**Nov. 22, 2014**

# ICS DEPARTMENT

## ICS 233

## COMPUTER ARCHITECTURE & ASSEMBLY LANGUAGE

**Midterm Exam**

**First Semester (141)**

**Time: 1:00-3:30 PM**

Student Name : _KEY____________________________________________

Student ID.     : ________________________________________________

| Question | Max Points | Score |
|---|---|---|
| Q1 | 35 | |
| Q2 | 7 | |
| Q3 | 20 | |
| Q4 | 18 | |
| Q5 | 10 | |
| Q6 | 10 | |
| Total | 100 | |

Dr. Aiman El-Maleh

Dr. Samer Arafat

**[35 Points]**

**(Q1)** Fill in the blank in each of the following questions:

**(1)** Each memory cell of DRAM holds 1 bit of information and consists of one transistor(s) and one capacitor(s). (1 Point)

**(2)** Typically, the SRAM technology is used for the cache memory. (1 Point)

**(3)** Given a magnetic disk with rotation speed = 15000 rotations per minute and average seek time 10 milliseconds, then the time needed for one rotation is 60/15000=0.004s=4 milliseconds. (1 Point)

**(4)** A processor with 2.0 GHz speed has a $1/(2x10^{9})=0.5x10^{-9}s=500x10^{-12}s=500$ picoseconds clock cycle duration. (1 Point)

**(5)** A certain chip manufacturing process produces 25 bad dies, on average. Given that the total number of dies on any given wafer is 300, then, this process has a yield equal to (300-25)/300=275/300=91.67 %. (1 Point)

**(6)** Cache memory is faster than random access memory and slower than registers. (1 Point)

**(7)** One main advantage for programming in high-level language is program development is faster or programs are portable. (1 Point)

**(8)** One main advantage for programming in assembly language is space and time efficiency or accessibility to system hardware. (1 Point)

**(9)** Assuming variable Array is defined as shown below:

Array: .byte 1, -1, 2, -2, 3, -3, 4, -4

After executing the following sequence of instructions, the content of the three registers is \$t1=0x00000003 \$t2=0xffffd03 and \$t3=0xfc04fd03.

(3 Points)

```
la $t0, Array
lbu $t1, 4($t0)
lh $t2, 4($t0)
lw $t3, 4($t0)
```

**(10)** Assume that the instruction j NEXT is at address 0x0040002c in the text segment, and the label NEXT is at address 0x00400018. Then, the address stored in the assembled instruction for the label NEXT is <u>0x0100006</u>. (2 Point)

**(11)** Assume that the instruction bne $t0, $t1, NEXT is at address 0x0040002c in the text segment, and the label NEXT is at address 0x00400018. Then, the address stored in the assembled instruction for the label NEXT is <u>(0x00400018-0x00400030)/4=0xfffa</u>. (2 Points)

**(12)** Given that the instruction jal MyProc is at address 0x0040002c in the text segment, and that MyProc is at address 0x00400018. Then, the address stored in $ra register after executing this instruction is <u>0x00400030.</u>

(1 Point)

**(13)** To allocate 10 words, each initialized by 0, we use the following assembler directive <u>.word 0:10</u>.

(1 Point)

**(14)** The pseudo instruction `bge $s2, $s1, Next` is implemented by the following minimum MIPS instructions:
<u>slt $at, $s2, $s1</u>
<u>beq $at, $0, Next</u>

(2 Points)

**(15)** The code given below prints the following:
<u>Midterm Exam</u>
<u>                ICS 233 is easy!!</u>

*Note that the ASCII code for the line feed character is 10 and the ASCII code for the carriage return character is 13.* (2 Points)

```
MSG: .ascii  "Midterm Exam "
     .byte 10
     .ascii "ICS 233 "
     .asciiz "is easy !! "

       li $v0, 4
       la $a0, MSG
       syscall
```

**(16)** Using minimum native MIPS instructions, the assembly code to Jump to label L1 if bits 0, 2, and 5 in $t0 are all set (i.e. =1) is:
ori $t1 $0, 0x25
andi $t0, $t0, $t1
beq $t0, $t1, L1

(3 points)

