ASSIGNMENT- 1

21AIE112

ELEMENTS OF COMPUTING SYSTEMS- 2

BATCH A TEAM 7

TEAM MEMBERS

I. GAJULA SRI VATSANKA	CB.EN.U4AIE21010
II. GUNNAM HIMAMSH	CB.EN.U4AIE21014
III. VIKYAT BANSAL	CB.EN.U4AIE21076
IV. M.PRASANNA TEJA	CB.EN.U4AIE21035

QUESTION-1

Write the instructions to add two numbers stored in registers t1and s3 and store the result in register s2

a. Add with overflow exception.

b. Add without overflow exception.

ANS:

a. Adding s2, t1, s3 with overflow exception

Given:

t1 and s3 are source registers which stores the input and s2 is the register where result is stored. So, the equation will be

t1+s3=s2

Instruction:

It is a R type Instruction since all the data values are located in registers

Format:

All R-type instructions have the format

[OP-rd-rs-rt-shamt-funct]

Procedure:

OP is the code for the operation here add is the operation so

OP=add

rd is destination register here s2 is the destination register so

rd=s2

rs and rt are input sources in order so

rs=t1

rt=s3

Here we know t1, s2, s3 are registers which are in assembly form.

So, we have to convert the assembly form to register and then to binary form because computers only understand binary form

t1=r9 =rs

s3=r19=rt

s2=r18=rd

OP code for R instruction is always ‘0’ and 6-bit binary whereas rd, rs, rt, shamt all are 5-bit so the funct and shamt should be 6-bit so does to make it 32-bit.

Binary form of r9 is 01001

Binary form of r19 is 10011

Binary form of r18 is 10010

OP	rd	rs	rt	shamt	funct
0	18	9	19	0	32
000000	10010	01001	10011	00000	100000

b. Adding s2, t1, s3 without overflow exception

There is no overflow exception handling at any circumstances here.

The changes:

The changes we see from the above question is here we have to do without overflow.

So, if the overflow is not there will be changes in funct value and the operation.

The operation adds changes itself to addu.

In the operation addu rd,rs,rt,OP,shamt values will not change.

The only change is in the funct value, it changes itself to 33 from 32, so the table looks like.

OP	rd	rs	rt	shamt	funct
0	18	9	19	0	33
000000	10010	01001	10011	00000	100001

QUESTION-2

Write the binary representation of the instruction

a. addi $t2, $s3, 4.

b. add $t0, $s1, $s2.

ANS)

a) addi $t2, $s3, 4.

STEP-1 :- Identify the type of instruction.

- I instruction
- **FORMAT:-**[6 bits][5 bits][5 bits][16 bits]

  [OPcode][rs][rt][IMM]

  OP,rs,rt,IMM

STEP-2: - Decoding the given code.

- **OP** = addi [binary code - 00100]
- **rs** = $t2
  - $t2 = r19 [binary code - 10011]
- **rt** = $s3
  - $s3 = r10 [binary code - 01010]
- **IMM** = 4 [binary code - 0000000000000100]

BINARY FORMAT =

[00100 10011 01010 0000000000000100]

b)add $t0, $s1, $s2.

STEP-1: - Identify the type of instruction.

R instruction

Format:- [6 bits][5 bits][5 bits][5 bits][5 bits][6 bits]

[opcode][rs][rt][rd][shamt][funct]

[OP, rd, rs, rt]

STEP-2: - Decoding the given code

- **OP** = 0 [binary code - 000000]
- **rs** = $s1
  - $s1 = r17 [binary code - 10001]
- **rt** = $s2
  - $s2 = r18 [binary code - 10010]
- **rd** = $t0
  - $t0 = r8 [binary code - 01000]
- **shamt** = 0 [binary code - 00000]
- **funct** = add [binary code - 100000]

* **BINARY FORMAT =**

[000000 10001 10010 01000 00000 100000]

QUESTION-3

What are the various jump instructions in MIPS instruction set?

Definition:

The Jump instruction is used to perform jump operations. The format for representing the jump instruction is OP LABEL. OP code specifies which type of jump it is, and LABEL tells the address.

There are 4 types of jump instructions:

i)j

ii)jal

iii)jr

iv)jalr

J(jump): This instruction just tells us where to jump but it will not tell the return address.

