CCIT4026 Introduction to Computer Organization (Assignment #3) Solutions

Q1.

In a Von Neumann architecture, groups of bits have no intrinsic meanings by themselves. What a bit pattern represents depends entirely on how it is used. The following table shows bit patterns expressed in hexadecimal notation.

i.	0x2210 0004
ii.	0xAD110000

a. What decimal number does the bit pattern represent if,
   i. It is a two's-complement integer?
   ii. It is an unsigned integer?

b. If this bit pattern is placed into the Instruction Register, what MIPS instruction will be executed?

c. What decimal number does each bit pattern represent if it is a floating-point number in IEEE 754 standard?

d. The following table shows decimal numbers.

i.	2821.25
ii.	-998.125

Write down the binary and hexadecimal representation of the decimal number, assuming the IEEE 754 single precision format.

Solutions:

(a)

0x22100004 = $(0010\ 0010\ 0001\ 0000\ 0000\ 0000\ 000\ 0100)_2$
0xAD110000 = $(1010\ 1101\ 0001\ 0001\ 0000\ 0000\ 0000\ 0000)_2$

	2's complement integer	***Unsigned integer***
i. 0x22100004	571,473,924	571,473,924
ii. 0xAD110000	-1,391,394,816	2,903,572,480

For (a), as it is a positive number, it is the same positive value whether it be in an unsigned representation of 2's complement representation
(note that the question is NOT asking you to "convert")

(b)

i.	0x2210 0004 $(0010\ 0010\ 0001\ 0000\ 0000\ 0000\ 000\ 0100)_2$ $(001000\ 10000\ 10000\ \ 0000000000000100)_2$ Opcode for $001000_2 = 8_d$ is addi, it is an I-type instruction Thus second field is register rs, register 16, $s0 Third field is register rt, register 16, $s0 Fourth field is immediate data: 4 The instruction should be "addi $s0, $s0, 4"
ii.	0xAD110000 $(1010\ 1101\ 0001\ 0001\ 0000\ 0000\ 0000\ 0000)_2$ $(101011\ 01000\ 10001\ 0000\ 0000\ 0000\ 0000)_2$ Opcode for 101011_2 is sw, it is an I-type instruction Thus second field is base register 8, $t0 Third field is register rt, register 17, $s1 Fourth field is 16-bit offset: 0 The instruction should be "sw $s1, 0($t0)"

(c)

i.	0 01000100 00100000000000000000100 sign (0) is positive Biasd exp is 68 => Exp is 68-127 = -59 There is a hidden 1 mantissa = 1.XXXX Significand = $1.0 + 2^{-3} + 2^{-21} = 1.125000477$ Answer: $+1.125000477 \times 2^{-59}$ $= 1.95156473765 \times 10^{-18}$
ii.	1 01011010 00100010000000000000000 sign is negative Biasd Exp=0x5A=90 =>Exp = 90–127 = –37 there is a hidden 1 Significand = $1.0 + 2^{-3} + 2^{-7} = 1.1328125$ answer = $-8.24229573482 \times 10^{-12}$

(d)

(i)

2821.25 = 1011 0000 0101.01

= 1.011 0000 010101x2^{11} (normalized)

Sign bit (1 bit) = 0

Exponent (8 bits) = 11 = 138 – 127

Based exponent = 138 = 10001010

Significand (23 bits) =11000111011000000000000 (zero-padding at the right-hand side)

The result is 0 10001010 01100000101010000000000

In HEX, 0x45305400

(ii)

-998.3125 =-$(1111100110.001)_2$

=-$(1.111100110001)_2$x 2^9

Sign bit = 1

Exponent = 9 = 136 – 127 => Bias Exp = 136 = 10001000_2

Significand = 1111 0011 0001 0000 0000 000_2

The result is 1 10001000 1111 0011 0001 0000 0000 000

In HEX, 0xC4798800

Q2.

Consider two different implementations, P1 and P2, of the same instruction set. There are five classes of instruction (A, B, C, D and E) in the instruction set.

P1 has a clock rate of 1 GHz. P2 has a clock rate of 2 GHz. The average number of cycles for each instruction class for P1 and P2 is as follows:

Class	CPI on P1	CPI on P2
A	1	2
B	2	3
C	3	3
D	3	4
E	4	4

(a) Assume that peak performance is defined as the fastest rate that a computer can execute any instruction sequence. What are the peak performances of P1 and P2 expressed in instructions per second?

(b) If the number of instructions executed in a certain program is divided equally among the classes of instructions except for class B, which occurs **3** times as often as each of the others, how much faster is P2 than P1?