**(17)** To multiply the **signed** content of register $t0 by 63.75 without using multiplications and division instructions, we use the following instructions:
sll $t1, $t0, 6
sra $t2, $t0, 2
subu $t0, $t1, $t2

(3 points)

**(18)** Assuming that all registers contain signed numbers, the MIPS assembly code (with minimum execution time) to implement the equation $v0=(5-16*$a0)/($a1) is :

ori $v0, $0, 5
shl $t0, $a0, 4
subu $v0, $v0. $t0
div $v0, $a1
mflo $v0

(3 points)

**(19)** Suppose that we would like to translate 8-bit numbers into characters according to a given translation table. Part of the translation table is shown below. The MIPS assembly code to translate a number in register $t0 according to the translation table below and store the resulting character in the same register is (e.g. if $t0=3 the program should store 'G' in $t0):

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | … |
|---|---|---|---|---|---|---|---|---|---|
| 'a' | 'C' | 'x' | 'G' | 'y' | '!' | 'h' | '?' | '-' | … |

.data Table .ascii "aCxGy!h?- …."
la $t1, Table
add $t0, $t0, $t1
lb $t0, 0($t0)

(2 points)

**(20)** Consider a simplified 5-bit floating-point representation following the general guidelines of the IEEE standard format. Suppose that the number of bits used for the exponent is 2 bits and for the fraction is 2 bits. Then, the smallest and largest positive values of normalized numbers that can be represented using this representation are $1.00\text{x}2^0=1$ and $1.11\text{x}2^1=14/4=3.5$ and the largest error in this representation is 0.25. (3 points)

**[7 Points]**

**(Q2)**

**(i)** Assuming that <u>one byte is typed into each small block</u> (one word per row) in the table below, fill out the table for the following data segment. Assume a Little Endian ordering, which is the same as the Mars 4.4 simulator default. Start from the top and work out the rest of the memory segment. Assume that the address of the first byte is 0x10010000. Remember that hex numbers start with 0x. Note that you do not need to show the ASCII code of characters. (5 Points)

```
.data
    var1:  .BYTE   3,'1', -7
    .ALIGN  0
    var2:  .WORD   10
    str1:  .ASCIIZ "ICS"
    .ALIGN  2
    var3:  .WORD   0xabcdef22
    .ALIGN  3
    Var4:  .HALF   -1
```

| MSB | | | LSB | Address |
|---|---|---|---|---|
| 0a | f9 | ‘1’ | 03 | 0x10010000 |
| ‘I’ | 00 | 00 | 00 | 0x10010004 |
| | 00 | ‘S’ | ‘C’ | 0x10010008 |
| ab | cd | ef | 22 | 0x1001000c |
| | | ff | ff | 0x10010010 |
| | | | | 0x10010014 |

**(ii)** Fill out the symbol table, below, that corresponds to the data segment in Part (i), above. (2 Points)

| Label | Address |
|---|---|
| var1 | 0x10010000 |
| var2 | 0x10010003 |
| str1 | 0x10010007 |
| var3 | 0x1001000c |
| var4 | 0x10010010 |

**[20 Points]**

**(Q3) Answer the following questions. Show how you obtained your answer:**

**(i)** Determine what will be displayed after executing the following code: (5 Points)

```
li $a0, 23
li $a1, 5
div $a0, $a1
mflo $a0
li $v0, 1
syscall
li $a0, '.'
li $v0, 11
syscall
li $t0, 10
mfhi $t1
mul $t1, $t1, $t0
div $t1, $a1
mflo $a0
li $v0, 1
syscall
```

The program will display 4.6 which is the result of dividing 23 by 5.

