King Fahd University of Petroleum and Minerals
College of Computer Science and Engineering
Computer Engineering Department

ICS 233: COMPUTER ARCHITECTURE & ASSEMBLY LANGUAGE
Term 142 (Spring 2014-2015)
Major Exam II
Sunday April 19, 2015

Time: 150 minutes, Total Pages: 13

Name:________________________________ ID:___________________ Section: _______

Notes:

- Do not open the exam book until instructed
- Answer all questions
- All steps must be shown
- Any assumptions made must be clearly stated

Question	Max Points	Score
Q1	12	
Q2	15	
Q3	18	
Q4	30	
Total	75	

Dr. Aiman El-Maleh
Dr. Mayez Al-Mouhamed

[12 Points]

(Q1) Write a procedure, **GCD**, that receives two positive numbers in $a0 and $a1 and returns their greatest common divisor in register $v0. It is required that the procedure **preserves the content of all used registers** according to the MIPS programming convention by saving them and restoring them on the stack. Then, write a program that reads two positive numbers from the user and displays the **GCD** obtained by the above procedure. The pseudo code of the GCD procedure is given below:

```
int gcd(int m, int n) {
   if ((m % n) == 0)
      return n;
   else
      return gcd(n, m % n);
}
```

A **sample execution** of the program is:

Enter two numbers:
90
24
GCD is: 6

A summary of syscall services you can use is given below:

Service	$v0	Arguments / Result
Print Integer	1	$a0 = integer value to print
Print String	4	$a0 = address of null-terminated string
Read Integer	5	$v0 = integer read
Exit	10	

```
################# Data segment #####################
.data

msg1: .asciiz "Enter two numbers:\n"
msg2: .asciiz "GCD is: "

################# Code segment #####################
.text
.globl main
main:   # main program

li $v0, 4
la $a0, msg1
syscall
li $v0, 5
syscall
```

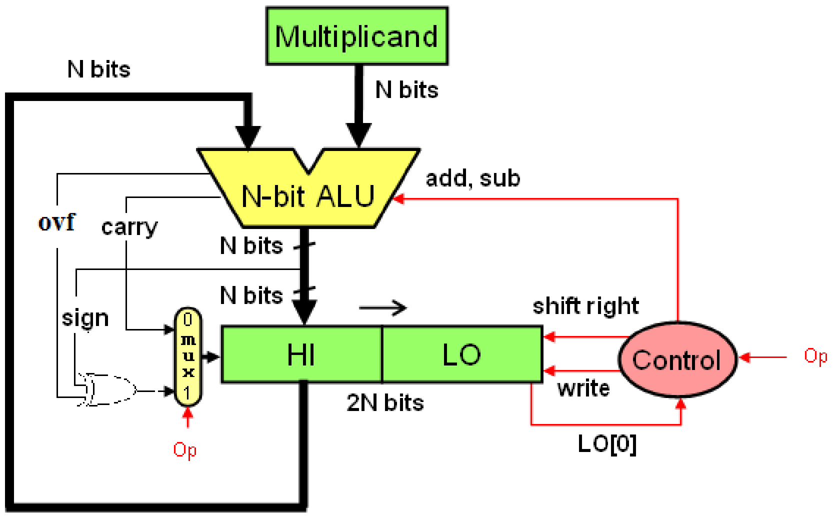
```
move $a0, $v0

li $v0, 5
syscall
move $a1, $v0

jal GCD
move $t0, $v0

li $v0, 4
la $a0, msg2
syscall

move $a0, $t0
li $v0, 1
syscall


li $v0, 10          # Exit program
syscall

GCD:
divu $a0, $a1
mfhi $t0
bne $t0, $0, Else
move $v0, $a1
jr $ra
Else:
addi $sp, $sp, -4
sw $ra, ($sp)
move $a0, $a1
move $a1, $t0
jal GCD
lw $ra, ($sp)
addi $sp, $sp, 4
jr $ra
```

[15 Points]