Solutions:

(a)

The ideal instruction sequence for P1 is one composed entirely of instructions from class A (which has CPI of 1).
So P1's peak performance is (2e9 cycles/second) / (1 cycle/instruction) = 2000 MIPS.
Similarly, the ideal sequence for P2 contains only instructions from A and B (which all have a CPI of 2).
So P2's peak performance is (3e9 cycles/second) / (2 cycles/instruction) = 1500 MIPS.

(b)

The average CPI of P1 is (1 + 2*3 + 3 + 3 + 4)/7 = 17/7.
The average CPI of P2 is (2 + 3*3 + 3 + 4 + 4)/7 = 22/7.
P2 then is ((3e9 cycles/second) / (22/7 cycles/instruction)) / ((2e9 cycles/second) / (17/7 cycles/instruction)) = 51/44 = 1.159 times faster than P1.

Q3.

(a) A benchmark program takes 80 seconds to finish on a machine, with floating point arithmetic representing 70% of the time. The hardware engineering team plans to optimize floating point arithmetic of the machine by a factor of 4. What is the expected overall improvement of speed?
(clearly show your steps)

(b) It turns out that the hardware team is only able to optimize the floating-point arithmetic by a factor of 2.4. To remedy the shortfall, the software team now wants to adjust the proportion of floating-point arithmetic in the program so as to still achieve the performance improvement claimed in (a) above. What is the required final proportion of floating-point arithmetic in the benchmark program?
(Clearly show your steps)

Solutions:

(a)

80/n = (0.7x80/4) + (80(1-0.7)),

$$\frac{1}{n} = \frac{0.7}{4} + 0.3 = 0.475$$

Overall speed improvement, n = 2.105

(b)

Still want to achieve the same performance boost so left hand side is 80*0.475
X is the proportion of floating point-related portion

80*0.475 = (x/2.4) + (80-x)
38 = x/2.4 + 80-x
x = 72
Percentage of final proportion = 72/80 = 90%

MIPS/SPIM Reference Card

CORE INSTRUCTION SET (INCLUDING PSEUDO INSTRUCTIONS)

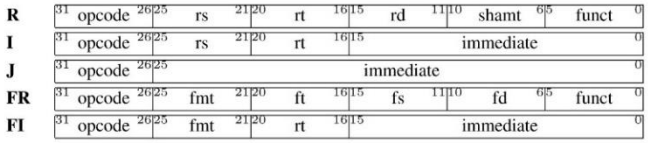
NAME	MNE-MON-IC	FOR-MAT	OPERATION (in Verilog)		OPCODE/ FUNCT (Hex)
Add	add	R	R[rd]=R[rs]+R[rt]	(1)	0/20
Add Immediate	addi	I	R[rt]=R[rs]+SignExtImm	(1)(2)	8
Add Imm. Unsigned	addiu	I	R[rt]=R[rs]+SignExtImm	(2)	9
Add Unsigned	addu	R	R[rd]=R[rs]+R[rt]	(2)	0/21
Subtract	sub	R	R[rd]=R[rs]-R[rt]	(1)	0/22
Subtract Unsigned	subu	R	R[rd]=R[rs]-R[rt]		0/23
And	and	R	R[rd]=R[rs]&R[rt]		0/24
And Immediate	andi	I	R[rt]=R[rs]&ZeroExtImm	(3)	c
Nor	nor	R	R[rd]=~(R[rs]|R[rt])		0/27
Or	or	R	R[rd]=R[rs]|R[rt]		0/25
Or Immediate	ori	I	R[rt]=R[rs]|ZeroExtImm	(3)	d
Xor	xor	R	R[rd]=R[rs]^R[rt]		0/26
Xor Immediate	xori	I	R[rt]=R[rs]^ZeroExtImm		e
Shift Left Logical	sll	R	R[rd]=R[rt]≪shamt		0/00
Shift Right Logical	srl	R	R[rd]=R[rt]≫shamt		0/02
Shift Right Arithmetic	sra	R	R[rd]=R[rt]≫>shamt		0/03
Shift Left Logical Var.	sllv	R	R[rd]=R[rs]≪R[rt]		0/04
Shift Right Logical Var.	srlv	R	R[rd]=R[rs]≫R[rt]		0/06
Shift Right Arithmetic Var.	srav	R	R[rd]=R[rs]≫>R[rt]		0/07
Set Less Than	slt	R	R[rd]=(R[rs]<R[rt])?1:0		0/2a
Set Less Than Imm.	slti	I	R[rt]=(R[rs]<SignExtImm)?1:0	(2)	a
Set Less Than Imm. Unsign.	sltiu	I	R[rt]=(R[rs]<SignExtImm)?1:0	(2)(6)	b
Set Less Than Unsigned	sltu	R	R[rd]=(R[rs]<R[rt])?1:0	(6)	0/2b
Branch On Equal	beq	I	if(R[rs]==R[rt]) PC=PC+4+BranchAddr	(4)	4
Branch On Not Equal	bne	I	if(R[rs]!=R[rt]) PC=PC+4+BranchAddr	(4)	5
Branch Less Than	blt	P	if(R[rs]<R[rt]) PC=PC+4+BranchAddr		
Branch Greater Than	bgt	P	if(R[rs]>R[rt]) PC=PC+4+BranchAddr		
Branch Less Than Or Equal	ble	P	if(R[rs]<=R[rt]) PC=PC+4+BranchAddr		
Branch Greater Than Or Equal	bge	P	if(R[rs]>=R[rt]) PC=PC+4+BranchAddr		
Jump	j	J	PC=JumpAddr	(5)	2
Jump And Link	jal	J	R[31]=PC+4; PC=JumpAddr	(5)	2
Jump Register	jr	R	PC=R[rs]		0/08
Jump And Link Register	jalr	R	R[31]=PC+4; PC=R[rs]		0/09
Move	move	P	R[rd]=R[rs]		
Load Byte	lb	I	R[rt]={24'b0, M[R[rs]+ZeroExtImm](7:0)}	(3)	20
Load Byte Unsigned	lbu	I	R[rt]={24'b0, M[R[rs]+SignExtImm](7:0)}	(2)	24
Load Halfword	lh	I	R[rt]={16'b0, M[R[rs]+ZeroExtImm](15:0)}	(3)	25
Load Halfword Unsigned	lhu	I	R[rt]={16'b0, M[R[rs]+SignExtImm](15:0)}	(2)	25
Load Upper Imm.	lui	I	R[rt]={imm,16'b0}		f
Load Word	lw	I	R[rt]=M[R[rs]+SignExtImm]	(2)	23
Load Immediate	li	P	R[rd]=immediate		
Load Address	la	P	R[rd]=immediate		
Store Byte	sb	I	M[R[rs]+SignExtImm] (7:0)=R[rt](7:0)	(2)	28
Store Halfword	sh	I	M[R[rs]+SignExtImm] (15:0)=R[rt](15:0)	(2)	29
Store Word	sw	I	M[R[rs]+SignExtImm]=R[rt]	(2)	2b

