Last Name: _____ First Name: _____

CruzID: _____

Final Exam

CMPE 012: Computer Systems and Assembly Language

University of California, Santa Cruz

Fall 2018

DO NOT BEGIN UNTIL YOU ARE TOLD TO DO SO.

This exam is closed book and closed notes. Only 4-function calculators are permitted. Answers must be marked on the Scantron form to be graded. All work must be written on the exam.

Write your first name, last name, and CruzID on this page. Write your CruzID on all subsequent pages of the exam. On the Scantron form, bubble in your name, student ID number, and test form. The test form can be found in the footer of all subsequent pages of the exam.

You must sit in your assigned seat. Keep your student or government issued ID on your desk. Brimmed hats must be removed or turned around backwards. Only unmarked water bottles are permitted. Backpacks must be placed at the front of the room. Your cell phone must be on a setting where it will not make noise or vibrate.

All questions are multiple choice. Some questions have more than one correct answer. You must mark all correct answers to receive credit for a question.

You have 120 minutes to complete this exam.

CruzID: _____ @ucsc.edu

CMPE 12 Final Exam Version A

Fall 2018

Bits

1. What is the size of a byte?
   - ◯ A. 32 bits
   - ◯ B. 10 bits
   - ◯ C. 4 bits
   - ◯ D. 6 bits
   - ◯ E. 8 bits
2. What is the size of a nybble?
   - ◯ A. 2 bits
   - ◯ B. 64 bits
   - ◯ C. 32 bits
   - ◯ D. 4 bits
   - ◯ E. 8 bits
3. What is the size of a word in MIPS?
   - ◯ A. 8 bits
   - ◯ B. 4 bits
   - ◯ C. 32 bits
   - ◯ D. 16 bits
   - ◯ E. 64 bits

Binary Addition

4. Which computations have overflow? Assume numbers are 16-bit two's complement. Select all that apply.
   - ◯ A. `0x76F9 + 0x801A = 0xF713`
   - ◯ B. `0xD02A + 0x57D9 = 0x2803`
   - ◯ C. `0xF02B + 0x57D9 = 0x5864`
   - ◯ D. `0x0308 + 0x1198 = 0x14A0`
   - ◯ E. `0x0308 + 0x1198 = 0x14A0`
5. Which computations have carry out but no overflow? Assume numbers are 8-bit two's complement. Select all that apply.
   - ◯ A. `0x2C + 0x2D`
   - ◯ B. `0xED + 0x09`
   - ◯ C. `0x0A + 0xFD`
   - ◯ D. `0xD9 + 0x5C`
   - ◯ E. `0x7F + 0x01`

CruzID: _____ @ucsc.edu

Data Representation

6. Decode the following ASCII string.

```
0x43 0x45 0x31 0x32 0x20 0x69 0x73 0x20 0x6c 0x6f 0x76 0x65
0x2e 0x20 0x43 0x45 0x31 0x32 0x20 0x69 0x73 0x20 0x6c 0x69
0x66 0x65 0x2e
```

   - ◯ A. Lab 4 was easy.
   - ◯ B. You're a wizard, Harry.
   - ◯ C. CE12 is love. CE12 is life.
   - ◯ D. CE12 was the best class ever.
   - ◯ E. No, I am your father!

7. Which IEEE 754 Single Precision floating point numbers are additive inverses of each other? (Select two)
   - ◯ A. `0x40400000`
   - ◯ B. `0xC0400000`
   - ◯ C. `0x26500000`
   - ◯ D. `0xF4500000`
   - ◯ E. `0x354FFFFF`

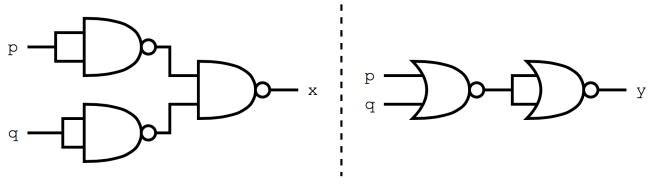
8. Convert this 8-bit 2's complement number to decimal: 11010110
   - ◯ A. -44
   - ◯ B. 212
   - ◯ C. -40
   - ◯ D. -42
   - ◯ E. 213