JAL (jump and link): This instruction tells us both where to jump and gives us the return address.

Jr (jump register): This jump instruction causes the PC to jump to the given register.

Jalr (jump and link register): This is the combination of JAL and JR. It transfers the control to the address of the register.

S.No	Type	OP	LABEL (INTERMEDIATE)	Ex
1	j	2(000010)	0000000000001101	j13
2	jal	3(000011)	0000000000001101	jal13
3	jr	8(001000)	0000000000001101	jr13
4	jalr	9(001001)	0000000000001101	jalr13

QUESTION-4

What are the operations performed by the following set of R instructions and write their function codes?

a.) mtlo [Move to LO**]**

- **Description:** Copy the value of register $rs into
  - register $lo.

Q. What is LO?
Ans. These are *special registers* used to store the result of multiplication and division. They are separate from the $0…$31 general purpose registers, not directly addressable. Their contents are accessed with special instructions mfhi and mflo (Move From HI/LO).

- **Operation:** $lo = $rs
- **Function Code:** 010011

b.) mult [MULTIPLY**]**

- **Description:**
  Multiply the contents in register $rs by the contents in $rt, store the upper 4 bytes in $hi, and the lower 4 bytes in $lo

- **Operation:** {$hi, $lo} = ($rs × $rt)
- Function Code: 011000

c.) srav [Shift Word Right Arithmetic Variable]

- **Description:**

  Right shifts the contents of register $rt by the amount stored in the lower 5 bit of register $rs, padding with a copy of $rt [31] (value at 31th bit) i.e., a sign preserving shift meaning if number is positive then zero will come and if number is negative then one will come to the shifted place. The result is stored in $rd.

- Operation: $rd = $rt >>> $rs [5:0]
- Function Code: 000111

d.) srlv [Shift Word Right Logical Variable]

- **Description:**
  Right shifts the contents of register $rt by the amount stored in the lower 5 bit of register $rs, padding with zeros. The result is stored in $rd.
- **Operation:** $rd = $rt >> $rs [5:0]
- **Function Code:** 000110

e.) subu [Subtract Unsigned]

- **Description:** Subtract the value contained in register $rt from the value contained in register $rs and place the result in $rd. No integer overflow exception occurs under any circumstances.

- **Operation:** $rd = $rs − $rt
- **Function Code:** 100011

f.) mfhi [Move from HI]

- **Description:** Copy value of register $hi into register $rd

- **Operation:** $rd = $hi
- **Function Code:** 010000

g.) jalr [Jump and link register]

- **Description:**
- The jump-and-link-register instruction (JALR) is the union of JAL and JR, meaning that it transfers control to the address in a specified register($rs), and stores the return address in the register file($ra). However, unlike JAL, JALR allows the programmer to specify the destination register of the return address.

- **Operation:** $ra = PC + 4; PC = $rs
- **Function Code:** 001001

h.) sra [Shift word right arithmetic]

- Description: Right shifts the contents of register $rt by shamt, padding with a copy of $rt [31] i.e., a sign-preserving shift. The result is stored in $rd.

- **Operation:** $rd = $rt >>> shamt
- **Function Code:** 000011

i.) **Syscall [**System Call**]**

- **Description:**
  A system call exception occurs, immediately and unconditionally transferring control to the exception handler. The code field is available for use as software parameters, but is retrieved by the exception handler only by loading the contents of the memory word containing the instruction.

- **Operation:** SignalException(SystemCall)
- **Function Code:** 001100

j.) **sltu [**Set on less than unsigned**]**

- **Description:**
  Compare the contents of $rs and $rt as unsigned integers and record the Boolean result of the comparison in $rd. If $rs is less than $rt the result is 1 (true), otherwise 0 (false). The result is place in $rd. The arithmetic comparison does not cause an Integer Overflow exception.