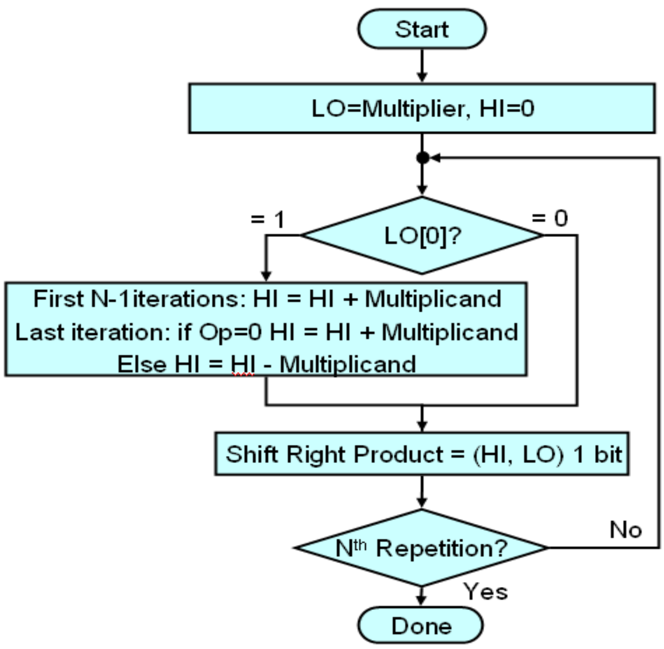
(Q2)

(i) You are required to design a circuit that can be used to perform either signed or unsigned multiplication of two 32-bit operands A and B depending on an input signal OP. When OP=0, the circuit will perform unsigned multiplication. Otherwise, it will perform signed multiplication.

a. Given the circuit below, which performs unsigned multiplication, add the necessary changes so that it can be used for performing either signed or unsigned multiplication depending on OP.



b. Show the algorithm that will be used along with your circuit in (a) for performing either signed or unsigned multiplication depending on OP.



(ii) Given that **Multiplicand=1010** and **Multiplier=1011,** using the **signed multiplication** hardware, show the **signed** multiplication of **Multiplicand** by **Multiplier**. The result of the multiplication should be an 8 bit **signed** number in HI and LO registers. Show the steps of your work.

Iteration		**Multiplicand**	**Sign**	**Product = HI,LO**
0	Initialize (LO = Multiplier)	1010		0000 101**1**
1	LO[0] = 1 => ADD		1	1010 1011
	Shift Product = (HI, LO) right 1 bit	1010		1101 010**1**
2	LO[0] = 1 => ADD		1	0111 0101
	Shift Product = (HI, LO) right 1 bit	1010		1011 101**0**
3	LO[0] = 0 => Do nothing		1	1011 1010
	Shift Product = (HI, LO) right 1 bit	1010		1101 110**1**
4	LO[0] = 1 => SUB		0	0011 1101
	Shift Product = (HI, LO) right 1 bit			**0001 1110**

(iii) Given that **Dividend=1011** and **Divisor=0010**, Using the **unsigned division** hardware, show the **unsigned** division of **Dividend** by **Divisor**. The result of division should be stored in the Remainder and Quotient registers. Show the steps of your work.

Iteration		**Remainder (HI)**	**Quotient (LO)**	**Divisor**	**Difference**
0	Initialize	0000	1011	0010	
1	1: SLL, Difference	0001	0110	0010	1111
	2: Diff < 0 => Do Nothing	0001	0110	0010	
2	1: SLL, Difference	0010	1100	0010	0000
	2: Rem = Diff, set lsb Quotient	0000	110**1**	0010	
3	1: SLL, Difference	0001	1010	0010	1111
	2: Diff < 0 => Do Nothing	0001	1010	0010	
4	1: SLL, Difference	0011	0100	0010	0001
	2: Rem = Diff, set lsb Quotient	**0001**	**0101**	0010	

[18 Points]

(Q3)

1. [3 Points] What is the decimal value of following single precision float:

[1, 0111 1000, 0111 0000 0000 0000 0000 000]

= - $(1.0111000000000000...0)_2 * 2^{(120-127)}$ = - $(1.0111000000000000...0)_2 * 2^{-7}$
= -0.011230469

2. [3 Points] Find the normalized single precision representation of –21.625.