9. Which IEEE 754 Single Precision floating point number is furthest from zero?
   - ◯ A. `0x42903333`
   - ◯ B. `0xC3018000`
   - ◯ C. `0xC2366666`
   - ◯ D. `0x425A6666`
   - ◯ E. `0x4377999A`

10. Convert this base 15 number into base 9: 42
    - ◯ A. 62
    - ◯ B. 46
    - ◯ C. 28
    - ◯ D. 68
    - ◯ E. 38

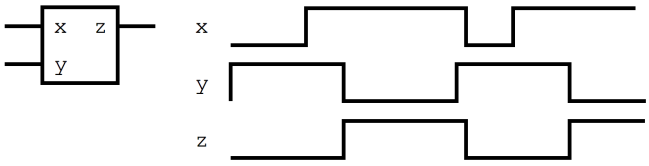
Logic

11. Given the logic circuit below, x and y are logically equivalent.



    - ◯ A. False
    - ◯ B. True

12. What device does this timing diagram represent:



- ◯ A. S-R latch, active high
- ◯ B. S-R latch, active low
- ◯ C. D latch
- ◯ D. D-R latch
- ◯ E. D flip flop, edge triggered

13. Select all expressions equivalent to $\bar{A} \cdot B + A \cdot B$
- ◯ A. A
- ◯ B. B
- ◯ C. $A \oplus B$
- ◯ D. $\overline{A \oplus B}$
- ◯ E. $(\bar{A} + B) \cdot (A + B)$

Addressability

Assume a 4MB memory space with 4096 memory locations i.e. addresses.

14. What is the addressability of this memory space? i.e. How many bytes are stored at each memory location?
- ◯ A. 1024
- ◯ B. 9
- ◯ C. 512
- ◯ D. 4096
- ◯ E. 10

15. How many bits are needed to represent the address?
- ◯ A. 12
- ◯ B. 9
- ◯ C. 34
- ◯ D. 36
- ◯ E. 22

MIPS

16. Translate the following Java statement into MIPS assembly code. Assume that x, y, z, q are stored in registers `$s1`, `$s2`, `$s3`, and `$s4` respectively.

$$x = x - y + z + q$$

○ A.
```
add $t0 $s1 $s2
sub $t0 $s2 $s3
add $s1 $t0 $s4
```

○ B.
```
add $s1 $s1 $s3
sub $s1 $s1 $s2
add $s1 $s1 $s4
```

○ C.
```
add $s3 $s3 $s4
add $s2 $s2 $s3
sub $s1 $s1 $s2
```

○ D.
```
sub $t0 $s1 $s2
add $t0 $t0 $s3
add $s1 $t0 $s4
```

○ E.
```
sub $s1 $s1 $s2
add $s1 $s1 $s4
add $s1 $s1 $s3
```


17. How can we isolate bits 20:14 of $t0 ?

○ A. `ANDI $t0 $t0 0x1FC000`
○ B. `ORI  $t0 $t0 0x0001FC`
○ C. `XORI $t0 $t0 0x11FFC`
○ D. `AND  $t0 $t0<<2`
○ E. `AND  $t0 $t0 0x1FC000`

The next four questions refer to this code.

```
.data
first_word:  .asciiz "flux"
second_word: .asciiz "bunny"

.text
main:
la  $a0, first_word
jal PUSH_STRING

#push "bunny"
la  $a0, second_word
jal PUSH_STRING

#pop and print ten characters
li  $a1, 8
jal POP_AND_PRINT

#exit program:
li  $v0, 10
syscall

#Pushes a string onto the stack, followed by the length of the string
#input:  $a0 = address of string to push
PUSH_STRING:
  lb   $t1, ($a0)
  beqz $t1, EXIT_PUSH_STRING
  subi $sp, $sp, 4
  sw   $t1, ($sp)
  addi $a0, $a0, 1
  b    PUSH_STRING

  EXIT_PUSH_STRING:  jr $ra

#Pops a number of characters off the stack, and printing each one
#a1 = number of characters to pop and print
POP_AND_PRINT:
  lb   $a0, ($sp)
  addi $sp, $sp, 4
  li   $v0, 11  #print character
  syscall

  subi $a1, $a1, 1
  bnez $a1, POP_AND_PRINT

  jr   $ra
```