- **Operation:** $rs < $rt, $rd = 1: $rd = 0
- **Function Code:** 101011

QUESTION-5

What are the operations performed by the following set of I instructions and write their function codes.

a. bgez b. bne c. bltz

d. slti e. sh f. sw

g. swc1 h. lhu I. lui

Ans)

a) Bgez

- **FULL FORM: -** Branch on Greater Than or Equal to Zero.
- **DESCRIPTION: -**
  - Branches if the register is greater than or equal to zero. the rt field is used as an extension of the opcode field.
- **Operation: -** (rs ≥ 0)
- **PURPOSE: -**To test a GPR then do a PC-relative conditional branch.
- **FORMAT: -** BGEZ rs, offset
- **FUNCTION CODE: -** 000001 [rt = 00001]

b) bne

- **FULL FORM: -**Branch on Not Equal
- **DESCRIPTION: -**
  - An 18-bit signed offset (the 16-bit offset field shifted left 2 bits) is added to the address of the instruction following the branch (not the branch itself), in the branch delay slot, to form a PC-relative effective target address.
- **OPERATION: -** (rs ≠ rt)
- **PURPOSE: -**To compare GPRs then do a PC-relative conditional branch.
- **FORMAT: -** BNE rs, rt, offset
- **FUNCTION CODE: -**000101

c) bltz

- **FULL FORM: -** Branch on Less Than Zero**.**
- **DESCRIPTION: -**
  - If the contents of GPR rs are less than zero (sign bit is 1), branch to the effective target address after the instruction in the delay slot is executed.
- **OPERATION: -** (rs < 0).
- **PURPOSE: -**To test a GPR then do a PC-relative conditional branching.
- **FORMAT: -** BLTZ rs, offset.
- **FUNCTION CODE: -** 000001 [rt = 00000]

d) slti

- **FULL FORM: -** Set on Less Than Immediate.
- **DESCRIPTION: -**
  - Compare the contents of GPR rs and the 16-bit signed immediate as signed integers and record the Boolean result of the comparison in GPR rt. If GPR rs is less than immediate the result is 1 (true), otherwise 0 (false).
  - The arithmetic comparison does not cause an Integer Overflow exception.
- **OPERATION: -** rt ← (rs < immediate).
- **PURPOSE:** -To record the result of a less-than comparison with a constant.
- **Format: -** SLTI rt, rs, immediate.
- **FUNCTION CODE: -** 001010

e) Sh

- **FULL FORM: -** Store Halfword.

- **DESCRIPTION: -**
  - The least-significant 16-bit halfword of register rt is stored in memory at the location specified by the aligned effective address.

  - The 16-bit signed offset is added to the contents of GPR base to form the effective address.

- **OPERATION: -** memory[base+offset] ← rt.

- **PURPOSE:** -To store a halfword to memory.

- **FUNCTION CODE: -** 101001

f) Sw

- **FULL FORM: -** Store Word.

- **DESCRIPTION: -**
  - The least-significant 32-bit word of register rt is stored in memory at the location specified by the aligned effective address.
  - The 16-bit signed offset is added to the contents of GPR base to form the effective address.

- **OPERATION: -** memory[base+offset] ← rt.

- **PURPOSE: -**
  To store a word to memory.

- **FUNCTION CODE: -** 101011

g) swc1

- **FULL FORM: -** Store Word from Floating-Point
- **DESCRIPTION: -**
  - The low 32-bit word from FPR ft is stored in memory at the location specified by the aligned effective address.
  - The 16-bit signed offset is added to the contents of GPR base to form the effective address.
- **OPERATION: -** memory[base+offset] ← ft.
- **PURPOSE: -** To store a word from an FPR to memory.
- **FUNCTION CODE: -** 111001

h) Lhu

- **FULL FORM: -** Load Halfword Unsigned.

- **DESCRIPTION: -**
  - The contents of the 16-bit halfword at the memory location specified by the aligned effective address are fetched, zero-extended, and placed in GPR rt.

  - The 16-bit signed offset is added to the contents of GPR base to form the effective address.

- **OPERATION: -** rt ← memory[base+offset].

- **PURPOSE: -**To load a halfword from memory as an unsigned value.

- **FUNCTION CODE: -** 100101

i) lui

- **FULL FORM: -**Load Upper Immediate.

- **DESCRIPTION: -**
  - The 16-bit immediate is shifted left 16 bits and concatenated with 16 bits of low-order zeros.

  - The 32-bit result is sign-extended and placed into GPR rt.

- **OPERATION: -** rt ← immediate || 016.

- **PURPOSE: -**To load a constant into the upper half of a word.

- **FUNCTION CODE: -** 001111

THANK YOU