Instructions:

- Please maintain academic integrity.
- State any assumptions you make.
- Show your work to get credits.
- Do not waste much time on one specific question, they might be of varying difficulty/ease.

- Questions marked ☺ are student contributed !

Question:	1	2	3	4	5	6	7	8	Total
Marks:	10	8	10	6	12	10	4	10	70
Score:									

1. [10 marks] The subparts are independent questions.

   (a) [4 marks] Give four differences between MiniMIPS and 8051 in terms of their architecture or ISA.

   Solution:

   - Instructions in 8051 are variable size (bytes)
   - Instructions in 8051 have variable number of cycles
   - Registers/memory locations are 8 or 16 bit addressable as opposed to 32-bit memory addresses in MiniMIPS
   - No pipelining in 8051
   - 8051 supports indirect addressing
   - 8051 operands for arithmetic instructions can be in registers, memory, or inline. Whereas MIPS operands for arithmetic instructions can only be in registers
   - The STACK in 8051 grows upwards, i.e., as you PUSH content to stack, the memory addresses increment.
   - Several other minor differences were also allowed.

   Marking scheme: Atleast 4 correct differences. Incorrect ones were not given marks.

   (b) [2 marks] In 8051, what are read-modify-write operations? For a read-modify-write instruction whose destination is a port, do we use data from the port pins or from the port latch (to compute the value to be written to the port) ?

   Solution: Read modify write instruction read the value of an operand, perform some operation on it and write it back to the same operand.

   If the destination of a read-modify-write operation is a port, the latch and not the pin values are read.

   Marking scheme: 1 mark each for each point above.

   (c) [2 marks] ☺ A sensor stores its 16-bit output value in a byte addressable memory device, where each memory location stores a byte. The sensor functions normally and stores in binary a decimal value of 256. An user using this sensor instead interprets the sensed value as 1. What could be the reason for this misinterpretation ? Justify.

   Solution: This could happen because of the difference in endianness used by the processor (sensor). A decimal value of 256 is 0x0100, which when stored in big-endian will be 01 in a lower address and 00 in a higher address. If the user uses little endian, he will interpret it as 00 and 01 or 0x0001 which is 1.

   Marking scheme: 2 marks given if endianness mentioned. Otherwise no marks.

(d) [2 marks] ☺ Consider the code shown below. If the Rx and Tx pins of the 8051 are not connected to any device, what will be the value stored in location 70H ? Justify.

```
ORG 0H
LJMP MAIN

ORG 100H
MAIN: CALL SERIAL
HERE: SJMP HERE

ORG 130H
SERIAL: MOV P3, #0FFH
        SETB EA
        SETB ES
        SETB SM1
        SETB REN
        MOV TMOD, #20H
        MOV TH1, #0FDH
        MOV TL1. #0FDH
        SETB TR1
        MOV SBUF, #0AAH
        MOV 70H, SBUF
        RET
```

Solution: Since the Rx and Tx pins are not connected, and there are two separate SBUFs, the value in 70H will be 0. The value written to SBUF in the earlier instruction will not be seen by the 2nd SBUF. - If they mention that earlier content in SBUF will be stored in 70H, that can also be accepted. It cannot be 0AAH.

Marking scheme:

- +2 marks for having '00' and justification.
- +1 mark for garbage value or prior value
- -1 for missing justification
- 0 for other reasons.

2. [8 marks] Consider the MIPS code below to answer the questions.

   Hint: Think like a computer, in machine language !

```
TRICKY: lui   $t0, [TRICKY]_31:16    ## NOTE: [X]_n2:n1 means bits

        ori   $t0, $t0, [TRICKY]_15:0     ##        n2 to n1 of X
        lw    $t1, 20($t0)
        addiu $t1, $t1, 1
        sw    $t1, 20($t0)
        addiu $v0, $0, 0             ## $v0 will hold result of function
        jr    $ra
```

   (a) [6 marks] What specific function does this MIPS code achieve (say, if it is repeatedly called N times) ? Justify.

   Solution: The call to this function will return the number of times it was called.