18. What is the value of the stack pointer after this code exits?
    - ◯ A. 0x2fdc
    - ◯ B. 0x2ffc
    - ◯ C. 0x2fd8
    - ◯ D. 0x3000
    - ◯ E. 0x2ff8

19. Which instructions are the "pop" operation?
    - ◯ A. `subi $sp, $sp, 4 / sw $t1, ($sp)`
    - ◯ B. `li $a1, 8 / jal POP_AND_PRINT`
    - ◯ C. `lb $a0, ($sp) / addi $sp, $sp, 4`
    - ◯ D. `subi $sp, $sp, 4 / addi $a0, $a0, 1`
    - ◯ E. `bnez $a1, POP_AND_PRINT / jr $ra`

20. What does this code print?
    - ◯ A. nnubxulf
    - ◯ B. xulfnnub
    - ◯ C. fluxbunn
    - ◯ D. ynnubxul
    - ◯ E. bunnyflu

21. What is the minimum value of $sp during execution of this code?
    - ◯ A. 0x2FF7
    - ◯ B. 0x2FDC
    - ◯ C. 0x2FF0
    - ◯ D. 0x3000
    - ◯ E. 0x2FD8

The following three questions refer to this code.

```
li  $t0, 1
li  $t1, 1

sll  $t0, $t0, 2
mul  $t1, $t1, 4
div  $t1, $t0
mfhi $t2
add  $t0, $t0, 3
div  $t0, $t1
mfhi $t3
mflo $t4
```

22. What is the value of $t2 after this code is executed?
    - ◯ A. 1
    - ◯ B. 0
    - ◯ C. 3
    - ◯ D. 4
    - ◯ E. 2

23. What is the value of $t4 after this code is executed?
    - ◯ A. 4
    - ◯ B. 2
    - ◯ C. 16
    - ◯ D. 0
    - ◯ E. 1

24. What is the value of $t3 after this code is executed?
    - ◯ A. 4
    - ◯ B. 0
    - ◯ C. 2
    - ◯ D. 3
    - ◯ E. 1

Data Movement

Consider a byte-addressable memory space with little endian memory storage. Assume $t1 contains the value 0x1000 and the initial memory state is as follows:

Address	Data
0x1005	0xFE
0x1004	0xCA
0x1003	0xAD
0x1002	0xBA
0x1001	0x00
0x1000	0x00

The following MIPS instructions are executed:

```
addiu $t0    $zero 0xCAFE
sw    $t0   ($t1)
lb    $t0  4($t1)
add   $t1,   $t1,  2
lh    $t3  0($t1)
```

25. What is the value of register $t1?
    - ◯ A. 0x00001004
    - ◯ B. 0xFFFF1000
    - ◯ C. 0x00001002
    - ◯ D. 0x00001000

26. What is the value of register $t3?
    - ◯ A. 0x0000DAAB
    - ◯ B. 0xFFFFBAAD
    - ◯ C. 0xFFFFADBA
    - ◯ D. 0x0000BAAD
    - ◯ E. 0x0000ADBA

27. What is the value of register $t0?
    - ◯ A. 0xCAFFFFFF
    - ◯ B. 0xCA000000
    - ◯ C. 0x000000CA
    - ◯ D. 0xFFFFFFCA
    - ◯ E. 0x000000AD

Arrays

The next three questions refer to the following code:

```
1   .data
2   space: .asciiz " "
3   array: .space  32
4
5   .text
6   main:
7       la $s0, array
8       li $t0, 0
9       li $t2, 65
10
11  loop:
12      bgt $t2, 75, printArray
13      add $t3, $s0, $t0
14
15      sb  $t2, 0($t3)
16      add $t0, $t0, 1
17      add $t2, $t2, 2
18      j loop
19
20  printArray:
21      la $a0, array
22      li $v0, 4
23      syscall
24
25      nop
26      li $v0, 10
27      syscall
```


28. What will be printed to the screen after the program completes execution?
    - ◯ A. ABCDEFGHIJKL
    - ◯ B. 65 66 67 68 69 70 71 72 73 74
    - ◯ C. ACEGIK
    - ◯ D. ABCDEFGHIJK
    - ◯ E. nothing will be printed

29. Assume you changed line 16 from

```
add $t0, $t0, 1
```

to

```
add $t0, $t0, 2
```

What would be printed to the screen after the program completes execution?