**(ii)** Given the following definition in the data segment:

```
Array: .word   0,  1,  2,  3,   4
       .word   5,  6,  7,  8,   9
       .word  10, 11, 12, 13,  14
```

Determine the content of Array after executing the following code**:** (5 Points)

```
       la $t0, Array
       li $t1, 5
       li $t2, 20
       addu $t2, $t2, $t0
       li $t3, 40
       addu $t3, $t3, $t0
Next:  lw $t4, 0($t2)
       lw $t5, 0($t3)
       sw $t4, 0($t3)
       sw $t5, 0($t2)
       addi $t2, $t2, 4
       addi $t3, $t3, 4
       addi $t1, $t1, -1
       bnez $t1 Next
```

The code will swap row 1 and row 2 in the array and the content of Array after executing the code will be:

```
Array: .word   0,  1,  2,  3,   4
       .word  10, 11, 12, 13,  14
       .word   5,  6,  7,  8,   9
```

**(iii)** Determine what will be displayed after executing the following code: (5 Points)

```
li $t0, 0x1d76
andi $a0, $t0, 0x1f
li $v0, 1
syscall
li $a0, '-'
li $v0, 11
syscall
srl $t0, $t0, 5
andi $a0, $t0, 0xf
li $v0, 1
syscall
li $a0, '-'
li $v0, 11
syscall
srl $t0, $t0, 4
andi $a0, $t0, 0x3ff
addu $a0, $a0, 2000
li $v0, 1
syscall
```

The program will display 22-11-2014, which is the date of the exam!.

**(iv)** Given the following definition in the data segment: (5 Points)

```
TABLE: .asciiz  "Emad Ali Anas"
```

Determine the content of TABLE after executing the following code:

```
        li $t0, 'a'
        li $a0, '*'
        la $t1, TABLE
        addi $t1, $t1, -1
Next:   addi $t1, $t1, 1
        lbu $t2, 0($t1)
        beq $t2, $0, ENL
        ori $t2, $t2, 0x20
        bne $t2, $t0, Next
        sb $a0, 0($t1)
        j Next
ENL:
```

The program will replace all occurrences of ‘a’ and ‘A’ in Table by ‘*. Thus the content of TABLE will be:

```
TABLE: .asciiz  "Em*d *li *n*s"
```

**[18 Points]**

**(Q4)** Write separate MIPS assembly code fragments with minimum instructions to implement each of the given requirements.

**(i)** Given the following high level language code structure, write down the corresponding MIPS assembly language instructions: (6 Points)

```
i = 1;
size = 10;
while (i < size || A[i] !=0){
      A[i] = A[i] + A[i - 1] ;
      i = i + 1;
 }
```

Assume that the assembler has assigned `i` to register `$s0`, `size` to register `$s1`, and has stored the address of array `A` in register `$s2`.

```
            li $s0, 1
            li $s1, 10
While:      sll $t0, $s0, 2
            addu $t0, $s2, $t0
            lw $t1, 0($t0)
            bne $t1, $0, WhileBody
            bge $s0, $s1, EndWhile
WhileBody:  lw $t2, -4($t0)
            addu $t1, $t1, $t2
            sw $t1, 0($t0)
            addiu $s0, $s0, 1
            j While
EndWhile:
```

**(ii)** Assuming that functions F and G receive two arguments in $a0 and $a1 and return their results in $v0, implement the function F given below saving needed registers on the stack. Save changed registers according to the assumed programming convention. (6 Points)

```
int F(int a, int b) {
        return  a+G(b, G(a, b));
}
```

```
F:    addiu   $sp, $sp, -12     # frame = 12 bytes
      sw      $ra, 0($sp)       # save $ra
      sw      $a0, 4($sp)       # save argument a
      sw      $a1, 8($sp)       # save argument b
      jal     G                 # call g(a,b)
      lw      $a0, 8($sp)       # $a0 = b
      move    $a1, $v0          # $a1 = g(a,b)
      jal     G                 # call g(b, g(a,b))
      lw      $a0, 4($sp)       # $a0 = a
      addu    $v0, $a0, $v0     # $v0 = a+G(b, G(a, b))
      lw      $ra, 0($sp)       # restore $ra
      addiu   $sp, $sp, 12      # free stack frame
      jr      $ra               # return to caller
```

**(iii)** Write a procedure that counts the number of even and odd integers that are input via a keyboard. The user will continue to enter nonnegative integers until he enters -1 to terminate the input. The procedure is called `countevenodd` and returns the total count of odd integers in register `$v1` and the total count of even integers in register `$v0`. Assume that a main program prompts the user asking for input and that the main program will print the output counts with proper messages. You do not need to write the main program code. (6 Points)

```
Countevenodd:
        xor $t0, $t0, $t0         # number counter
        xor $v1, $v1, $v1         # odd counter
Loop:
        li $v0, 5
        syscall                   # read integer
        beq $v0, -1, EndLoop
        andi $v0, $v0, 1
        add  $v1, $v1, $v0        # count number of odd's
        addi $t0, $t0, 1
        j Loop
EndLoop:
        subu $v0, $t0, $v1        # compute number of even
        jr $ra
```