   - One should recognize that the `lw` instruction is actually loading the binary value corresponding to the instruction `addiu` $v0$,`0, 0` in `t1`
   - When the instruction `addiu` $t1$,`t1, 1` is executed, it is actually generating `addiu,` $v0$,`0, 1`.
   - The instruction `sw` $t1, 20$(`t0`) is replacing `addiu` $v0$,`0, 0` with `addiu,` $v0$,`0, 1`, essentially incrementing $v0$.
   - Hence, $v0$ gets incremented every time the function is called.

   Marking scheme:

   - 6 marks for having the above solution.
   - 5 marks for saying there's summation over N.
   - 3 marks for just saying has N in memory location.

   (b) [2 marks] What is the most number of times the above function `TRICKY` can be called so that it does what you described in (a). Justify.

   Solution: It can be called $2^{15} - 1$ times as we have unsigned addition in `addiu`.
   Marking scheme:

   - -1 for 2^{32}
   - -0.5 for 2^{16}

Here is some information about the MIPS instruction set that might be relevant to this question, in addition to that in the reference sheet. The reference sheet also has the instruction format and opcodes for the instructions.

In `addiu` , the immediate operand is actually a 16-bit signed value that is sign-extended to 32-bits before being added to the unsigned operand in the specified register.

Table 1: MIPS register details.

Name	Number	Use
$v0-$v1	2-3	Values for function results and expression evaluations
$t0-$t7	8-15	Temporary registers
$ra	31	Return address

3. [10 marks] We will not consider pipelining for this problem. We are to add a new instruction `soifz` that does the equivalent of the following C-code in assembly instruction.

```
if (ptr[IMM] == 0) {
  ptr[IMM] = 1;
}
```

This new instruction `soifz` will *store one if zero*. If the `IMM`-th element of the array `ptr` is 0, then set that element to 1. (i.e., `IMM`-th element is `IMM` words past the base address of `ptr` in `rs`).

(a) [2 marks] Show the MIPS usage (assembly code) for this new instruction, if `ptr` start address is in `$t0` and `IMM` is 4.

Solution: `soifz 4($t0)`
If equivalent assembly code was given, that was also given marks. Though, this was not the expected answer.

(b) [4 marks] Make the minimum changes to the datapath to enable `soifz` and show these in Figure 1. List all the changes in the Table 2. You may modify or use additional components and control signals. But you may **not otherwise change the basic components** of the datapath such as memory, register file, ALU, etc.

Table 2: Changes in datapath to support `soifz`

(i)	We need a mux that uses DataOut from 'Data Cache' as one input. This will be compared to 0 in a comparator that will be controlled by `soifz` signal.
(ii)	We need a mux in the input 'Data in' to 'Data Cache', which will be $0x00000001$ or 'Data in' depending on the control signal `soifz`.
(iii)	(or) variants with 4:1 mux is also fine
(iv)	

(c) [4 marks] Update the Table 3 with control signals for executing the `soifz` instruction. Include additional signals as needed. Each box should be filled with a 0, 1, 2 or X (don't care).

Table 3: Control values to execute `soifz`

BR_Jump	RegDst	RegWrite	ALUSrc	DataRead	DataWrite	RegInSrc	`soifz`	
0	x	0	1 SE(Imm)	1	1	x	1	

ALU Function: **Addition for address update.**

Solution: Marking scheme

- a) 2 marks for correct usage with syntax also if proper equivalent assembly code for executing the operation.