- ◯ A. none of the other answers
- ◯ B. ABCDEFGHIJK
- ◯ C. ACEGIK
- ◯ D. 65 67 69 71 73
- ◯ E. A

30. What will be stored in register $*s*0 right after execution of line 7?

- ◯ A. none of the other answers
- ◯ B. register $*s*0 will contain the value of 32
- ◯ C. register $*s*0 will be set to zero
- ◯ D. register $*s*0 will contain the address of 'array'

Decoding Instructions

31. The machine code `0x8D6E0101` represents which instruction?

- ◯ A. `LW   $t3 0x0101($t6)`
- ◯ B. `SUBU $zero $t3  $t6`
- ◯ C. `SUBU $t3 $t6 $zero`
- ◯ D. `LW   $t6 0x0101($t3)`
- ◯ E. `ANDI $t2   $t3  0x1010`

32. Which MIPS32 assembly language instruction assembles into the machine code `0x2A690003`?

- ◯ A. `addi $t1, $s3, 3`
- ◯ B. `sw $t1, 3($t2)`
- ◯ C. `slti $t1, $s3, 3`
- ◯ D. `xor $t1, $v1, $s3`

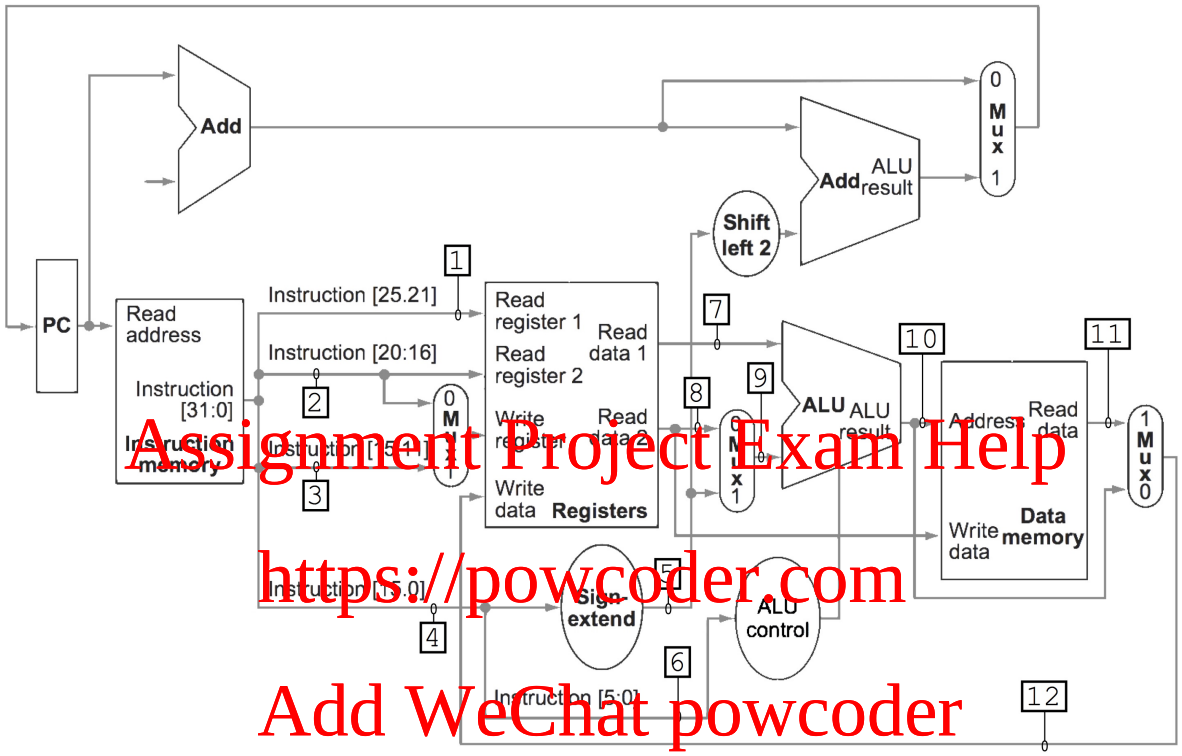
33. Encode the following instruction:

```
BNE $t1 $t0 0x10
```

- ◯ A. `0x21090010`
- ◯ B. `0x15280010`
- ◯ C. `0x35090010`
- ◯ D. `0x15280100`
- ◯ E. `0x21090010`

Data Path

Refer to this data path diagram for the next five questions.



