Student Name__________________________________

StudentId_____________________________________

NWEN 242
Computer Organization

Mid-Term Test

16 August 2011

Model Solution

Instructions:

Maximum time: 100 minutes.
Answer all questions.
There are 100 marks in total.
Write your answer in the boxes in this paper and hand in all sheets.
Paper foreign language dictionaries are allowed.
Non-programmable calculators are allowed.
Appendix provides some commonly used MIPS instructions and registers for your reference. You may want to tear off the Appendix.

Questions	Marks
1) Basic Concepts	[35 marks]
2) MIPS Assembly Language	[25 marks]
3) ALU	[10 marks]
4) Single Cycle Data Path and Control	[15 marks]
5) Single Instruction Computer	[15 marks]
Total	[100 marks]

Question 1. Basic Terms [35 marks]

Briefly answer the following questions.

a) [2 marks] List the names of the basic components of a computer.

ANSWER

Central processor.
Main memory.
Input/Output units.

b) [2 marks] What is the name of the computer's component that executes program instructions and what are its main components.

ANSWER

Central processor.
The main Components: Control Unit and DataPath

c) [2 marks] What are the names of the three main components of the MIPS data path? (**Note:** data and instruction memory do not belong to the data path.)

ANSWER

Program Counter.
Register File.
Arithmetic-Logic Unit (ALU).

d) [2 marks] The size (or capacity) of a memory unit can be expressed in words, bytes, or bits. Express the size (capacity) of the MIPS Register File in all three of them: words, bytes, and bits.

ANSWER

32 words,
128 bytes,
1024 bits

e) [2 mark] MIPS main memory is often considered as 1-d array of words. How does this view of the main memory affect its addressing?

ANSWER

Addresses of consecutive words are expressed as multiples of 4, i.e. 0,4, 8, 12,…

f) [1 mark] Where does reside the currently executing program in a computer.

ANSWER

In (instruction) memory.

g) [2 marks] What is the name of a program that translates another program written in a higher programming language like Java and what does it produce?

ANSWER

Compiler.
Produces an assembly language program.

h) [2 marks] Assume an assemble language does not have pseudo instructions. What is the relationship between the assembly language and the corresponding machine language instructions?

ANSWER

One assembly language instruction maps into a machine language instruction.

i) [2 marks] Consider an R-type MIPS assembly symbolic instruction (like `add $rd, $rs, $rt`). Show its machine instruction format using fields. Show filed length using the range of bit position numbers.

ANSWER

op	rs	rt	rd	shamnt	funct
31-26	25-21	20-16	15-11	10-6	5-0

j) [2 marks] Consider a branch-type MIPS assembly symbolic instruction (like `beq $rs, $rt, label`).

ANSWER

op	rs	rt	immediate
31-26	25-21	20-16	15-0

k) [2 marks] What is the basic difference between combinational and sequential logic?

ANSWER

The output of a combinational logic circuit depends solely on its inputs. It is memory less.
The output of a sequential logic circuit depends on inputs and the current state. It contains memory.

l) [2 marks] We discussed three basic sequential logic circuits in lectures. What are their names?

ANSWER

RS Latch
Clocked D Latch
D Flip-Flop

m) [2 marks] List the addressing modes of MIPS assembly instructions.

ANSWER

Register addressing
Immediate addressing
Base addressing
PC relative addressing
Pseudo-direct addressing

n) [4 marks] Transform 100111001101_2 into decimal. Show your working.

ANSWER

$2^0 = 1$
$2^2 = 4$
$2^3 = 8$
$2^6 = 64$
$2^7 = 128$
$2^8 = 256$
$2^{11} = 2048$

$100111001101_2 = 2509$

o) [6 marks] Transform 4625_{10} into binary. Show your working.

ANSWER

$4625 = 4096 + 512 + 16 + 1 = 2^{12} + 2^9 + 2^4 + 2^0$

$4625_{10} = 1\ 0010\ 0001\ 0001_2$

The same can be achieved by dividing by 2.

Question 2. Assembly Language [25 marks]

a) [8 marks] Consider the following C code segment:

```
if (x < 15)
   x = x + m;
else
   x = x – m
x++
```

i. [6 marks] Write a sequence of MIPS instructions that directly correspond to this C code segment. Assume register `$s0` contains the integer variable `x` and register `$s1` contains the integer variable `m`. Use temporary registers if necessary. To alter the order of execution of instructions in your MIPS code, do not use jump (`j`) instructions, but only branch (`beq` and `bne`) instruction.