$21.625=(10101.101)_2 = (1.0101101)_2 * 2^4$
Exp. = 4 +127=131
<u>Single precision binary representation</u>:

1100 0001 1010 1101 0000 0000 0000 0000

3. [2 Points] Find the smallest positive normalized float for single precision.

- ✧ Exponent – bias = 1 – 127 = –126 (smallest exponent for SP)
- ✧ Significand = $(1.000 \ldots 0)_2 = 1$
- ✧ Value in decimal = $1 \times 2^{-126} = 1.17549 \ldots \times 10^{-38}$

4. [2 Points] Find the largest positive denormalized float for single precision.

- ✧ Exponent = –126
- ✧ Significand = $(0.111 \ldots 1)_2 \approx 1$
- ✧ Value in decimal $\approx 1 \times 2^{-126}$ Exact value $=1.1754942 \times 10^{-38}$

5. [3 Points] Give the representation of Zero, -infinity, and NAN for single precision:

+Zero: [0 , 00000000 , 000 0000 0000 0000 0000 0000]

-infinity: [1 , 11111111 , 000 0000 0000 0000 0000 0000]

NAN: [0 or 1, 11111111, any non-zero value]

6. [5 Points] Find the normalized difference between A and B by using rounding to nearest even. Perform the operation using **guard**, **round** and **sticky** bits

A= + 1.000 0001 0000 1111 1000 0001 $\times 2^4$
B = +1.000 0111 1100 0000 1010 0000 $\times 2^{-3}$

```
       1.000 0001 0000 1111 1000 0001 000 x 2^4
 -     1.000 0111 1100 0000 1010 0000 000 x 2^-3
-------------------------------------------------------------
      01.000 0001 0000 1111 1000 0001 000 x 2^4
 -    00.000 0001 0000 1111 1000 0001 010 x 2^4  (align)
-------------------------------------------------------------
      01.000 0001 0000 1111 1000 0001 000 x 2^4
 +    11.111 1110 1111 0000 0111 1110 110 x 2^4 (2's complement)
-------------------------------------------------------------
      00.111 1111 1111 1111 1111 1111 110 x 2^4
 = +   0.111 1111 1111 1111 1111 1111 110 x 2^4
 = +   1.111 1111 1111 1111 1111 1111 100 x 2^3 (normalize)
 = + 10.000 0000 0000 0000 0000 0000      x 2^3 (round)
 = +  1.000 0000 0000 0000 0000 0000      x 2^4 (renormalize)
```

[30 Points]

(Q4)

The instruction set (ISA_x) of a 32-bit RISC processor with 32 registers is defined as follows:

a. R-type: register-d = register-t op register-s, where each register denotes one of the 32-bit user registers, and op denotes an arithmetic or logic operation.

INTRUCTION	OPCODE
Add	0000
Sub	0001
Or	0010
And	0011

b. I-type: register-t = register-s op imm16, where imm16 denotes a 16-bit immediate data.

INTRUCTION	OPCODE
Addi	0100
Subi	0101
Ori	0110
Andi	0111

c. LS-type: load using register-t = M(register-s) and store using M(register-s) = register-t, where M is the data memory and register-s contains the base address.

INTRUCTION	OPCODE
Lw	1000
Sw	1001

d. I/O-type (I/O) or input and output defined as: (1) read the I/O Unit (IOU) addressed by register-s and store data in destination register-t = IO(register-s), and (2) write register-t data to IOU at address register-s as IO(register-s) = register-t.

INTRUCTION	OPCODE
In	1010
Out	1011

The IOU is a read/write unit such as reading from keyboard and writing to CRT. The interface of the CPU with the IOU is similar to the interface to data memory, i.e., it is a box with input address, data-in and data-out and other control signals.

e. CB-type (CB): conditional branch (CB) using PC = PC+4+ Ext[Imm16]*4 if condition is True or PC= PC+4 if Condition is False. The condition checked is whether register-s and register-t are either equal or not equal. Note that the ALU has a Zero flag (Z).