The next two questions refer to the following MIPS32 instruction:

```
bne $t1, $t3, init_count_loop
```

This instruction is stored at memory address `0x3008`. The label "`init_count_loop`" refers to an instruction stored at memory address `0x308C`. Prior to executing this instruction, $t1 and $t3 contain `0x0E66` and `0xBABE`, respectively.

Note: A branch instruction uses the format
`bne rs rt label`

34. During execution, what signal is on line 4?
    - ◯ A. `0x3008`
    - ◯ B. `0x0020`
    - ◯ C. `0x0080`
    - ◯ D. `0x000B`
    - ◯ E. `0x305C`

35. During execution, what signal is on line 9?
    - ◯ A. `0x305C`
    - ◯ B. `0x0020`
    - ◯ C. `0xBABE`
    - ◯ D. `0x0E66`
    - ◯ E. `0x000B`

The next three problems refer to the following MIPS32 instruction:

```
sw $t3, -2($t1)
```


Again, $t1 and $t3 contain `0x0E66` and `0xBABE`, respectively.

36. What is the signal on line 12?
    - ◯ A. `0x0E64`
    - ◯ B. `0xBABC`
    - ◯ C. `0x0E66`
    - ◯ D. `0xBABE`
    - ◯ E. This line is not used in this instruction

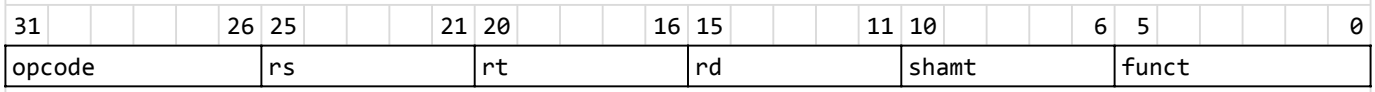
37. Assume the address range of the data memory in this MIPS processor is 0x00000000-0x00001000. Does this instruction execute without error?
    - ◯ A. yes
    - ◯ B. no

38. What is the signal on line 10?
    - ◯ A. `0x0E64`
    - ◯ B. `0xBABC`
    - ◯ C. `0x0E66`
    - ◯ D. `0xBABE`
    - ◯ E. This line is not used in this instruction

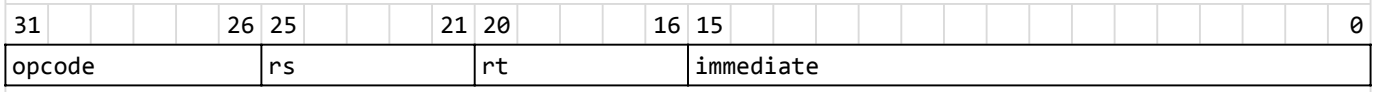
REG NAME	REG #	MNEMONIC	MEANING	TYPE	OPCODE	FUNCT
$zero	0	sll	Logical Shift Left	R	0x00	0x00
$at	1	srl	Logical Shift Right (0-extended)	R	0x00	0x02
$v0	2	sra	Arithmetic Shift Right (sign-extended)	R	0x00	0x03
$v1	3	jr	Jump to Address in Register	R	0x00	0x08
$a0	4	mfhi	Move from HI Register	R	0x00	0x10
$a1	5	mflo	Move from LO Register	R	0x00	0x12
$a2	6	mult	Multiply	R	0x00	0x18
$a3	7	multu	Unsigned Multiply	R	0x00	0x19
$t0	8	div	Divide	R	0x00	0x1A
$t1	9	divu	Unsigned Divide	R	0x00	0x1B
$t2	10	add	Add	R	0x00	0x20
$t3	11	addu	Add Unsigned	R	0x00	0x21
$t4	12	sub	Subtract	R	0x00	0x22
$t5	13	subu	Unsigned Subtract	R	0x00	0x23
$t6	14	and	Bitwise AND	R	0x00	0x24
$t7	15	or	Bitwise OR	R	0x00	0x25
$s0	16	xor	Bitwise XOR (Exclusive-OR)	R	0x00	0x26
$s1	17	nor	Bitwise NOR (NOT-OR)	R	0x00	0x27
$s2	18	slt	Set to 1 if Less Than	R	0x00	0x2A
$s3	19	sltu	Set to 1 if Less Than Unsigned	R	0x00	0x2B
$s4	20	j	Jump to Address	J	0x02	NA
$s5	21	jal	Jump and Link	J	0x03	NA
$s6	22	beq	Branch if Equal	I	0x04	NA
$s7	23	bne	Branch if Not Equal	I	0x05	NA
$t8	24	blez	Branch if Less Than or Equal to Zero	I	0x06	NA
$t9	25	addi	Add Immediate	I	0x08	NA
$k0	26	addiu	Add Unsigned Immediate	I	0x09	NA
$k1	27	slti	Set to 1 if Less Than Immediate	I	0x0A	NA
$gp	28	sltiu	Set to 1 if Less Than Unsigned Immediate	I	0x0B	NA
$sp	29	andi	Bitwise AND Immediate	I	0x0C	NA
		ori	Bitwise OR Immediate	I	0x0D	NA
		xori	Bitwise XOR (Exclusive-OR) Immediate	I	0x0E	NA
		lui	Load Upper Immediate	I	0x0F	NA
		mfc0	Move from Coprocessor 0	R	0x10	NA
		lb	Load Byte	I	0x20	NA
		lh	Load Halfword	I	0x21	NA
		lw	Load Word	I	0x23	NA
		lbu	Load Byte Unsigned	I	0x24	NA
		lhu	Load Halfword Unsigned	I	0x25	NA
		sb	Store Byte	I	0x28	NA
		sh	Store Halfword	I	0x29	NA
		sw	Store Word	I	0x2B	NA

