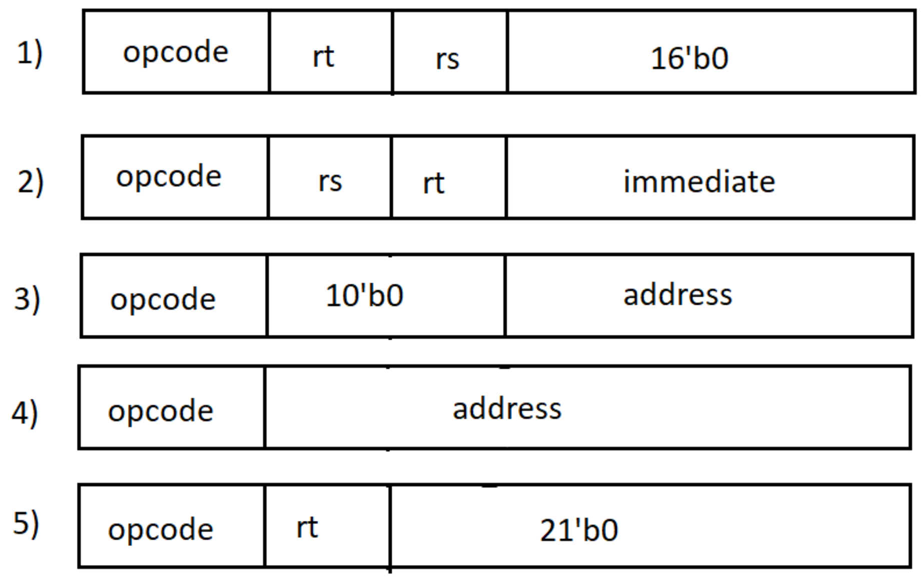
SINGLE CYCLE CPU REPORT

INSTRUCTION FORMAT:



Formats for Assembly code to be generated:

1. Format for arithmetic operations performed in ALU
   a) Opcode: 6bits
   b) rt (destination register) – 5bits
   c) rs (source register) – 5bits
2. Format for performing load store operations
   a) Opcode: 6bits
   b) rs (source register) – 5bit
   c) rt (destination register) – 5bit
   d) immediate – offset from address in source register from where data is loaded or stored – 16bits
3. Format for branch instructions with flag comparisons
   a) Opcode: 6bits
   b) Address: 16 bits address extended to 32’b address
4. Format for unconditional jumps
   a) Opcode: 6bits
   b) Address: 26bits address where:
      PC_next = address<<2 + leading 2bits of old PC address–
5. Format for jumps to addresses in registers
   a) Opcode: 6bits
   b) rt: optional register in which address is stored

INSTRUCTION ENCODING:

Instruction	**Opcode**
add	000000
comp	000001
addi	010000
compi	010001
and	000010
xor	000011
shll	000100
shrl	000101
shllv	000110
shrlv	000111
shra	001000
shrav	010001
lw	010010
sw	010011
b	101000
br	101001
bz	100000
bnz	100001
bcy	100010
bncy	100011
bs	100100
bns	100101
bv	100110
bnv	100111
Call	101010
ret	101011

REGISTER USAGE CONVENTIONS:

0-10: \$t0-\$t9

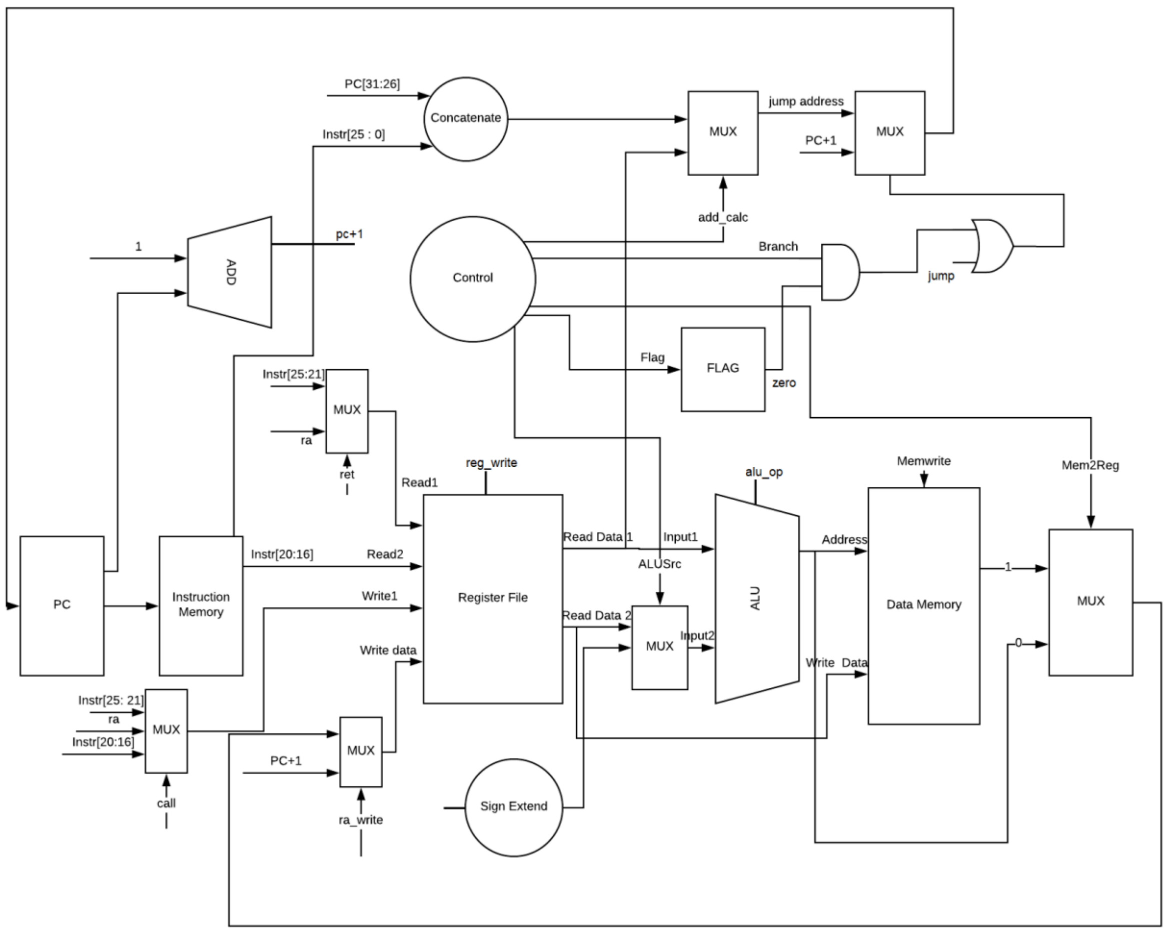
11-15: \$s0-\$4

28: \$gp

29: \$sp

30: Zero register

31: ra



Control Signals:

a) Memwrite: enables writing data into DataMemory when signal is high
b) Regwrite: enables writing data into registers when signal is high
c) MemtoReg: selects data ALU output or data memory output to be sent to be written in registers
d) Branch: selects branch on flag comparisons type address calculation when high
e) Jump: selects unconditional jump address when high
f) Add_calc: selects normal address when high and address from registers (for br/ret) when high
g) Aluop: (3 bits) selects which arithmetic operation to be performed
h) Call: (2 bits) selects between (rs, rt or ra) registers in which data is to be written
i) Ret: selects between (rt or rs) registers from which data 1 is to read.
j) Ra_write: selects to be written in normal registers or “ra” register
k) Alu_src: selects between read data 2 and sign extended output to be read into second input of ALU.
l) Flag: (3 bits) selects which flag comparison is being done

Bubble sort algorithm in our ISA:

```
lw t7, mem[n]

Bubble_sort:
 lw $t0, 0 //mem

loop1:
addi $t0,1
//bgt $t0,$t7 , end  // t7 =n
comp t8, t7   //i>n -> end
lw t9, 0
xor t9, t0 // t9=1
add $t9, $t8 //t9 = -3
bs label1         //  8
xor t9, $0
bnz end
//bns end

label1:
lw $t1, 0
add $t1, $t7  // j=n, t7 = n

loop2:
comp t10, t1 // //bge $t0, $t1, loop1   -> i>j go back to loop1
lw t11, 0
xor t11, t0  //here
add $t11, $t10
bns loop1
addi $t1,-1  // j =j-1 //ERROR
lw t4, 0 //added
xor t4, t1
lw t3, 0
xor t3, t4
addi $t3, -1   // t3= j-1
lw $t5,2($t4) // t5= a[j] ; MEM[2+J]
lw $t6,2($t3) // t6= a[j-1]; MEM[2+J-1]

swap:
comp t8, t6
lw t9, 0
xor t9, t5
add $t9, $t8
bns loop2

label2:
sw $t5,2($t3)
sw $t6,2($t4)
b loop2
End: 32'b0
```