ANSWER

```
      slti $t0, $s0, 15
      beq  $t0, $zero, else
      add  $s0, $s0, $s1
      beq  $zero, $zero, exit
else: sub  $s0, $s0, $s1
exit: addi $s0, $s0, 1
```

ii. [2 Marks] Replace the symbolic address in each branch instruction in your assembly program above by a decimal branch offset.

ANSWER

```
      else = 2
      exit = 1
```

b) [7 marks] We designed a combinational logic block in lectures that detects overflow. Overflow detection can be also performed using MIPS assembly instructions in a simple way.

Assume A and B are negative integers stored in registers `$s1` and `$s2`, respectively. Write a sequence of MIPS assembly instructions that will store A + B in the register `$s3` and if there is an overflow branch to a symbolic address `overflow`.

ANSWER

```
add  $s3, $s1, $s2            #1 mark
slt  $t0, $s3, $zero          #3 marks
bne  $t0, $zero, overflow     #3 marks
```

c) [10 marks] A friend of yours came to you asking for help. He/she had to make an assembly program that corresponds directly to the following C code segment:

```
A[10] = A[20] + 20
```

where the array base address was 1081344_{10}. He/she transformed 1081344_{10} into the following binary

0000 0000 0001 0000 1000 0000 0000 0000

assumed register $s0 is the base address register and made the following MIPS assembly code

```
lui    $s0, 16
addi $s0, $zero, 32768
lw    $t0, 80($s0)
addi $t0, $t0, 20
sw    $t0, 40($s0)
```

but his/her program failed to produce the expected result.

i. [8 marks] What was wrong with your friend's program? Justify your claim.

ANSWER

As a result of

```
    lui    $s0, 16
    addi $s0, $zero, 32768
```

instructions, the content of `$s0` is 1015808 (1048576 - 32768), since sign extension in

```
addi $s0, $zero, 32768
```

turned 32768 into - 32768

ii. [2 marks] Correct your friends program to perform as needed.

ANSWER

```
lui  $s0, 16
ori  $s0, $zero, 32768
lw   $t0, 80($s0)
addi $t0, $t0, 20
sw   $t0, 40($s0)
```

Question 3. ALU Arithmetic – Logic Unit [10 marks]

The 1-bit adder is a combinational logic block that has three inputs and two outputs. The three inputs are: the first operand bit *a*, the second operand bit b, and the *carry-in*. The two outputs are: *carry-out* and *sum*. Your task is to design the combinational logic block that generates the *sum* output.

a) [3 marks] Complete the truth table for the *sum* output below.

ANSWER

a	*b*	*carry-in*	*sum*
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	1	1

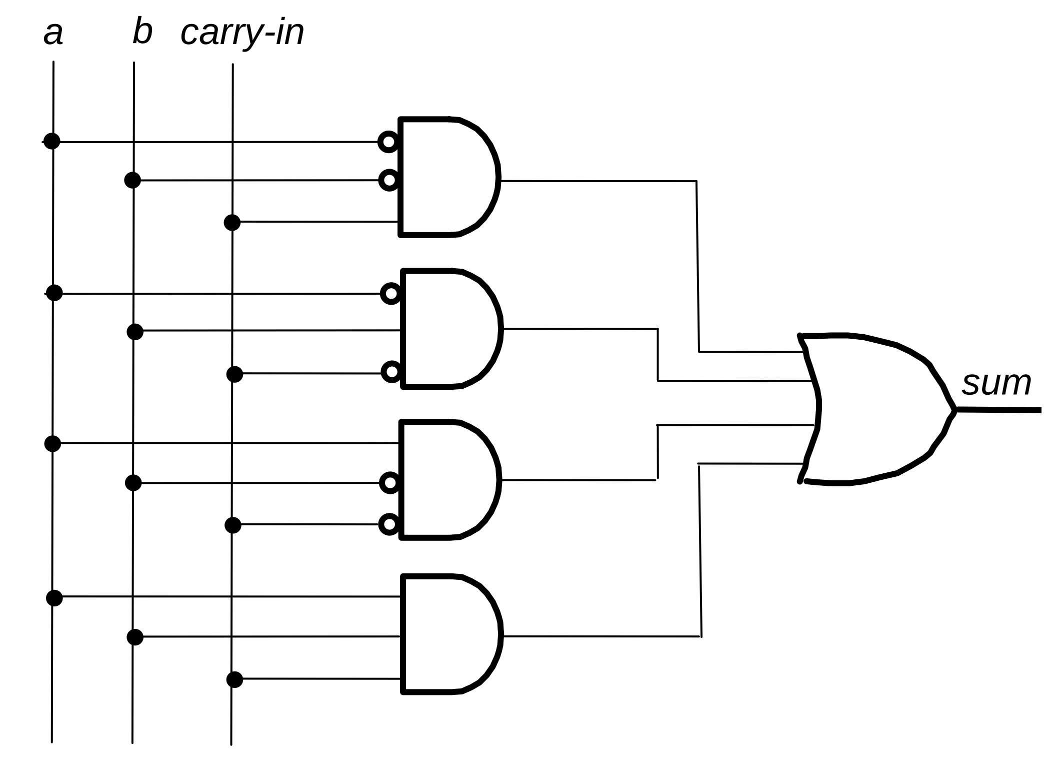
b) [2 marks] Write an expression in Boolean algebra for the sum output o an 1-bit adder.

