LAB #03 Integer Arithmetic

Add/Subtract Instructions

The difference between **add/sub** and **addu/subu** instructions is that in case of overflow occurrence, the **add/sub** instructions will cause an arithmetic exception and the result will not be written to the destination register. However, for the instructions **addu/subu**, overflow occurrence is ignored.

Operation	Meaning
`add  $s1, $s2, $s3`	`$s1 = $s2 + $s3`
`addu  $s1, $s2, $s3`	`$s1 = $s2 + $s3`
`sub  $s1, $s2,`	`$s1 = $s2 – $s3`
`subu  $s1, $s2, $s3`	`$s1 = $s2 – $s3`
`addi  $s1, $s2, 10`	`$s1 = $s2 + 10`
`addiu $s1, $s2, 10`	`$s1 = $s2 + 10`

Fig. 1 Arithmetic instructions

Logical Bitwise Instructions

Operation	Meaning
`and  $s1, $s2, $s3`	`$s1 = $s2 & $s3`
`or  $s1, $s2, $s3`	`$s1 = $s2 | $s3`
`xor  $s1, $s2, $s3`	`$s1 =  $s2 ^ $s3`
`nor  $s1, $s2, $s3`	`$s1 = ~($s2|$s3)`
`andi $s1, $s2, 10`	`$s1 = $s2 & 10`
`ori  $s1, $s2, 10`	`$s1 = $s2 | 10`
`xori $s1, $s2, 10`	`$s1 = $s2 ^ 10`

Fig. 2 Logical instructions

Fig. 3 shows one example of logical operations.

The pseudo-instruction `li` instruction initialize $s1 value with 32 bits value at once, however, there is no instruction to initialize 32 bits value at once in MIPS because the constant value used in MIPS is at most 16 bits. Therefore, if we do so, the following combination of instructions are the way. (Let's say, we want to initialize $s1 with 0x12345678 value, and $s1 contains 0 value.)

```
lui $s1, 0x1234   # load the upper 16 bits immediately with 0x1234.
ori $s1, 0x5678   # fill the lower 16 bits with 0x5678.
```

Consequently, when we use `li $s1, 0x12345678`, assembler use the above instructions instead.

```
.text
.globl main
main: # main program

li $s1, 0xabcd1234      # Pseudo instruction to initialize a register
li $s2, 0xffff0000

and $s3,$s1,$s2         # $s3 = 0xabcd0000
or  $s4,$s1,$s2         # $s4 = 0xffff1234
xor $s5,$s1,$s2         # $s5 = 0x54321234
nor $s6,$s1,$s2         # $s6 = 0x0000edcb
li  $v0, 10             # Exit program
syscall
```

Fig. 3 Example of Logical operations

Shift Instructions

Operation	Meaning
`sll  $s1,$s2,10`	`$s1 = $s2 << 10`
`srl  $s1,$s2,10`	`$s1 = $s2>>>10`
`sra  $s1,$s2,10`	`$s1 = $s2 >> 10`
`sllv $s1,$s2,$s3`	`$s1 = $s2 << $s3`
`srlv $s1,$s2,$s3`	`$s1 = $s2>>>$s3`
`srav $s1,$s2,$s3`	`$s1 = $s2 >> $s3`

Fig. 4 Shift Instructions

Shifting is to move all the bits in a register left or right.
`sll/srl :` mean shift left/right logical.
`sra` means shift right arithmetic for which the sign-bit (rather than 0) is shifted from the left. (i.e., if the sign bit was 1 then 1 is repeated as many times as shamt, a.k.a. arithmetic shift).
`sll, srl, sra`: the shift amount is a 5-bit constant.
`sllv, srlv, srav`: the shift amount is variable and is stored in a register.

ASCII value	Char-acter	ASCII value	Char-acter	ASCII value	Char-acter	ASCII value	Char-acter	ASCII value	Char-acter	ASCII value	Char-acter
32	space	48	0	64	@	80	P	96	`	112	p
33	!	49	1	65	A	81	Q	97	a	113	q
34	"	50	2	66	B	82	R	98	b	114	r
35	#	51	3	67	C	83	S	99	c	115	s
36	$	52	4	68	D	84	T	100	d	116	t
37	%	53	5	69	E	85	U	101	e	117	u
38	&	54	6	70	F	86	V	102	f	118	v
39	'	55	7	71	G	87	W	103	g	119	w
40	(	56	8	72	H	88	X	104	h	120	x
41	)	57	9	73	I	89	Y	105	i	121	y
42	*	58	:	74	J	90	Z	106	j	122	z
43	+	59	;	75	K	91	[	107	k	123	{
44	,	60	<	76	L	92	\	108	l	124	|
45	-	61	=	77	M	93	]	109	m	125	}
46	.	62	>	78	N	94	^	110	n	126	~
47	/	63	?	79	O	95	_	111	o	127	DEL

Fig. 5 ASCII representation of characters (ASCII values are in decimal numbers.)

Task to do

1. Write a program to ask the user to enter two integers A and B and then display the result of computing the expression: A + 2B - 5.

2. Assume that $s1 = 0x12345678 and $s2 = 0xffff9a00. Determine the content of registers $s3 to $s6 after executing the following instructions:

```
and $s3,$s1,$s2 # $s3 =
or  $s4,$s1,$s2 # $s4 =
xor $s5,$s1,$s2 # $s5 =
nor $s6,$s1,$s2 # $s6 =
```

Write a program to execute these instructions and verify the content of registers $s3 to $s6.

3. Assume that $s1 = 0x87654321. Determine the content of registers $s2 to $s4 after executing the following instructions:

```
sll $s2,$s1, 16 # $s2 =
srl $s3,$s1, 8  # $s3 =
sra $s4,$s1, 12 # $s4 =
```

Write a program to execute these instructions and verify the content of registers $s2 to $s4.

4. Write a program that asks the user to enter an alphabetic character (either lower or upper case) and change the case of the character from lower to upper and from upper to lower and display it. (Hint: ASCII code of 'a' = 0x61 and 'A' = 0x41. So, in case of ‘a’, 0x61 – 0x20 will give us upper case letter ‘A’. For the ASCII values, you can refer to Fig. 5)

5. Write a program that asks the user to enter an unsigned number and read it. Then swap the bits at odd positions with those at even positions and display the resulting number. For example, if the user enters the number 9, which has binary representation of 1001, then bit 0 is swapped with bit 1, and bit 2 is swapped with bit 3, resulting in the binary number 0110. Thus, the program should display 6.

NOTE: Write a report according to the LAB questions and submit a word (or pdf) file.
For each question, **write** assembly code (**not** screen capture… hard to read).