**[10 Points]**

**(Q5)**

**(i)** Assume that we have a **Multiplicand = 1100** and a **Multiplier = 0011.**

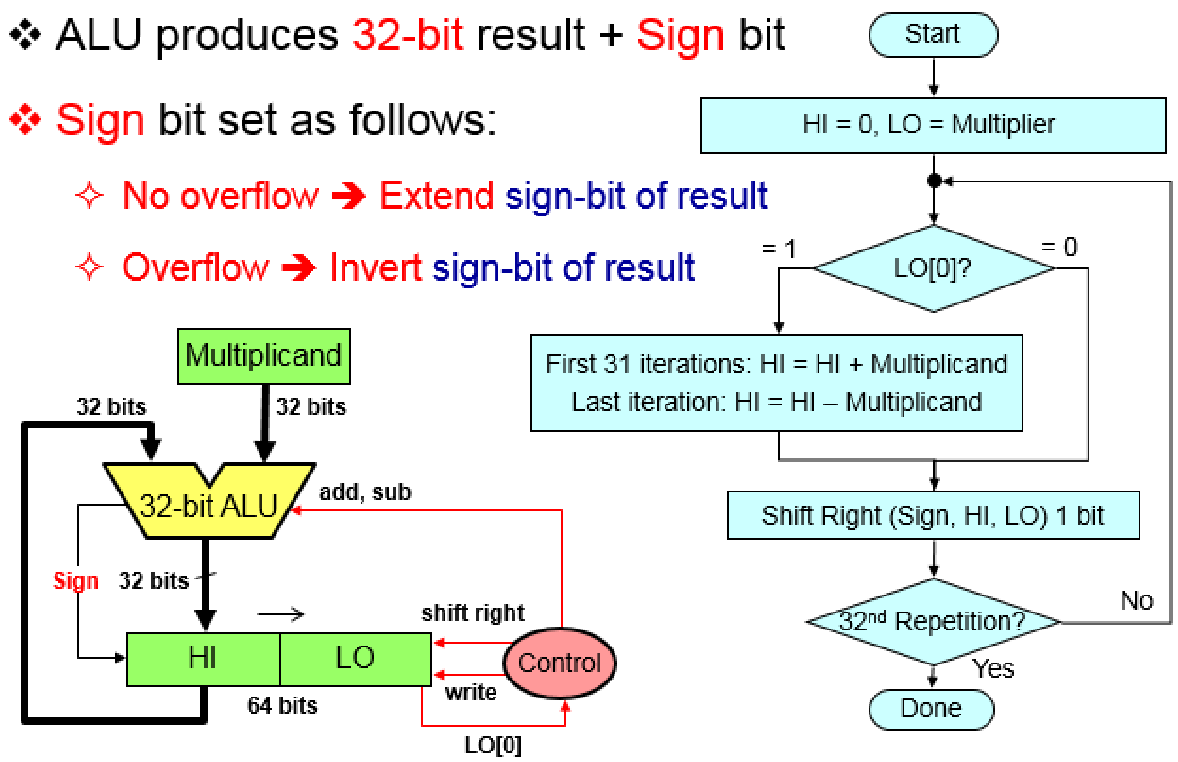
Using the revised unsigned multiplication hardware, show the unsigned multiplication product for the given numbers, above. The result of the multiplication should be an 8 bit unsigned number in the HI and LO registers. Show all the steps of your work.