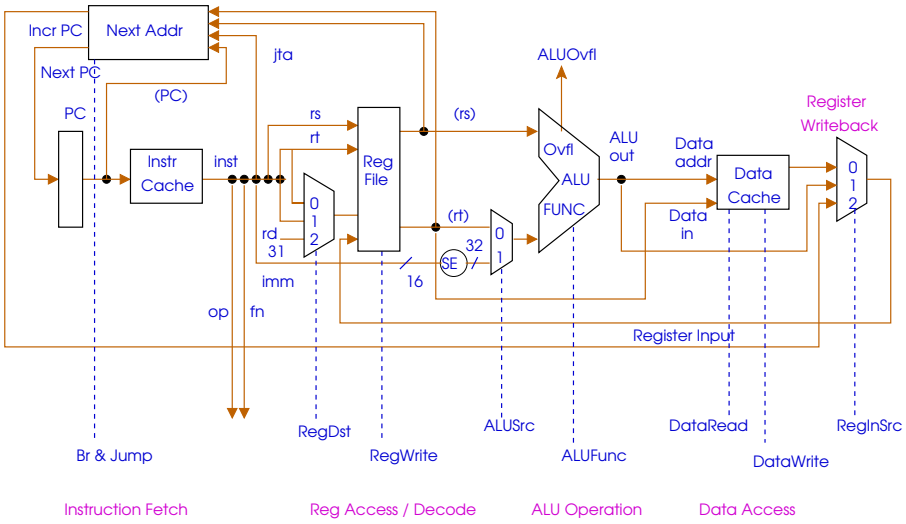

Figure 1: Single-cycle datapath.

- b) Marks given only for operations with d-cache and not registers.
  - 2 marks for identifying correct mechanism ie, dcache output comparison to zero and write back to d cache.
  - 2 marks for proper hardware and control signal to do that .
  - Must have Zero checking logic and MUX at data in input.
- c)The split for 3rd question is as follows :
  - 1 mark for regWrite = 0 (Compulsory)
  - 1 mark for proper data cache signals
  - 1 mark for proper additional control signal usage
  - 0.5 mark for ALUsrc and 0.5 marks for ADD as ALU function

4. [6 marks] These are based on miniMIPS datapaths discussed in class.
   (a) [3 marks] Mention 3 differences between the single cycle and multicycle datapaths.

Solution:

- Single-cycle has separate Instruction and Data cache, whereas multicycle had a single cache (memory)
- Multicycle datapath has intermediate registers to store Instruction and values read from the register file
- It has a single ALU for both arithmatic operations and address computation

   (b) [3 marks] Mention 3 differences between the multicycle and pipelined datapaths.

Solution:

- Pipelined datapath has separate ALU for arithmatic and address computations.
- Pipelined datapath has partitioned registers between stages.
- Pipelined datapath has a Next address computation block in the 2nd stage. The multicycle does not have this.
- Pipelined datapath has separate instruction and data cache.

Solution: Marking scheme:

- if differences are valid (relating to datapath) 1 mark;
- if related to general stuff (like CPI, throughput) 0.5 mark;
- Clearly mentioned in paper that the differnces are not related to the datapath
- incorrect differences 0 mark
- When more than 3 differences, the maximum marks by selecting best 3 points

5. [12 marks] Given the following MIPS code snippet:

```
loop:
        1 addi $t0, $t0, 4
        2 lw   $t1,  0($t0)
        3 sw   $t1, 20($t0)
        4 lw   $r0, 60($t0)
        5 bne  $r0, $0, loop
        6                   <- ##  The following instruction could be anything!
```

NOTE: Here, "no interlocked pipeline stages" implies that if there is a hazard there is no detection mechanism present in the hardware to stall the pipeline. Hence such hazards can lead to incorrect result if not handled appropriately in programming.

(a) [4 marks] Detect hazards and insert no-ops to insure correct operation. Assume no branch delay slot, no forwarding units, and no interlocked pipeline stages. Your answer should take the form of pair(s) of numbers: num@location indicating numberof no-ops should be placed at location.,e.g., if you wanted to place 4 no-ops between lines 2 and 3 (i.e., location=2.5) and 8 no-ops between lines 5 and 6 (i.e., location=5.5), you would write: "4@2.5, 8@5.5".

Solution: 2@1.5, 2@2.5, 2@4.5, 1@5.5

(b) [8 marks] Now, reorder/rewrite the program to maximize performance. Assume branch delay slot and data forwarding units exist, but no interlocked pipeline stages. For unknown reasons, the first instruction after the loop label must be the `addi`. Feel free to insert no-ops where needed. You should be able to do it using 6 instructions per loop (**half credit**) or only 5 (**full credit**). Even a no-op will be considered an instruction. The code following the given loop should not be affected by your reordering.

