowerPC architecture does work with 32 bit numbers, sometimes in unsigned binary, sometimes in two's complement. It is possible to assume that there is a radix point somewhere in the word, which enables us to consider fractional values as well as whole values. The architecture can also move 16 bit and 8 bit quantities. Complete the table below with the appropriate information – dealing with 16 bit numbers, sometimes as integers, sometimes as fixed point values. Remember that the 'p' quantity – in fixed point numbers – identifies the number of places to move the radix point from its normal position at the right of the value.

Number	Bit Pattern	Value of Representation
Value of this bit pattern; 16 bit two's complement integer	1001101010010100	-25964
Generate bit pattern for: in 16 bit two's complement integer	0010010100011110	14,000
Value of this bit pattern: 16 bit two's complement, fixed point, p=8	1000100110010110	-30314
Generate bit pattern for: In 16 bit two's complement, fixed point, p=8	1101010.00111001	$42\ ^{23}/_{32}$
Bit pattern for biggest possible number, fixed point, two's complement, p=8	0111111111111111	Biggest
Bit pattern for smallest possible number, fixed point, two's complement, p=8	1.0000000000000000	Smallest (leftmost on number line)

0110010101101011 +1

0110010101101100 = 25964

14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

14,0000 = 11011010110000

= 0010010100011111

= 0010010100011110

1000100110010110

0111011001101001 +1

0111011001101010 = -30314

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

42 23/32 = 42.71875

00101010.11000101

- 11010101.00111010

11010101.00111001

2. General information question:

a) What is the basic fundamental principle on which all stored program computers are based?

fetch decode execute

b) How is the concept of Memory Mapped I/O demonstrated by the PowerPC architecture?

the MMI/O look like memory so we can write to them. we can read/write them from the memory location they are stored in

c) What instructions in the PowerPC architecture move multiple register values to and from memory?

Lmw
Stmw

d) George is building a vector table for use with the PowerPC. His table starts at location 0x33330000. An Interrupt Service Routine (ISR) has been created to handle both UART interactions and button pushes. This is a fairly large routine, more than 300 instructions. It starts at location 0x33338080. What instruction should the user place at what address to use this ISR when either a button is pushed or the UART needs service?

0x33330000
0x00000500
0x33330700

branch-to-0x33338080 @ 0x33330500

-2

e) What is the purpose of the CTR register?

This is the counter register is count several things we assign it to

f) What are the instruction types? Identify each type and give an example from the PowerPC architecture.

Work - add
Movement - Lmw
program control - Branch
system control - Interrupts

g) Assume that a user has hooked a data source to a bunch of wires controlled by a GPIO IP module, and that the address associated with the module is 0x81480000. Also assume that R4 is set to zero, and that R5 is set to 0x81480000. Give a three instruction sequence that will read into R6 the value on the set of wires.

lis r4, 0x00000000
lis r5, GIE(r1)
stw r5, 0x81480000
Li r5, 0x81480000
stw r5, IPIER(r1)

Li r5, 0(r4)

-1

h) Suppose that the processor clock was set to 268,435,456 hertz. How often would TBU increment?

— Time Base Upper

Always 268,435,456 micro seconds.

-2

3. Instruction execution question 1: The following routine has created by a user for some strange manipulations on data. Hence, there appears to the observer some un-explainable operations (in other words, don't try to make sense out of what this routine does…) The routine "CTIO3" is called in the code below, and the purpose of this question is to determine the final value of the registers *after* the code fragment executes. When the program activity is at the location indicated "before", a snapshot of the registers is given in the "Before" area of the table. Please indicate in the "After" area of the table the register contents when "after" is reached after execution of the subroutine **(remember to enter values only for those registers that have changed).**

```
Addr  Bits          Instruction
3300 60000000  before: nop
3304 4800002D          bl CTIO3
3308 60000000  after:  nop

3330 39401234  CTIO3:  li r10,0x1234
3334 614A5678          ori r10,r10,0x5678
3338 39C00004          li r14,0x04
333c 7DC903A6          mtctr r14
3340 7C6D8670  over:   srawi r13,r3,0x10  # shift right arith immed
3344 4200FFFC          bdnz over
3348 7082F0F0  xyza:   andi. r2,r4,0xf0f0
334c 4182FFFC          bt 2,xyza
3350 7CC74B78          or r7,r6,r9
3354 7D0F6038          and r15,r8,r12
3358 61698281          ori r9,r11,0x8281
335c 4E800020          blr
```

r10 = 12345678
r14 = 0xEEEEEEF2
CTR = 0xEEEEEEF2
r13 = 0xFFFF3333
r2 = 0x44444040
r7 = 0xFFFFFFFF
r15 = 0x88888888
r9 = 0xBBBBBBBB