REGISTERS

NAME	NMBR	USE	STORE?
$zero	0	The Constant Value 0	N.A.
$at	1	Assembler Temporary	No
$v0-$v1	2-3	Values for Function Results and Expression Evaluation	No
$a0-$a3	4-7	Arguments	No
$t0-$t7	8-15	Temporaries	No
$s0-$s7	16-23	Saved Temporaries	Yes
$t8-$t9	24-25	Temporaries	No
$k0-$k1	26-27	Reserved for OS Kernel	No
$gp	28	Global Pointer	Yes
$sp	29	Stack Pointer	Yes
$fp	30	Frame Pointer	Yes
$ra	31	Return Address	Yes
$f0-$f31	0-31	Floating Point Registers	Yes

(1) May cause overflow exception
(2) SignExtImm = {16{immediate[15]},immediate }
(3) ZeroExtImm = {16{1b'0},immediate }
(4) BranchAddr = {14{immediate[15]},immediate,2'b0 }
(4) JumpAddr = {PC[31:28], address, 2'b0 }
(6) Operands considered unsigned numbers (vs. 2 s comp.)

BASIC INSTRUCTION FORMATS, FLOATING POINT INSTRUCTION FORMATS



ARITHMETIC CORE INSTRUCTION SET

NAME	MNE-MON-IC	FOR-MAT	OPERATION (in Verilog)		OPCODE/ FMT/FT/ FUNCT
Divide	`div`	R	Lo=R[rs]/R[rt]; Hi=R[rs]%R[rt]		0/–/–/1a
Divide Unsigned	`divu`	R	Lo=R[rs]/R[rt]; Hi=R[rs]%R[rt]	(6)	0/–/–/1b
Multiply	`mult`	R	{Hi,Lo}=R[rs]∗R[rt]		0/–/–/18
Multiply Unsigned	`multu`	R	{Hi,Lo}=R[rs]∗R[rt]	(6)	0/–/–/19
Branch On FP True	`bc1t`	FI	if(FPCond) PC=PC+4+BranchAddr	(4)	11/8/1/–
Branch On FP False	`bc1f`	FR	if(!FPCond) PC=PC+4+BranchAddr	(4)	11/8/0/–
FP Compare Single	`c.`*x*`.s`*	FR	FPCond=(F[fs] *op* F[ft])?1:0		11/10/–/*y*
FP Compare Double	`c.`*x*`.d`*	FR	FPCond=({F[fs],F[fs+1]} *op* {F[ft],F[ft+1]})?1:0		11/11/–/*y*
			*(*x* is `eq`, `lt` or `le`) (*op* is ==, < or <=) (*y* is 32, 3c or 3e)		
FP Add Single	`add.s`	FR	F[fd]=F[fs]+F[ft]		11/10/–/0
FP Divide Single	`div.s`	FR	F[fd]=F[fs]/F[ft]		11/10/–/3
FP Multiply Single	`mul.s`	FR	F[fd]=F[fs]∗F[ft]		11/10/–/2
FP Subtract Single	`sub.s`	FR	F[fd]=F[fs]-F[ft]		11/10/–/1
FP Add Double	`add.d`	FR	{F[fd],F[fd+1]}={F[fs],F[fs+1]}+{F[ft],F[ft+1]}		11/11/–/0
FP Divide Double	`div.d`	FR	{F[fd],F[fd+1]}={F[fs],F[fs+1]}/{F[ft],F[ft+1]}		11/11/–/3
FP Multiply Double	`mul.d`	FR	{F[fd],F[fd+1]}={F[fs],F[fs+1]}∗{F[ft],F[ft+1]}		11/11/–/2
FP Subtract Double	`sub.d`	FR	{F[fd],F[fd+1]}={F[fs],F[fs+1]}-{F[ft],F[ft+1]}		11/11/–/1
Move From Hi	`mfhi`	R	R[rd]=Hi		0/–/–/10
Move From Lo	`mflo`	R	R[rd]=Lo		0/–/–/12
Move From Control	`mfc0`	R	R[rd]=CR[rs]		16/0/–/0
Load FP Single	`lwc1`	I	F[rt]=M[R[rs]+SignExtImm]	(2)	31/–/–/–
Load FP Double	`ldc1`	I	F[rt]=M[R[rs]+SignExtImm]; F[rt+1]=M[R[rs]+SignExtImm+4]	(2)	35/–/–/–
Store FP Single	`swc1`	I	M[R[rs]+SignExtImm]=F[rt]	(2)	39/–/–/–