Solution: Case 1: 6 instructions per loop

```
loop:
        1 addi $t0, $t0, 4
        2 lw $t1, 0($t0)
        4 lw $r0, 60($t0)
        3 sw $t1, 20($t0)
        5 bne $r0, $0, loop
        6 NOP
                            <- ##  The following instruction could be anything!
```

Case 2: 6 instructions per lopp

```
loop:
        1 addi $t0, $t0, 4
        2 lw $t1, 0($t0)
        4 lw $r0, 60($t0)
        6 NOP
        5 bne $r0, $0, loop
        3 sw $t1, 20($t0)
                            <- ##  The following instruction could be anything!
```

Case 3: 5 instructions per loop

```
loop:
        1 addi $t0, $t0, 4
        4 lw $r0, 60($t0)
        2 lw $t1, 0($t0)
```

```
5 bne $r0, $0, loop
3 sw $t1, 20($t0)
                    <- ##  The following instruction could be anything!
```

Solution: Marking scheme:

5a) 0.5 marks for identifying the correct hazard, 0.5 for the correct number of nops for that hazard.

5b) 8 marks for correct 5 cycle implementation, 4 for 6 cycles, 2 for 7 cycle implementation, 0 for anything that changes program.

6. [10 marks] In this problem, we consider a new datapath to improve the performance of the fully-bypassed (data forwarding enabled) 5-stage 32-bit MIPS processor datapath discussed in class. In the new datapath the ALU in the Execute (EX) stage is replaced by a simple adder and the original ALU is moved from the EX stage to the Memory (MEM) stage. The adder in the new 3rd stage (formerly EX) is used only for address calculations involving load/store instructions. For all other instructions, the data is simply forwarded to the 4th stage.

   The ALU will now run in parallel with the data memory in the 4th stage of the pipeline (formerly MEM). During a load/store instruction the ALU is inactive, while the data memory is inactive during the ALU instructions. *In this problem we will ignore jump and branch instructions.*

   (a) [3 marks] Give an example sequence of MIPS instructions (three or fewer instructions) that would cause a pipeline bubble in the original datapath, but not in the new datapath.

   Solution:

   ```
   lw $t1, 8($t2)
   add $t3, $t4, $t1
   ```

   (b) [3 marks] Give an example sequence of MIPS instructions (three or fewer instructions) that would cause a pipeline bubble in the new datapath, but not in the original datapath.

   Solution:

   ```
   add $t1, $t0, $t3
   lw $t4, 8($t1)
   ```

   (c) [4 marks] Consider a MIPS instruction structure architecture (ISA) which only supports register indirect addressing i.e., it does not support base + offset (displacement) addressing mode and all addresses have to be in a register. Assume that the new machine will only have to support this ISA. How should the original (or new) pipelined datapath be modified ? If needed provide a high-level block diagram with important components in the datapath for this new machine. Give one advantage of this modification compared to the original datapath.

   Solution: By eliminating the displacement addressing mode, we need not compute effective addeesses for load or store instructions. This helps us eliminate the ALUs requirement for load/store instructions. Instead of having the 5 stages, we only need to have fours stages. These will be IF, ID, EX/MEM, WB. The EX and MEM stages can be done parallelly as the ALU is not needed by load/store instructions and MEM is not needed by arithmatic instructions.

   A diagram showing the ALU and MEM in the same stage will be sufficient. (high-level block diagram). The remaining will be similar to the earlier pipelined datapath.

   Advantage: Latency for a single instruction has been reduced. The RAW hazard which would happen in the earlier pipelined version will not happen now.

   Solution: Marking scheme:

   - 6 (a) 2 for correct instructions, 1 for justification
   - 6 (b) 2 for correct instructions, 1 for justification
   - 6 (c) 3 for reduction of number of stages from 5 to 4 and justification,
   - 6 (c) 1 for latency reduction advantage, 0.5 for RAW hazard elimination advantage

7. [4 marks] A processor (generating 32-bit **byte addresses**) is connected to a fully-associative cache having four blocks, each block containing four bytes of data. The cache uses Least Recently Used (LRU) replacement. Access times are 5 ns on a hit and a total of 50 ns on a miss (including the 5 ns cache access time).