Before		After	
r0 = 0x00000000	r1 = 0x11111111	r0 =	r1 =
r2 = 0x22222222	r3 = 0x33333333	r2 = 0x44444040	r3 =
r4 = 0x44444444	r5 = 0x55555555	r4 =	r5 =
r6 = 0x66666666	r7 = 0x77777777	r6 =	r7 = 0xFFFFFFFF
r8 = 0x88888888	r9 = 0x99999999	r8 =	r9 = 0xBBBBBBBB
r10 = 0xAAAAAAAA	r11 = 0xBBBBBBBB	r10 = 0x12345678	r11 =
r12 = 0xCCCCCCCC	r13 = 0xDDDDDDDD	r12 =	r13 = 0xFFFF3333
r14 = 0xEEEEEEEE	r15 = 0xFFFFFFFF	r14 = 0xEEEEEEF2	r15 = 0x88888888
LR →	0x00000000	LR →	3330
CTR →	0x00000000	CTR →	0xEEEEEEF2

-3
-4
-7

F = 15
16 = 10

r14, 0x04

0xEEEEEE 4

r4 = 0100 0100 0100 0100
and 1111 0000 1111 0000
0100 0000 0100 0000

0011 0011 0011 0011 0011 0011 0011 0011

FFFF 3333

1110 1110
- 0100 +
1011 1111 0010
EEEF 2

or r6 0110 0110
r9 1001 1001
1111 1111

r11 1011 1011 1011 1011
ori 1000 0010 1000 0001
r9 = 1011 1011 1011 1011

r8 1000 1000
and r12 1100 1100
1000 1000

← r14
10 = 16
shifted 16's

4. Instruction execution question 2: Consider the code segment below. As in the previous question, this question basically is to determine the work done by the instructions, and to modify the registers and the memory segment shown below to reflect the instructions shown. And, once again, don't try to attach any meaning to the work done. **Identify in the register area and the memory area only those values that are changed!**

```
Addr   Bits
11100 3DC00007   before: lis r14,0x4567
11104 61CE3040          ori r14,r14,0x3040
11108 89EE0019          lbz r15,0x19(r14)
1110c 7C855214          add r4,r5,r10
11110 7D495B78          or r9,r10,r11
11114 806E0008          lwz r3,0x08(r14)
11118 912E002C          stw r9,0x2c(r14)
1111c 990E0032          stb r8,0x32(r14)
11120 A1AE000A          lhz r13,0x0A(r14)
11124 B1AE003A          sth r12,0x3A(r14)
```

Before		After	
r0 = 0x00000000	r1 = 0x11111111	r0 =	r1 =
r2 = 0x22222222	r3 = 0x33333333	r2 =	r3 = 0x99AABBCC
r4 = 0x44444444	r5 = 0x55555555	r4 = 0xFFFFFFFF	r5 =
r6 = 0x66666666	r7 = 0x77777777	r6 =	r7 =
r8 = 0x88888888	r9 = 0x99999999	r8 =	r9 = 0xBBBBBBBB
r10 = 0xAAAAAAAA	r11 = 0xBBBBBBBB	r10 =	r11 =
r12 = 0xCCCCCCCC	r13 = 0xDDDDDDDD	r12 =	r13 = 0xBBCC
r14 = 0xEEEEEEEE	r15 = 0xFFFFFFFF	r14 = 0x45673040	r15 = 0xBA

Address	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
45673040	11	22	33	44	55	66	77	88	99	AA	BB	CC	DD	EE	FF	01
45673050	23	45	67	89	AB	CD	EF	FE	DC	BA	98	76	54	32	10	65
45673060													BB	BB	BB	BB
45673070			88								CC	CC				