MNEMONIC	MEANING	TYPE	OPCODE	FUNCT
add	Add	R	0x00	0x20
addi	Add Immediate	I	0x08	NA
addiu	Add Unsigned Immediate	I	0x09	NA
addu	Add Unsigned	R	0x00	0x21
and	Bitwise AND	R	0x00	0x24
andi	Bitwise AND Immediate	I	0x0C	NA
beq	Branch if Equal	I	0x04	NA
blez	Branch if Less Than or Equal to Zero	I	0x06	NA
bne	Branch if Not Equal	I	0x05	NA
div	Divide	R	0x00	0x1A
divu	Unsigned Divide	R	0x00	0x1B
j	Jump to Address	J	0x02	NA
jal	Jump and Link	J	0x03	NA
jr	Jump to Address in Register	R	0x00	0x08
lb	Load Byte	I	0x20	NA
lbu	Load Byte Unsigned	I	0x24	NA
lh	Load Halfword	I	0x21	NA
lhu	Load Halfword Unsigned	I	0x25	NA
lui	Load Upper Immediate	I	0x0F	NA
lw	Load Word	I	0x23	NA
mfc0	Move from Coprocessor 0	R	0x10	NA
mfhi	Move from HI Register	R	0x00	0x10
mflo	Move from LO Register	R	0x00	0x12
mult	Multiply	R	0x00	0x18
multu	Unsigned Multiply	R	0x00	0x19
nor	Bitwise NOR (NOT-OR)	R	0x00	0x27
or	Bitwise OR	R	0x00	0x25
ori	Bitwise OR Immediate	I	0x0D	NA
sb	Store Byte	I	0x28	NA
sh	Store Halfword	I	0x29	NA
sll	Logical Shift Left	R	0x00	0x00
slt	Set to 1 if Less Than	R	0x00	0x2A
slti	Set to 1 if Less Than Immediate	I	0x0A	NA
sltiu	Set to 1 if Less Than Unsigned Immediate	I	0x0B	NA
sltu	Set to 1 if Less Than Unsigned	R	0x00	0x2B
sra	Arithmetic Shift Right (sign-extended)	R	0x00	0x03
srl	Logical Shift Right (0-extended)	R	0x00	0x02
sub	Subtract	R	0x00	0x22
subu	Unsigned Subtract	R	0x00	0x23
sw	Store Word	I	0x2B	NA
xor	Bitwise XOR (Exclusive-OR)	R	0x00	0x26
xori	Bitwise XOR (Exclusive-OR) Immediate	I	0x0E	NA