ANSWER

$$sum = \bar{a} \bullet \bar{b} \bullet carry-in + \bar{a} \bullet b \bullet \overline{carry-in} + a \bullet \bar{b} \bullet \overline{carry-in} + a \bullet b \bullet carry-in$$

c) [5 marks] Use AND, OR, and NOT logic circuits to draw a combinational logic block that produces the *sum* output of an 1-bit adder. Label clearly inputs and the output.

ANSWER



Question 4. Single Cycle Data Path and Control [15 marks]

a) [10 marks] Consider the diagram of the Single Cycle Data Path in Figure 1 on the facing page. On the diagram, show only those lines and components that will be used to execute a MIPS **add** instruction. Trace the lines and circle the components using a coloured pen. Do not pay attention to control signals when answering this question.

b) [5 marks] Consider the diagram of the Single Cycle Data Path in Figure 1 on the facing page again. On the diagram, show only those control signals that will be asserted during the execution of a MIPS **add** instruction. Trace these control signals using a coloured pen.

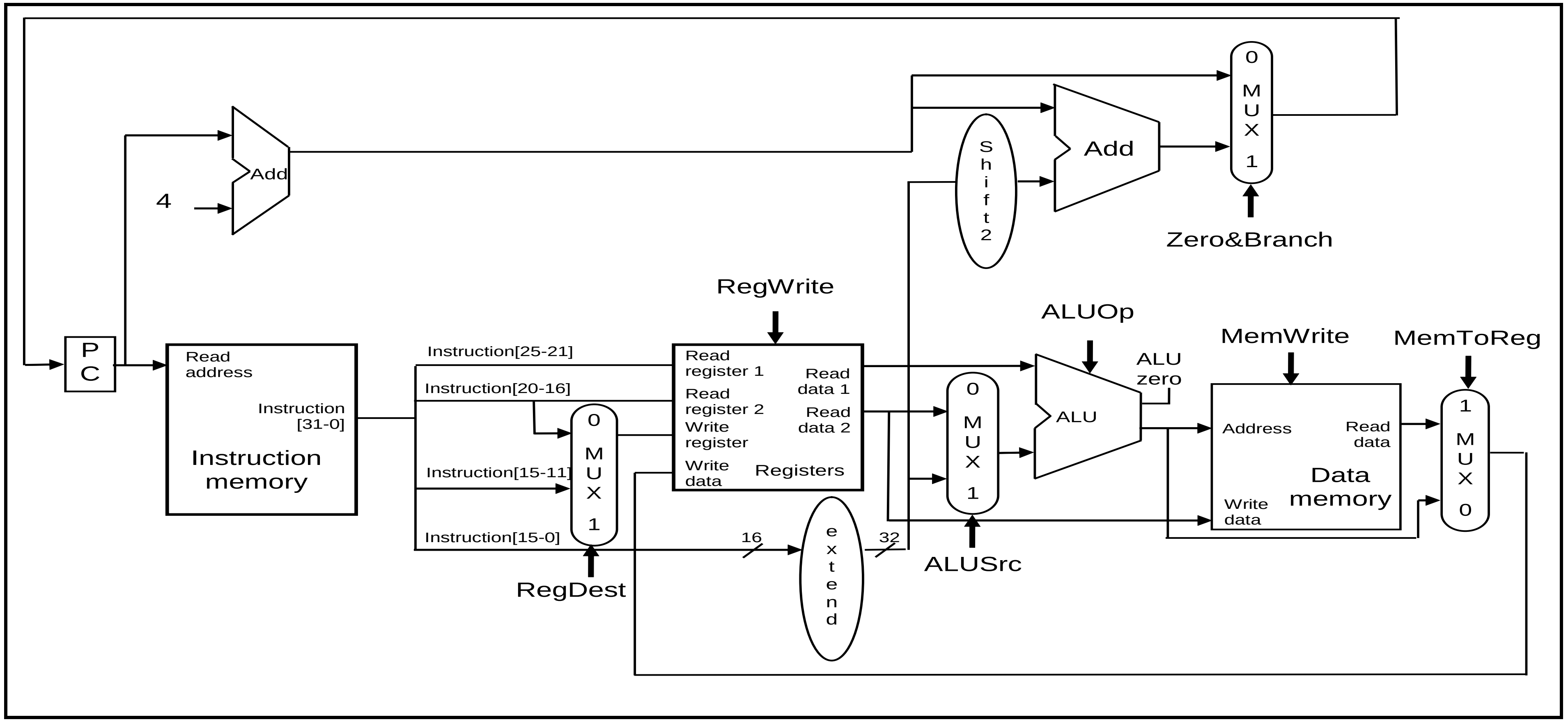


Figure 1. Single Cycle Data Path

ANSWER

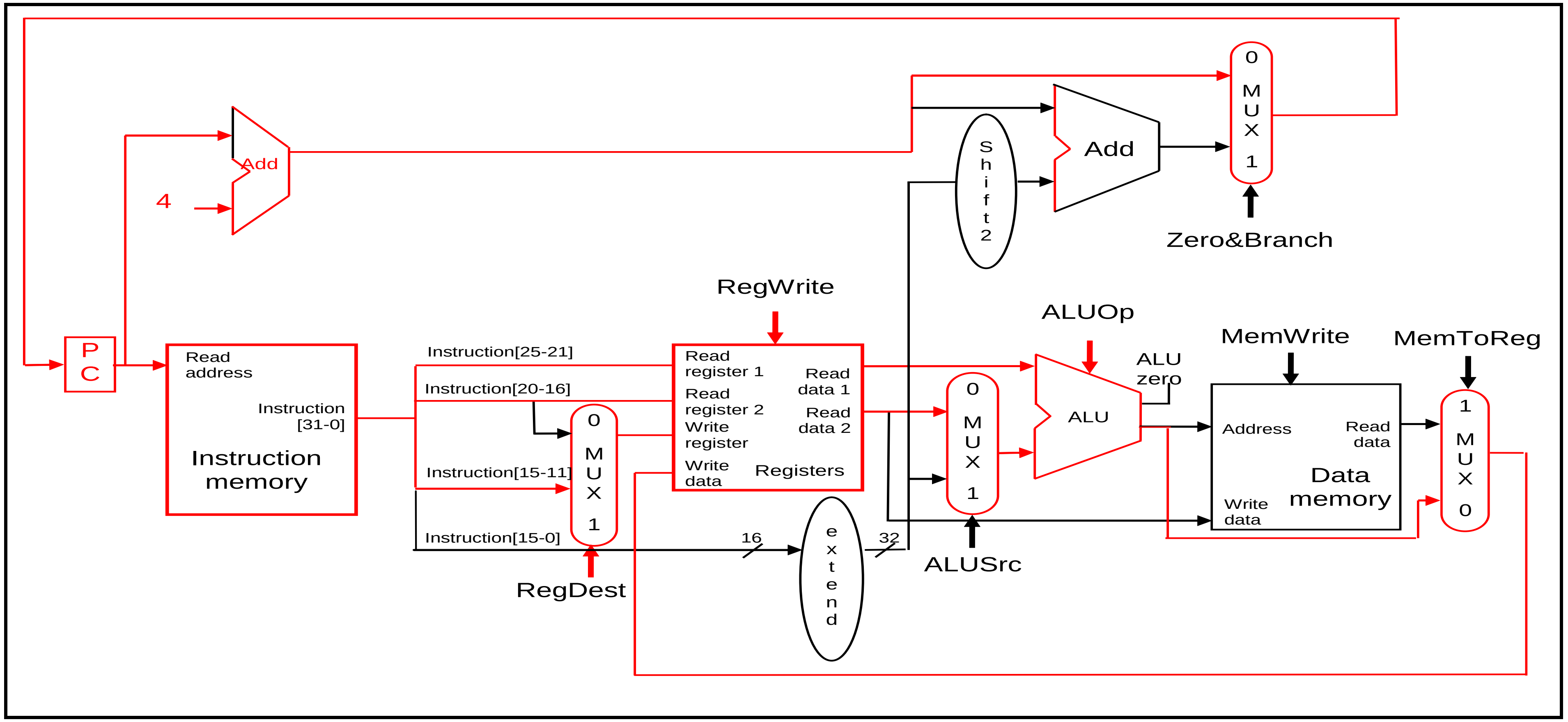


Figure 1. Single Cycle Data Path

Question 5. Single Instruction Computer [15 marks]

Consider a hypothetical machine called SIC:Single Instruction Computer. As its name implies, SIC has only one instruction: subtract and branch if negative (`sbn` for short). The `sbn` instruction has three operands, each being an address of a word memory. For example:

```
sbn a, b, c
```

subtracts the number at address `b` from the number at address `a` and puts the result back into address `a`. If the result is less than zero then the computer will branch to address `c`, otherwise the control will ‘fall through’ to the next instruction. SIC has no registers and no other instructions. Although SIC is so simple, some interesting operations can be emulated.

a) [5 marks] There is a SIC program that copies the number from address `a` to address `b` given in the answer box below. Comment program instructions to show that it really copies number from address `a` (call it `a`) to address `b`.

ANSWER

Memory Address	SIC Instruction	Comment
n	`sbn m, m, n+1`	m = 0
n + 1	`sbn m, a, n+2`	m = -a
n + 2	`sbn b, b, n+3`	b = 0
n + 3	`sbn b, m, n+4`	b = a

b) [10 marks] Write the shortest program to multiply `a` and `b`, leaving the result in `c`. Assume: `a` and `b` are positive integers stored at memory addresses `a` and `b`, respectively, and memory location `m` contains 1. Write and comment your SIC program into the answer box below. To exit the program, branch to the first memory word below the last instruction of your program.

Note: the number of rows in the answer box is greater than the number of lines of the shortest program that multiplies `a` and `b`.

ANSWER

Memory Address	SIC Instruction	Comment
n	`sbn c, c, n+1`	c = 0