3040 + 3A = 7A

3040 + 32 = 3072

3040 + 0A = 304A

r14 = 0x45673040 + 10 = 3050

r5 + r10 = r4 : 0101 + 1010 = 1111

r10 or r11 = r9 : 1010 | 1011 = 1011

0x45673040 + 2C = 306C

0x45673040 + 8 = 3048

5. Interrupt question #1: This question deals with setting up interrupts. In the space below provide code for the trainer board system to set up two I/O interrupts (UART and first timer, so second and fourth LSBs) and the PIT interrupt. The code should set up the system to be sensitive to the three interrupts. Also, initialization of the PIT interrupt should set a value to zero; this value will be used later when the ISR is invoked. PIT count value is 0x22222, timer count value is 0x33333. (PIT SPR is 987)

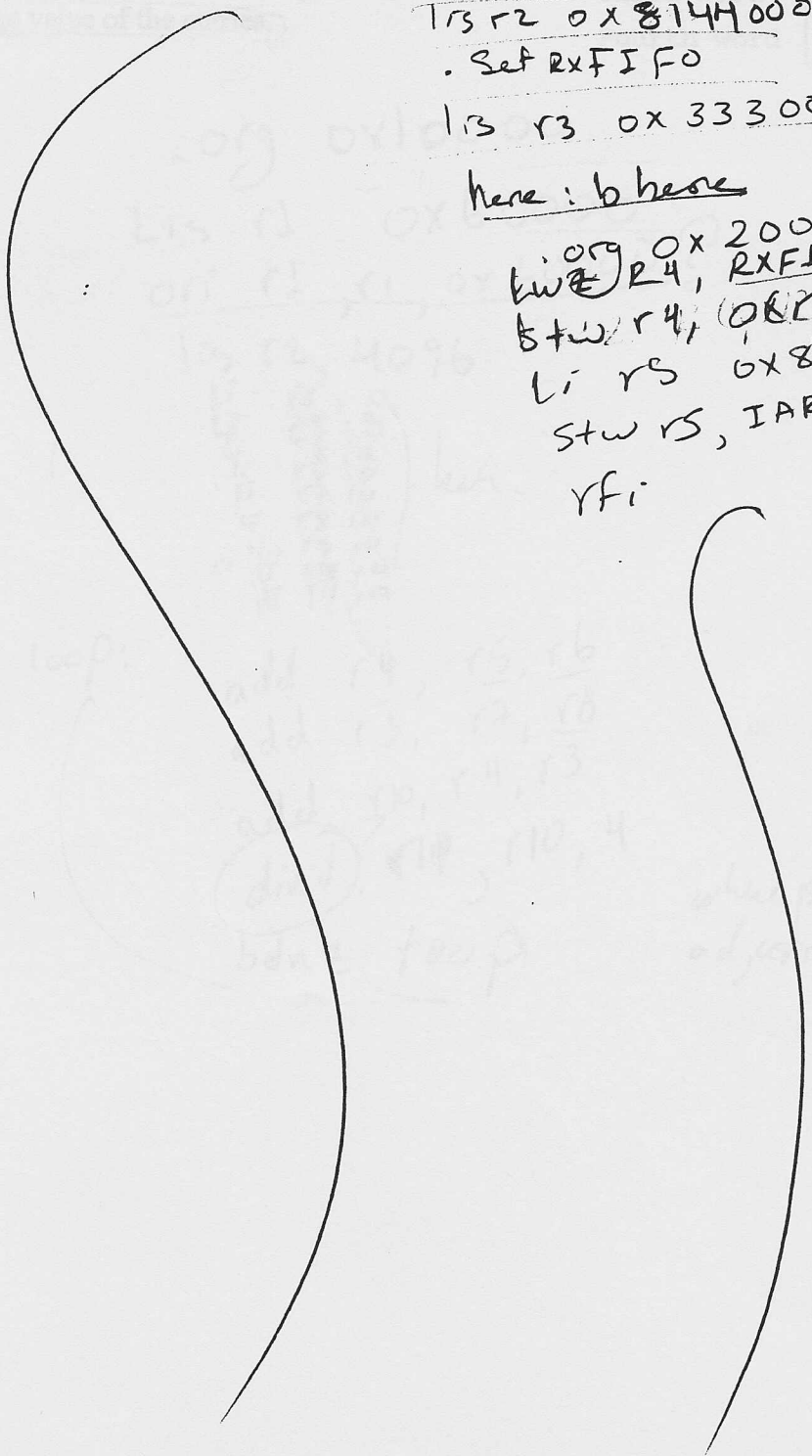
```
lis r1, 0x33333
.set IER
.set MER
lis r2, 0x81480000
.SET CTR
lis r3, 0x81440000
lis r4, 0x81450000
.set IPR
.set ISR
.set CIE
lis r8, 0x2222
mtspr 987, r8
lwz r9, 0x10 (r3)
stw r9, CTR(r1)
lis r9, 0x8
stw r9, IER(r1)
stw r9, IAR(r1)
lis r9, 0x0
stw r9, 4(r3)
stw r9, 4(r4)
wrteei 1
```