   (a) [1 mark] What cache hit-ratio is necessary to yield an average memory access time of 14ns ?

   Solution:

   $$14 = h * 5 + (1 - h) * 50 \implies h = 36/45 = 0.8$$

   Marking scheme:

   - 1 for Correct hitrate, 0.5 for mistake in formula
   - 0.5 if value correct but arrived incorrectly (incorrect formula)

   (b) [2 marks] The processor produces 32-bit byte addresses, `A[31:0]`. Which of these bits are compared with `Tags` stored in the cache ?

   Solution: Its a fully associative cache. The bits used for offset is 2 bits (4 bytes in each block). Hence, the remaining bits are used for Tag `A[31:2]`.
   Marking scheme:

   - 2 mark if all correct
   - 1 mark for mentioning 30 bits but not indicating the bits

   (c) [1 mark] How many such comparisons are performed simultaneously for each memory read ?

   Solution: It needs four simultaneous Tag comparisons as it has four blocks.
   Marking scheme:

   - 1 mark for 4 compares, any other value 0 marks

8. [10 marks] Consider a direct-mapped cache, with two blocks, and block capacity of 4-words each (word is 4-bytes). Let the computation that needs to be performed with this cache be an inner-product of two vectors **A** and **B** of length 8 words each. It is the standard inner-product, which involves reading A_i and B_i, multiplying them, and adding the product to a running total in a register. The pseudocode for inner-product computation is as below.

```
out = 0
for i = 0 to 7
  out = out + A[i] * B[i]
end
RETURN ou
```

(a) [3 marks] What is the cache hit-rate if the starting addresses of **A** and **B** are both multiples of 4 ? Justify.

Solution: In direct-mapped cache, with capacity 8-words and block size 4-words, we have two blocks. They are numbered 0 and 1.

- First access to read `A`, will be a miss and a block of 4 words corresponding to `A` will be read and placed in Block-0 of the cache.
- First access to read `B`, will be a miss and a block of 4 words corresponding to `B` will be read and they will replace the data in Block-0.
- Subsequent access to Block 1 will also have the same issue. Missess for both `A` and `B`.
- This leads to 0% hit rate. This is also called as 'thrashing'.

Marking scheme:

- 1 mark for correct hit rate
- 2 marks for justification

(b) [2 marks] Suggest ways of improving the hit-rate in part (a) without changing the cache organization.

Solution:

- One way to eliminate the above 'thrashing' is to store one of the vectors in a position that is not a multiple of 4. This will lead to blocks that are accessed for `A` being not replaced by reading the other vector. This will lead to a hit-rate of 75%.

Marking scheme:

- 2 marks for correct approach
- 0 mark for incorrect approach

(c) [3 marks] What would happen to the hit-rate (relating to (a)) if the cache were two-way set-associative ? Assume cache size same as before (32-bytes).

Solution:

- If it was 2-way set associative, then also thrashing will be avoided. This iwill also have a hit rate of 75%.

Marking scheme:

- 1 mark for just saying hit rate increases
- 2 marks for justification
- 2 marks for correct hit rate but no justification
- 3 marks for correct hit rate and justification

(d) [2 marks] Does this application exploit temporal or spatial locality ? Justify.

Solution: This applications exploits both spatial (successive elements of a vector) and temporal (for loop) locality.

Marking scheme:

- 1 mark for spatial and justification
- 1 mark for temporal and justification

Questions End

Blank sheet

Reference information for 8051

Port P3 of 8051 is a multi-function port.

Additional functions of Port 3 lines

Port Line	P3.7	P3.6	P3.5	P3.4	P3.3	P3.2	P3.1	P3.0
Function	$\overline{RD}$	$\overline{WR}$	T1 in	T0 in	$\overline{INT1}$	$\overline{INT0}$	TxD	RxD

Lines P3.5 and P3.4 can be used as inputs to Timers T1 and T0 respectively.