Store FP Double	`sdc1`	I	M[R[rs]+SignExtImm]=F[rt]; M[R[rs]+SignExtImm+4]=F[rt+1]	(2)	3d/–/–/–

ASSEMBLER DIRECTIVES

`.data` [*addr*]*	Subsequent items are stored in the data segment
`.kdata` [*addr*]*	Subsequent items are stored in the kernel data segment
`.ktext` [*addr*]*	Subsequent items are stored in the kernel text segment
`.text` [*addr*]*	Subsequent items are stored in the text
	* starting at [*addr*] if specified
`.ascii` *str*	Store string *str* in memory, but do not null-terminate it
`.asciiz` *str*	Store string *str* in memory and null-terminate it
`.byte` $b_1, \ldots, b_n$	Store the n values in successive bytes of memory
`.double` $d_1, \ldots, d_n$	Store the n floating-point double precision numbers in successive memory locations
`.float` $f_1, \ldots, f_1$	Store the n floating-point single precision numbers in successive memory locations
`.half` $h_1, \ldots, h_n$	Store the n 16-bit quantities in successive memory halfwords
`.word` $w_1, \ldots, w_n$	Store the n 32-bit quantities in successive memory words
`.space` n	Allocate n bytes of space in the current segment
`.extern` *symsize*	Declare that the datum stored at *sym* is *size* bytes large and is a global label
`.globl` *sym*	Declare that label *sym* is global and can be referenced from other files
`.align` n	Align the next datum on a 2^n byte boundary, until the next `.data` or `.kdata` directive
`.set at`	Tells SPIM to complain if subsequent instructions use `$at`
`.set noat`	prevents SPIM from complaining if subsequent instructions use `$at`

SYSCALLS

SERVICE	$v0	ARGS	RESULT
`print_int`	1	integer `$a0`	
`print_float`	2	float `$f12`	
`print_double`	3	double `$f12/$f13`	
`print_string`	4	string `$a0`	
`read_int`	5		integer (in `$v0`)
`read_float`	6		float (in `$f0`)
`read_double`	7		double (in `$f0`)
`read_string`	8	buf `$a0`, buflen `$a1`	
`sbrk`	9	amount `$a`	address (in `$v0`)
`exit`	10		

EXCEPTION CODES

Number	Name	Cause of Exception
0	Int	Interrupt (hardware)
4	AdEL	Address Error Exception (load or instruction fetch)
5	AdES	Address Error Exception (store)
6	IBE	Bus Error on Instruction Fetch
7	DBE	Bus Error on Load or Store
8	Sys	Syscall Exception
9	Bp	Breakpoint Exception
10	RI	Reserved Instruction Exception
11	CpU	Coprocessor Unimplemented
12	Ov	Arithmetic Overflow Exception
13	Tr	Trap
15	FPE	Floating Point Exception

[1] Patterson, David A; Hennessy, John J.: Computer Organization and Design, 3rd Edition. Morgan Kaufmann Publishers. San Francisco, 2005.