A

not done

6. Interrupt question #2. This question deals with the steady state activities associated with interrupts set up in Question 5. On this page, give the code for Interrupt Service Routine activities for the interrupts in question. The UART should echo the character. The timer interrupt should increment (by one) the value stored at 0x33300. And the PIT activity should complement the value established in the initialization and send it out to address 0x81440000.

```
lis r1  0x33333
lis r2 0x81440000
. Set RxFIFO
lis r3 0x33300
here : b here
.org 0x200000000
lwz r4, RXFIFO
stw r4, 0(r3)
li r5  0x81430000
stw r5, IAR(r3)
rfi
```



7. A user has collected data over an extended period. This data has been organized in a data structure that is an array of records. Each record is organized as shown at right: the value (in the first word slot), followed by the time (two 32 bit words from time base) and a sensor number (another 32 bit word). The user has collected 4096 of these records, and they are stored in consecutive memory locations starting at address 0x60000. In the space provided below, present a code fragment that will determine the average value of the entries.

First word	Value
Second word	Time (MS half)
Third word	Time (LS half)
Fourth word	Sensor Number

```
.org 0x10000
Lis r1  0x60000
: ori r1, r1, 0x60000 ?
lis r2, 4096
li  r3, 0
li  r4, 0
li  r5, 0
li  r6, 0
li  r7, 0
li  r8, 0
li  r9, 0
li  r10, 0
li  r11, 0
loop:
add r4, r5, r6
add r3, r7, r8
add r10, r4, r3
divd r10, r10, 4
bdnz loop
```

huh.

where is address adjustment?

8. Let's try this again. Let R1 be the stack pointer. Present two routines, one for PUSH and one for POP. The Push routine should take the current return address, place it into R31, and then push onto the stack registers R16 thru R31. The POP removes from the stack the 16 register values and restores the values to appropriate registers, then restores the return address. Make sure that the stack pointer is dealt with appropriately.

R16 → 31

need 68 places

PUSH

```
addi  r1, r1, -68
stw   r31, 60(r1)
stw   r30, 56(r1)
stw   r29, 52(r1)
stw   r28, 48(r1)
stw   r27, 44(r1)
stw   r26, 40(r1)
stw   r25, 36(r1)
stw   r24, 32(r1)
stw   r23, 28(r1)
stw   r22, 24(r1)
stw   r21, 20(r1)
stw   r20, 16(r1)
stw   r19, 12(r1)
stw   r18, 8(r1)
stw   r17, 4(r1)
stw   r16, 0(r1)
mflr
blr target
```

stmw!!

POP

```
lwz   r11, 0(r1)
mtlr  r11
lwz   r16, 0(r1)
lwz   r17, 4(r1)
lwz   r18, 8(r1)
lwz   r19, 12(r1)
lwz   r20, 16(r1)
lwz   r21, 20(r1)
lwz   r22, 24(r1)
lwz   r23, 28(r1)
lwz   r24, 32(r1)
lwz   r25, 34(r1)
lwz   r26, 36(r1)
lwz   r17, 40(r1)
lwz   r28, 44(r1)
lwz   r29, 48(r1)
lwz   r30, 52(r1)
lwz   r31, 56(r1)
addi  r1, r1, 68
blr
```

lmw!!