TCON register at BYTE address 88H								
Bit No.	7	6	5	4	3	2	1	0
Bit Name	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
Bit Addr	8F	8E	8D	8C	8B	8A	89	88
TMOD register at BYTE address 89H								
Bit No.	7	6	5	4	3	2	1	0
Timer:	T1				T0			
Bit Name	G1	C/T1	T1M1	T1M0	G0	C/T0	T0M1	T0M0

Reg. Name	Address	Function
TCON	88H	Timer status
TMOD	89H	Timer modes and Config
TL0	8AH	T0 count register: Low
TL1	8BH	T1 count register: Low
TH0	8CH	T0 count register: High
TH1	8DH	T1 count register: High

SFR `IE` at byte address A8H

Bit No.	7	6	5	4	3	2	1	0
Bit Addr	AF	AE	AD	AC	AB	AA	A9	A8
Bit Name	EA	-	-	ES	ET1	EX1	ET0	EX0
Interrupt on	IE	U	U	SI	TF1	Ex1	TF0	Ex0

SFR `IP` at byte address B8H

Bit No.	7	6	5	4	3	2	1	0
Bit Addr	BF	BE	BD	BC	BB	BA	B9	B8
Bit Name	U	U	U	PS	PT1	PX1	PT0	PX0

Note: The IP bits are in the polling order of interrupts.

SFR `SCON` at byte address 98H

Bit No.	7	6	5	4	3	2	1	0
Bit Addr	9F	9E	9D	9C	9B	9A	99	98
Bit Name	SM0	SM1	SM2	REN	TB8	RB8	TI	RI

SM0	SM1	Serial Mode
0	0	0
0	1	1
1	0	2
1	1	3

`PSW` is located at the address D0H in SFR memory area.

Bit No.	7	6	5	4	3	2	1	0
Bit Name	CY	AC	F0	RS1	RS0	OV	F1	P

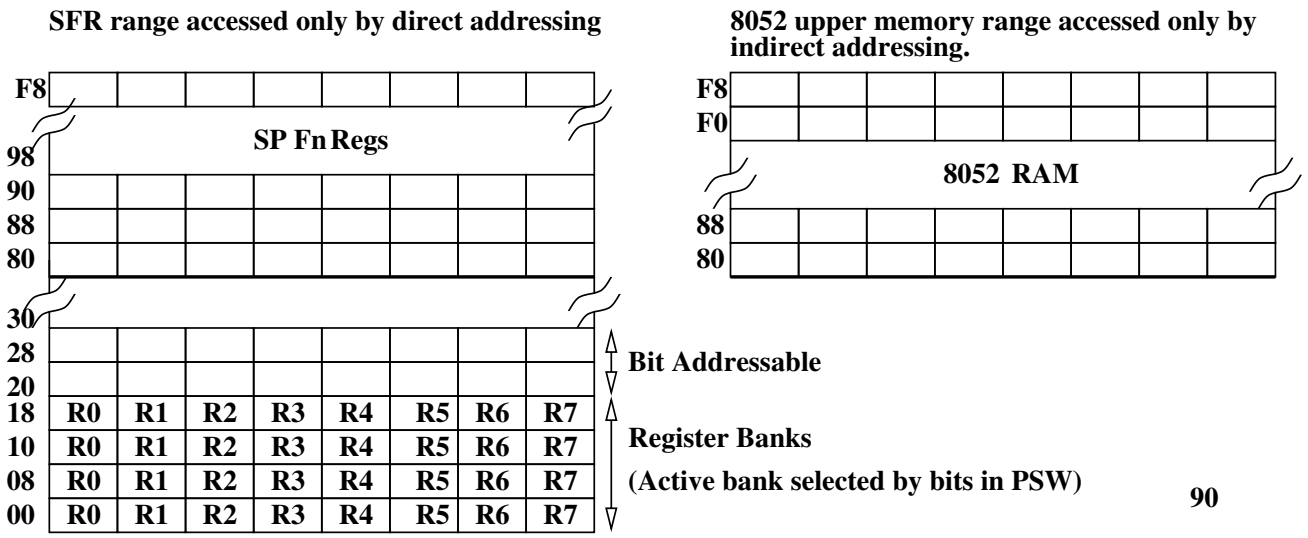

Figure 2: Layout of internal RAM of 8051

F8									FF
F0	B								F7
E8									EF
E0	ACC								E7
D8									DF
D0	PSW								D7
C8									CF
C0									C7
B8	IP								BF
B0	P3								B7
A8	IE								AF
A0	P2								A7
98	SCON	SBUF							9F
90	P1								97
88	TCON	TMOD	TL0	TL1	TH0	TH1			8F
80	P0	SP	DPL	DPH				PCON	87
	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F	