(4 Points)

| Iteration | | Multiplicand | Carry | Product = HI,LO |
|---|---|---|---|---|
| 0 | Initialize | 1100 | | 00000011 |
| 1 | LO[0] = 1 => ADD | | 0 | 11000011 |
| | Shift Right Product = (HI, LO) | | | 01100001 |
| 2 | LO[0] = 1 => ADD | | 1 | 00100001 |
| | Shift Right Product = (HI, LO) | | | 10010000 |
| 3 | LO[0] = 0 => Do Nothing | | 0 | 10010000 |
| | Shift Right Product = (HI, LO) | | | 01001000 |
| 4 | LO[0] = 0 => Do Nothing | | 0 | 01001000 |
| | Shift Right Product = (HI, LO) | | | 00100100 |

**(ii)** Show the hardware diagram that corresponds to the revised integer (signed) multiplication. **<u>Carefully label all parts and connections</u>** in your diagram.

(6 Points)



**[10 Points]**

**(Q6)**

**(i)** What is the decimal value of the following single-precision floating-point number?

`1100 0010 1110 1101 1000 0000 0000 0000.` (2 Points)

= - $(1.1101101100000000...0)_2$ * $2^{(133-127)}$
= - $(1.1101101100000000...0)_2$ * $2^6$
= - $(1110110.1100000000...0)_2$ = - 118.75

**(ii)** Show the single-precision floating-point binary representation for: **120.125**.

(2 Points)

120.125=$(1111000.001)_2$ = $(1.111000001)_2$ * $2^6$
Exp. = 6 +127=133
Single precision binary representation:

`0100 0010 1111 0000 0100 0000 0000 0000`

**(iii)** Perform the following floating-point operation rounding the result to the **nearest even**. Perform the operation using **guard**, **round** and **sticky** bits.

```
    1100 0001 1000 0000 0000 0000 0000 0100
+   0100 0011 1000 1000 0000 0000 0000 0000
```

(6 Points)

```
     1.000 1000 0000 0000 0000 0000 000 x 2^8
-    1.000 0000 0000 0000 0000 0100 000 x 2^4
=    1.000 1000 0000 0000 0000 0000 000 x 2^8
-    0.000 1000 0000 0000 0000 0000 010 x 2^8 (align)

=   01.000 1000 0000 0000 0000 0000 000 x 2^8
+   11.111 0111 1111 1111 1111 1111 110 x 2^8 (2's complement)
=   00.111 1111 1111 1111 1111 1111 110  x 2^8
=   +0.111 1111 1111 1111 1111 1111 110  x 2^8
=   +1.111 1111 1111 1111 1111 1111 100  x 2^7 (normalize)
```

Next, we round to the nearest even by adding 1 and the result becomes:

```
=  +10.000 0000 0000 0000 0000 0000      x 2^7 (round)
```

Next, we renormalize the result and the result becomes:

```
=   +1.000 0000 0000 0000 0000 0000      x 2^8 (renormalize)
```

# Syscall Services:

| Service | $v0 | Arguments / Result |
|---|---|---|
| Print Integer | 1 | $a0 = integer value to print |
| Print Float | 2 | $f12 = float value to print |
| Print Double | 3 | $f12 = double value to print |
| Print String | 4 | $a0 = address of null-terminated string |
| Read Integer | 5 | Return integer value in $v0 |
| Read Float | 6 | Return float value in $f0 |
| Read Double | 7 | Return double value in $f0 |
| Read String | 8 | $a0 = address of input buffer<br>$a1 = maximum number of characters to read |
| Print Char | 11 | $a0 = character to print |
| Read Char | 12 | Return character read in $v0 |

# MIPS Instructions:

| Instruction | Meaning | R-Type Format | | | | | |
|---|---|---|---|---|---|---|---|
| add $s1, $s2, $s3 | $s1 = $s2 + $s3 | op = 0 | rs = $s2 | rt = $s3 | rd = $s1 | sa = 0 | f = 0x20 |
| addu $s1, $s2, $s3 | $s1 = $s2 + $s3 | op = 0 | rs = $s2 | rt = $s3 | rd = $s1 | sa = 0 | f = 0x21 |
| sub $s1, $s2, $s3 | $s1 = $s2 – $s3 | op = 0 | rs = $s2 | rt = $s3 | rd = $s1 | sa = 0 | f = 0x22 |
| subu $s1, $s2, $s3 | $s1 = $s2 – $s3 | op = 0 | rs = $s2 | rt = $s3 | rd = $s1 | sa = 0 | f = 0x23 |