INTRUCTION	OPCODE
BEQ	1100
BNE	1101

f. J-type (Jump): Jump using PC = [$(PC+4)_{31-28}$, (Imm26),00], i.e. concatenation

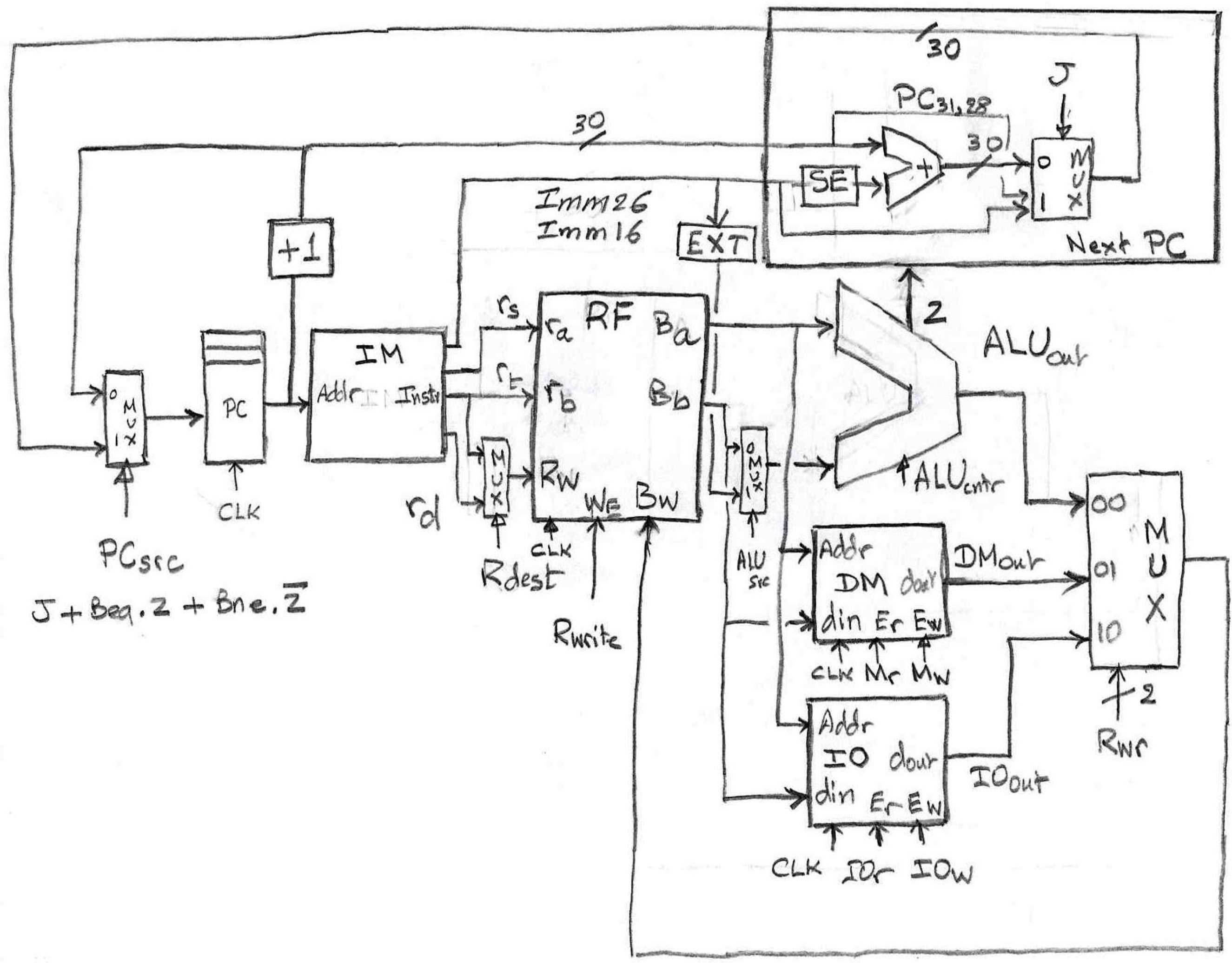
INTRUCTION	OPCODE
J	1110

The objective of this question is to design a Single Cycle Datapath (SCD). Assume the availability of the following components: PC, IM, RF, ALU, I/O, and DM.