n + 1	`sbn p, p, n+2`	p = 0
n + 2	`sbn d, d, n+3`	d = 0
n + 3	`sbn b, m, n+4`	b = b – 1
n + 4	`sbn d, a, n+5`	d = -a
n + 5	`sbn c, d, n+6`	c = i*a
n + 6	`sbn b, m, n+8`	i = i + 1, b = b – (i + 1) if b < 0 go to exit
n + 7	`sbn p, m, n+5`	go to n + 5
n + 8		

APPENDIX

Commonly Used MIPS Instructions

Arithmetic and Logical Instructions

`add` `Rdest, Rsrc1, Src2`

Addition (with overflow)

`addi` `Rdest, Rsrc1, Imm`

Addition Immediate (with overflow)

`addu` `Rdest, Rsrc1, Src2`

Addition (without overflow)

`and` `Rdest, Rsrc1, Src2`

AND

`andi` `Rdest, Rsrc1, Imm`

AND Immediate. Put the logical AND of the integers from register Rsrc1 and Src2 (or Imm) into register Rdest.

`or` `Rdest, Rsrc1, Src2`

OR

`ori` `Rdest, Rsrc1, Imm`

OR Immediate. Put the logical OR of the integers from register Rsrc1 and Src2 (or Imm) into register Rdest.

`sll` `Rdest, Rsrc1, Src2`

Shift Left Logical

`srl` `Rdest, Rsrc1, Src2`

Shift Right Logical

`sub` `Rdest, Rsrc1, Src2`

Subtract (with overflow)

`subu` `Rdest, Rsrc1, Src2`

Subtract (without overflow) Put the difference of the integers from register Rsrc1 and Src2 into register Rdest.

Constant-Manipulating Instructions

`li` `Rdest, imm`

Load Immediate. Move the immediate imm into register Rdest.

`lui` `Rdest, imm`

Load Upper Immediate Load the lower halfword of the immediate imm into the upper halfword of register Rdest. The lower bits of the register are set to 0.