| Instruction | Meaning | R-Type Format | | | | | |
|---|---|---|---|---|---|---|---|
| and $s1, $s2, $s3 | $s1 = $s2 & $s3 | op = 0 | rs = $s2 | rt = $s3 | rd = $s1 | sa = 0 | f = 0x24 |
| or $s1, $s2, $s3 | $s1 = $s2 | $s3 | op = 0 | rs = $s2 | rt = $s3 | rd = $s1 | sa = 0 | f = 0x25 |
| xor $s1, $s2, $s3 | $s1 = $s2 ^ $s3 | op = 0 | rs = $s2 | rt = $s3 | rd = $s1 | sa = 0 | f = 0x26 |
| nor $s1, $s2, $s3 | $s1 = ~($s2|$s3) | op = 0 | rs = $s2 | rt = $s3 | rd = $s1 | sa = 0 | f = 0x27 |

| Instruction | Meaning | R-Type Format | | | | | |
|---|---|---|---|---|---|---|---|
| sll $s1,$s2,10 | $s1 = $s2 << 10 | op = 0 | rs = 0 | rt = $s2 | rd = $s1 | sa = 10 | f = 0 |
| srl $s1,$s2,10 | $s1 = $s2>>>10 | op = 0 | rs = 0 | rt = $s2 | rd = $s1 | sa = 10 | f = 2 |
| sra $s1, $s2, 10 | $s1 = $s2 >> 10 | op = 0 | rs = 0 | rt = $s2 | rd = $s1 | sa = 10 | f = 3 |
| sllv $s1,$s2,$s3 | $s1 = $s2 << $s3 | op = 0 | rs = $s3 | rt = $s2 | rd = $s1 | sa = 0 | f = 4 |
| srlv $s1,$s2,$s3 | $s1 = $s2>>>$s3 | op = 0 | rs = $s3 | rt = $s2 | rd = $s1 | sa = 0 | f = 6 |
| srav $s1,$s2,$s3 | $s1 = $s2 >> $s3 | op = 0 | rs = $s3 | rt = $s2 | rd = $s1 | sa = 0 | f = 7 |

| Instruction | Meaning | I-Type Format | | | |
|---|---|---|---|---|---|
| addi $s1, $s2, 10 | $s1 = $s2 + 10 | op = 0x8 | rs = $s2 | rt = $s1 | $imm^{16}$ = 10 |
| addiu $s1, $s2, 10 | $s1 = $s2 + 10 | op = 0x9 | rs = $s2 | rt = $s1 | $imm^{16}$ = 10 |
| andi $s1, $s2, 10 | $s1 = $s2 & 10 | op = 0xc | rs = $s2 | rt = $s1 | $imm^{16}$ = 10 |
| ori $s1, $s2, 10 | $s1 = $s2 | 10 | op = 0xd | rs = $s2 | rt = $s1 | $imm^{16}$ = 10 |
| xori $s1, $s2, 10 | $s1 = $s2 ^ 10 | op = 0xe | rs = $s2 | rt = $s1 | $imm^{16}$ = 10 |
| lui $s1, 10 | $s1 = 10 << 16 | op = 0xf | 0 | rt = $s1 | $imm^{16}$ = 10 |

| Instruction | Meaning | Format | | | |
|---|---|---|---|---|---|
| j label | jump to label | $op^6$ = 2 | $imm^{26}$ | | |
| beq rs, rt, label | branch if (rs == rt) | $op^6$ = 4 | $rs^5$ | $rt^5$ | $imm^{16}$ |
| bne rs, rt, label | branch if (rs != rt) | $op^6$ = 5 | $rs^5$ | $rt^5$ | $imm^{16}$ |
| blez rs, label | branch if (rs<=0) | $op^6$ = 6 | $rs^5$ | 0 | $imm^{16}$ |
| bgtz rs, label | branch if (rs > 0) | $op^6$ = 7 | $rs^5$ | 0 | $imm^{16}$ |
| bltz rs, label | branch if (rs < 0) | $op^6$ = 1 | $rs^5$ | 0 | $imm^{16}$ |
| bgez rs, label | branch if (rs>=0) | $op^6$ = 1 | $rs^5$ | 1 | $imm^{16}$ |