Table 4: **XPS UART Lite Registers**

Base Address + Offset (hex)	Register Name	Access Type	Default Value (hex)	Description
C_BASEADDR + 0x0	Rx FIFO	Read	0x0	Receive Data FIFO
C_BASEADDR + 0x4	Tx FIFO	Write	0x0	Transmit Data FIFO
C_BASEADDR + 0x8	STAT_REG	Read	0x4	UART Lite Status Register
C_BASEADDR + 0xC	CTRL_REG	Write	0x0	UART Lite Control Register

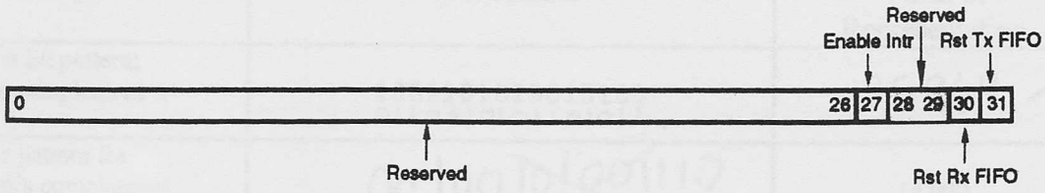


Figure 4: **UART Lite Control Register**

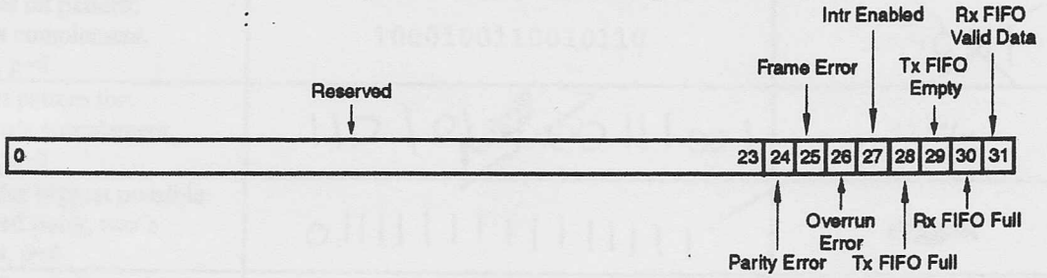


Figure 5: **UART Lite Status Register**

The addresses of the XPS Timer/ Counter registers are shown in the Table 4:

Table 4: **XPS Timer/Counter Register Address Map**

Register	Address (Hex)	Size	Type	Description
TCSR0	C_BASEADDR + 0x00	Word	Read/Write	Control/Status Register 0
TLR0	C_BASEADDR + 0x04	Word	Read/Write	Load Register 0
TCR0	C_BASEADDR + 0x08	Word	Read	Timer/Counter Register 0

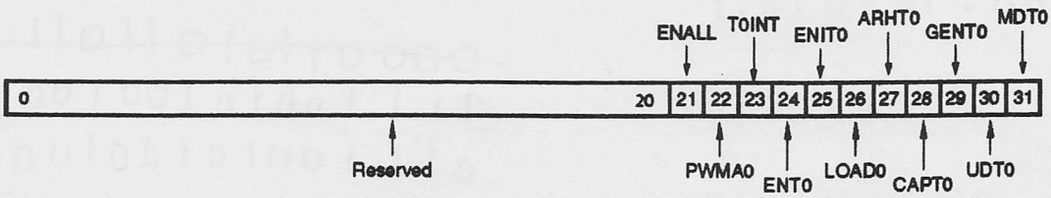


Figure 6: **Timer Control/Status Register 0 (TCSR0)**

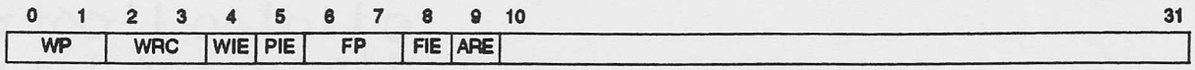


Figure 8-4: **Timer-Control Register (TCR)**

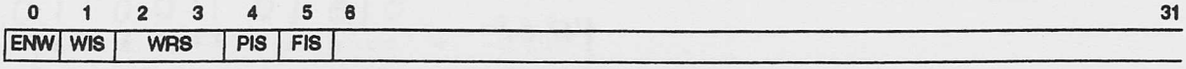


Figure 8-5: **Timer-Status Register (TSR)**

ECE-3 Final Exam Solutions

Two's Complement Pattern	1001100 1001000	-103.34375
Unsigned Binary Pattern	1001100 1001000	152.5625
Two's Complement Pattern	101110 000000	24,000
Bit Two's Complement	1111100 1010100	-7.34375
Bit Two's Complement	11010010 111000	-45 17/32
Unsigned Binary	10101010 0101010	170.664063

1.