Comparison Instructions

slt `Rdest, Rsrc1, Src2`
: Set Less Than

slti `Rdest, Rsrc1, Imm`
: Set Less Than Immediate

sltu `Rdest, Rsrc1, Src2`
: Set Less Than Unsigned

Branch and Jump Instructions

beq `Rsrc1, Src2, label`
: Branch on Equal. Conditionally branch to the instruction at the label if the contents of register Rsrc1 equals Src2.

bne `Rsrc1, Src2, label`
: Branch on Not Equal. Conditionally branch to the instruction at the label if the contents of register Rsrc1 are not equal to Src2.

j `label`
: Jump. Unconditionally jump to the instruction at the label.

jal `label`
: Jump and Link

jalr `Rsrc`
: Jump and Link Register. Unconditionally jump to the instruction at the label or whose address is in register Rsrc. Save the address of the next instruction in register 31.

jr `Rsrc`
: Jump Register. Unconditionally jump to the instruction whose address is in register Rsrc.

Load Instructions

lw `Rdest, address`
: Load Word. Load the 32-bit quantity (word) at *address* into register Rdest.

lb `Rdest, address`
: Load Byte. Load the 8-bit quantity (byte) at *address* into register Rdest.

Store Instructions

sw `Rsrc, address`
: Store Word. Store the word from register Rsrc at *address*.

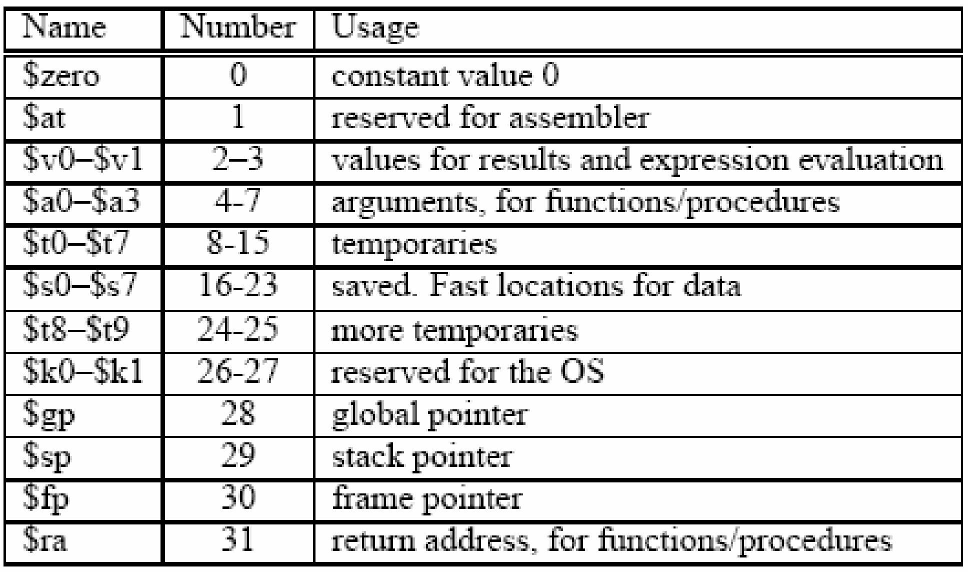
sb `Rsrc, address`
: Store Byte. Store the byte from register Rsrc at *address*.

Commonly Used MIPS Fields

There are six commonly used MIPS fields: *op, rs, rt, rd, shamt*, and *funct*. The *op* and *funct* are usually used to represent and distinguish between different operations/instructions. The following table gives the *op* and *funct* for the commonly used MIPS instructions.

Instructions	**op**	**Funct**
add	0	32
sub	0	34
lw	35	NA
sw	43	NA
beq	4	NA
bne	5	NA
slt	0	42
j	2	NA
jal	3	NA
jr	0	8

Registers:



Name	Number	Usage
$zero	0	constant value 0
$at	1	reserved for assembler
$v0–$v1	2–3	values for results and expression evaluation
$a0–$a3	4-7	arguments, for functions/procedures
$t0–$t7	8-15	temporaries
$s0–$s7	16-23	saved. Fast locations for data
$t8–$t9	24-25	more temporaries
$k0–$k1	26-27	reserved for the OS
$gp	28	global pointer
$sp	29	stack pointer
$fp	30	frame pointer
$ra	31	return address, for functions/procedures