| Instruction | Meaning | Format | | | | | |
|---|---|---|---|---|---|---|---|
| slt rd, rs, rt | rd=(rs<rt?1:0) | $op^6$ = 0 | $rs^5$ | $rt^5$ | $rd^5$ | 0 | 0x2a |
| sltu rd, rs, rt | rd=(rs<rt?1:0) | $op^6$ = 0 | $rs^5$ | $rt^5$ | $rd^5$ | 0 | 0x2b |
| slti rt, rs, $imm^{16}$ | rt=(rs<imm?1:0) | 0xa | $rs^5$ | $rt^5$ | $imm^{16}$ | | |
| sltiu rt, rs, $imm^{16}$ | rt=(rs<imm?1:0) | 0xb | $rs^5$ | $rt^5$ | $imm^{16}$ | | |

| Instruction | Meaning | I-Type Format | | | |
|---|---|---|---|---|---|
| lb rt, $imm^{16}$(rs) | rt = MEM[rs+$imm^{16}$] | 0x20 | $rs^5$ | $rt^5$ | $imm^{16}$ |
| lh rt, $imm^{16}$(rs) | rt = MEM[rs+$imm^{16}$] | 0x21 | $rs^5$ | $rt^5$ | $imm^{16}$ |
| lw rt, $imm^{16}$(rs) | rt = MEM[rs+$imm^{16}$] | 0x23 | $rs^5$ | $rt^5$ | $imm^{16}$ |
| lbu rt, $imm^{16}$(rs) | rt = MEM[rs+$imm^{16}$] | 0x24 | $rs^5$ | $rt^5$ | $imm^{16}$ |
| lhu rt, $imm^{16}$(rs) | rt = MEM[rs+$imm^{16}$] | 0x25 | $rs^5$ | $rt^5$ | $imm^{16}$ |
| sb rt, $imm^{16}$(rs) | MEM[rs+$imm^{16}$] = rt | 0x28 | $rs^5$ | $rt^5$ | $imm^{16}$ |
| sh rt, $imm^{16}$(rs) | MEM[rs+$imm^{16}$] = rt | 0x29 | $rs^5$ | $rt^5$ | $imm^{16}$ |
| sw rt, $imm^{16}$(rs) | MEM[rs+$imm^{16}$] = rt | 0x2b | $rs^5$ | $rt^5$ | $imm^{16}$ |

| Instruction | Meaning | Format | | | | | |
|---|---|---|---|---|---|---|---|
| jal label | \$31=PC+4, jump | $op^6 = 3$ | $imm^{26}$ | | | | |
| jr Rs | PC = Rs | $op^6 = 0$ | $rs^5$ | 0 | 0 | 0 | 8 |
| jalr Rd, Rs | Rd=PC+4, PC=Rs | $op^6 = 0$ | $rs^5$ | 0 | $rd^5$ | 0 | 9 |

| Instruction | Meaning | Format | | | | | |
|---|---|---|---|---|---|---|---|
| mult Rs, Rt | Hi, Lo = Rs × Rt | $op^6 = 0$ | $Rs^5$ | $Rt^5$ | 0 | 0 | 0x18 |
| multu Rs, Rt | Hi, Lo = Rs × Rt | $op^6 = 0$ | $Rs^5$ | $Rt^5$ | 0 | 0 | 0x19 |
| mul Rd, Rs, Rt | Rd = Rs × Rt | 0x1c | $Rs^5$ | $Rt^5$ | $Rd^5$ | 0 | 0x02 |
| div Rs, Rt | Hi, Lo = Rs / Rt | $op^6 = 0$ | $Rs^5$ | $Rt^5$ | 0 | 0 | 0x1a |
| divu Rs, Rt | Hi, Lo = Rs / Rt | $op^6 = 0$ | $Rs^5$ | $Rt^5$ | 0 | 0 | 0x1b |
| mfhi Rd | Rd = Hi | $op^6 = 0$ | 0 | 0 | $Rd^5$ | 0 | 0x10 |
| mflo Rd | Rd = Lo | $op^6 = 0$ | 0 | 0 | $Rd^5$ | 0 | 0x12 |