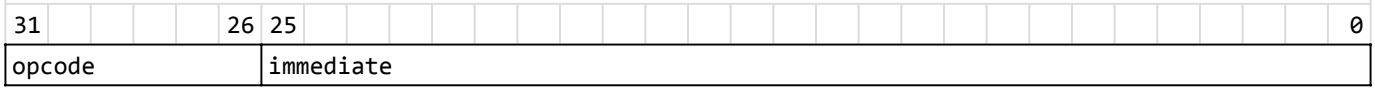
R Type: instr rd rs rt (arithmetic, logical)
instr rd rt shamt (shifts)



I Type: instr rt rs immediate (arithmetic, logical)
branch rs rt immediate (branches)
instr rt immediate(rs) (loads, stores)



J Type: j immediate (jumps)



ASCII CODE					ASCII CODE				
BIN	OCT	DEC	HEX	CHARACTER	BIN	OCT	DEC	HEX	CHARACTER
010 0000	40	32	20	space	101 0000	120	80	50	P
010 0001	41	33	21	!	101 0001	121	81	51	Q
010 0010	42	34	22	"	101 0010	122	82	52	R
010 0011	43	35	23	#	101 0011	123	83	53	S
010 0100	44	36	24	$	101 0100	124	84	54	T
010 0101	45	37	25	%	101 0101	125	85	55	U
010 0110	46	38	26	&	101 0110	126	86	56	V
010 0111	47	39	27	'	101 0111	127	87	57	W
010 1000	50	40	28	(	101 1000	130	88	58	X
010 1001	51	41	29	)	101 1001	131	89	59	Y
010 1010	52	42	2A	*	101 1010	132	90	5A	Z
010 1011	53	43	2B	+	101 1011	133	91	5B	[
010 1100	54	44	2C	,	101 1100	134	92	5C	\
010 1101	55	45	2D	-	101 1101	135	93	5D	]
010 1110	56	46	2E	.	101 1110	136	94	5E	^
010 1111	57	47	2F	/	101 1111	137	95	5F	_
011 0000	60	48	30	0	110 0000	140	96	60	`
011 0001	61	49	31	1	110 0001	141	97	61	a
011 0010	62	50	32	2	110 0010	142	98	62	b
011 0011	63	51	33	3	110 0011	143	99	63	c
011 0100	64	52	34	4	110 0100	144	100	64	d
011 0101	65	53	35	5	110 0101	145	101	65	e
011 0110	66	54	36	6	110 0110	146	102	66	f
011 0111	67	55	37	7	110 0111	147	103	67	g
011 1000	70	56	38	8	110 1000	150	104	68	h
011 1001	71	57	39	9	110 1001	151	105	69	i
011 1010	72	58	3A	:	110 1010	152	106	6A	j
011 1011	73	59	3B	;	110 1011	153	107	6B	k
011 1100	74	60	3C	<	110 1100	154	108	6C	l
011 1101	75	61	3D	=	110 1101	155	109	6D	m
011 1110	76	62	3E	>	110 1110	156	110	6E	n
011 1111	77	63	3F	?	110 1111	157	111	6F	o
100 0000	100	64	40	@	111 0000	160	112	70	p
100 0001	101	65	41	A	111 0001	161	113	71	q
100 0010	102	66	42	B	111 0010	162	114	72	r
100 0011	103	67	43	C	111 0011	163	115	73	s
100 0100	104	68	44	D	111 0100	164	116	74	t
100 0101	105	69	45	E	111 0101	165	117	75	u
100 0110	106	70	46	F	111 0110	166	118	76	v
100 0111	107	71	47	G	111 0111	167	119	77	w
100 1000	110	72	48	H	111 1000	170	120	78	x
100 1001	111	73	49	I	111 1001	171	121	79	y
100 1010	112	74	4A	J	111 1010	172	122	7A	z
100 1011	113	75	4B	K	111 1011	173	123	7B	{
100 1100	114	76	4C	L	111 1100	174	124	7C	|
100 1101	115	77	4D	M	111 1101	175	125	7D	}
100 1110	116	78	4E	N	111 1110	178	126	7E	~
100 1111	117	79	4F	O	111 1111	177	127	7F	DEL