Figure 3: Layout of SFRs in 8051

Reference information for MiniMIPS

Table 4: Subset of MIPS Instruction Set

Class	Instruction	Usage	Meaning	op	fn
Arithmetic	Add	`add rd, rs, rt`	$rd \leftarrow (rs) + (rt)$; with overflow	0	32
	Add unsigned	`addu rd, rs, rt`	$rd \leftarrow (rs) + (rt)$; no overflow	0	33
	Subtract	`sub rd, rs, rt`	$rd \leftarrow (rs) - (rt)$; with overflow	0	34
	Subtract unsigned	`subu rd, rs, rt`	$rd \leftarrow (rs) - (rt)$; no overflow	0	35
	Set less than	`slt rd, rs, rt`	$rd \leftarrow$ if $(rs) < (rt)$ then 1 else 0	0	42
	Add immediate	`addi rt, rs, imm`	$rt \leftarrow (rs) +$ imm; with overflow	8	
	Add immediate unsigned	`addiu rt, rs, imm`	$rt \leftarrow (rs) + imm$; no overflow	9	
	Set less than immediate	`slti rt, rs, imm`	$rt \leftarrow$ if $(rs) < imm$ then1 else 0	10	
Shift	Shift left logical	`sll rd, rt, sh`	$rd \leftarrow (rt)$ left-shifted by sh	0	0
	Shift right logical	`srl rd, rt, sh`	$rd \leftarrow (rt)$ right-shifted by sh	0	2
	Shift left logical variable	`sllv rd, rt, rs`	$rd \leftarrow (rt)$ left-shifted by (rs)	0	4
	Shift right logical variable	`srlv rd, rt, rs`	$rd \leftarrow (rt)$ right-shifted by (rs)	0	6
Logic	AND	`and rd, rs, rt`	$rd \leftarrow (rs)$ AND (rt)	0	36
	OR	`or rd, rs, rt`	$rd \leftarrow (rs)$ OR (rt)	0	37
	XOR	`xor rd, rs, rt`	$rd \leftarrow (rs)$ XOR (rt)	0	38
	NOR	`nor rd, rs, rt`	$rd \leftarrow ((rs)$ OR $(rt))'$	0	39
	AND Immediate	`andi rt, rs, imm`	$rt \leftarrow (rs)$ AND (imm)	12	
	OR Immediate	`ori rt, rs, imm`	$rt \leftarrow (rs)$ OR (imm)	13	
	XOR Immediate	`xori rt, rs, imm`	$rt \leftarrow (rs)$ XOR (imm)	14	
Copy	Load upper immediate	`lui rt,imm`	$rt \leftarrow \{imm, 0x0000\}$	15	
Memory access	Load word	`lw rt, imm(rs)`	$rt \leftarrow mem[(rs) + imm]$	35	
	Load byte	`lb rt, imm(rs)`	Load byte-0, sign-extend	32	
	Load byte unsigned	`lbu rt, imm(rs)`	Load byte-0, zero-extend	36	
	Store word	`sw rt, imm(rs)`	$mem[(rs) + imm] \leftarrow rt$	43	
	Store byte	`sb rt, imm(rs)`	Store byte-0	40	
Control transfer	Jump	`j L`	goto L	2	
	Jump and link	`jal L`	goto L; \$31 $\leftarrow PC + 4$	3	
	Jump register	`jr rs`	goto (rs)	0	8
	Branch less than 0	`bltz rs, L`	if $(rs) < 0$ then goto L	1	
	Branch equal	`beq rs, rt, L`	if $(rs) = (rt)$ then goto L	4	
	Branch not equal	`bne rs, rt, L`	if $(rs) \neq (rt)$ then goto L	5	

Figure 4: MIPS Instruction Formats