2.

a) fundamental principle on which stored program computers are based:

Fetch, decode, execute

b) how is Memory Mapped I/O implemented?

I/O activity is carried out by writing and reading locations in memory.

I/O elements are assigned addresses in the memory space of the processor

c) what instructions move multiple register values to and from memory?

lmw, stmw

d) vector table starts at 0x00100000. ISR is created to handle privilege violations,

where should the ISR be located?

0x00100700

e) give a two instruction sequence to look for the third legal location of interrupt table

lis r3, 0x0002

mtevpr r3

f) why does PowerPC have both a user mode and a privilege mode?

for protection and to manage the resources for the operating system

g) how does the system enter user mode?

Setting PR bit to 1, in the MSR

li r2 0x00010000 mtspr msr r2

h) how far can a branch-to-subroutine go? Assume instruction is at 0x00010000.

0x07ffffc + 0x00010000 = 0x0100fffc

3. registers changed:

r1 = 0x11111118

r2 = 0x00000008

r7 = 0xFFFFFFFF

r9 = 0xBBBBBBBB

r10 = 0x12345678

LR = 0x3008

4. registers changed:

r0 = 0xDDDDDDDD

r3 = 0x54321065

r4 = 0xFFFFFFFF

r13 = 0xFFCCCCCC

r14 = 0x40000400

r15 = 0x0000EFFE

memory changed:

address: 0x4000042c → 99 99 99 99

address: 0x40000432 → 88

5. Interrupt Question #1: Provide code to configure the ML403 as we had in the lab, but fixed so that the only active interrupt is from the UART.

```
lis r1, 0x83e0

lis r2, 0x80

stw r2, 0x100C(r1) #LCR

li r3, 0x45

stw r3, 0x1000(r1) #DLL

li r3, 0x01

stw r3, 0x1004(r1) #DLM

li r3, 0x7

stw r3, 0x100C(r1)

li r3, 0x8

stw r3, 0x1004(r1)

lis r4, 0x8140

li r0, 0

stw r0, 4(r4)

lis r6, 0x8141

stw r0, 4(r6)

lis r7, 0x8142

stw r0, 4(r7)

lis r8, 0x8143

stw r0, 0(r8)

lis r10, 0x8180 #INTC

li r1, 0x3

stw r1, 0xC(r10) #MER

li r1, 0x8

stw r1, 0xC(r10) #IER

li r9, 0xF
```

```
stw r9, 0(r10) #ISR

lis r22, 0020

mtevpr r22

wrteei 1

b here
```

6. Interrupt Question #2: deals with steady state activities from Question 5; give code for the ISR for the UART

```
lis r6, 0x8180 #INTC

lis r5, 0x83e0 #UART

.org 0x500

lwz r1, 0x1000(r5) #load from RBR

lis r2, 0xffff

ori r2, r2, 0x0100

stw r1, 0(r2)

li r3, 0x60

stw r3, 0x1014 #LSR

li r3, 0x8

stw r3, 0xC(r6) #IAR

rfi
```

7. Provide two sets of instructions, one for the "push" operation and one for the "pop" operation

Push:

```
.org 0xhe11  (for Mike)

lis r1, 0x2000

stw r3, 0x10(r1)

stw r8, c(r1)

stw r14, 8(r1)
```

stw r22, 4(r1)

stw r25, 0(r1)

Pop:

.org 0x0200

lwz r25, 0(r1)

addi r1, r1, 4

lwz r22, 4(r1)

addi r1, r1, 4

lwz r14, 8(r1)

addi r1, r1, 4

lwz r8, c(r1)

addi r1, r1, 4

lwz r3, 0x10(r1)

8.

a) What is the instruction represented by the bit pattern 0x38857342?

addi r4, r5, 0x7342

b) what is the instruction represented by 0x4800 1511 located at 00004000?

bl 0x1510

0x1510 away from 0x00004000

c) give the coding for the instruction xor. r5, r4, r5

0x7C832A79