1. [16 points] Give the drawing of the SCD Design and answer each of the following questions:

   a. Show the needed components for SCD with controls, clocking (if any), and write enables (if any),
   b. Explain why a component may require clocking and/or write enable! When the write should take place and why!,
   c. Show all needed multiplexers and their controls,
   d. Show the PC with all possible target values.

Solution



a. The RF, DM, and IO must have Write Enable and be clocked because their write should take place at the end of the single cycle. (Directly or indirectly on the design: 4 points)
b. In total we need 4 2-to-1 Muxes and one 4-to-1 Mux (same as above: 4 points).
c. The PC values can be PC+4 (R, I-R, L/S, and Beq/Bne if not taken),
   PC + E(Imm16) in the case of Beq/Bne if taken, or
   [PC31-28, Imm26,00] for the J.

   Same as above + details of PCnext (4 points)
d. Correct connectivity details (Addr, din, dout, WE, etc) of M and IO (4 points).
e. Note: The above SCD arrangement may shorten the average execution time for R and IR types because the ALU, DM, and I/O are parallel. The exception is the L/S types, for which 2 instructions will be required instead of one if a base address is added to an offset. In some cases we may avoid the use of offset if we directly increment a register address.

2. [8 points] List in the following table all the controls needed for your SCD. Fill in the Table for all value of control corresponding to each listed instruction.

Solution

Type	Instr.	OPCODE $X_3\ X_2\ X_1\ X_0$	Rdest	Rwrite	Ext	ALUsrc	ALUop	Beq	Bne	J	Mrd	Mwt	IOrd	IOwt	Rwr
R	Add	0000	1	1	1	0	00	0	0	0	0	0	0	0	00
	Sub	0001	1	1	1	0	01	0	0	0	0	0	0	0	00
	Or	0010	1	1	0	0	10	0	0	0	0	0	0	0	00
	And	0011	1	1	0	0	11	0	0	0	0	0	0	0	00
I	Addi	0100	0	1	1	1	00	0	0	0	0	0	0	0	00
	Subi	0101	0	1	1	1	01	0	0	0	0	0	0	0	00
	Ori	0110	0	1	0	1	10	0	0	0	0	0	0	0	00
	Andi	0111	0	1	0	1	11	0	0	0	0	0	0	0	00
L/S	Lw	1000	0	1	x	x	xx	0	0	0	1	0	0	0	01
	Sw	1001	x	0	x	x	xx	0	0	0	0	1	0	0	xx
I/O	In	1010	0	1	x	x	xx	0	0	0	0	0	1	0	10
	Out	1011	x	0	x	x	xx	0	0	0	0	0	0	1	xx
CB	Beq	1100	x	0	x	0	01	1	0	0	0	0	0	0	xx
	Bne	1101	x	0	x	0	01	0	1	0	0	0	0	0	xx
J	J	1110	x	0	x	x	xx	0	0	1	0	0	0	0	xx

Note: (1) Controls must appear on the design, (2) Beq, Bne, J, Mrd, Mw, IOr and IOw are given less weight, and (3) ALUop must be expressed as function of OPCODE.

3. [6 Points] Find the **simplest combinational logic function** for each of the SCD control as function of Opcode.

Solution

Rdest = x3'x2'
Rwrite= X3'+ x3x2'x0'
Ext = x3'x1'
ALUsrc = x2
ALUop = (x1, x0+x3x2)
Beq =x3x2x1'x0'
Bne = x3x2x1'x0
J = x3x2x1x0'
Mrd = x3x2'x1'x0'
Mwt = x3x2'x1'x0
IOrd = x3x2'x1x0'
IOwt = x3x2'x1x0
Rwr = (x3x2'x1, x3x2'x1')

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