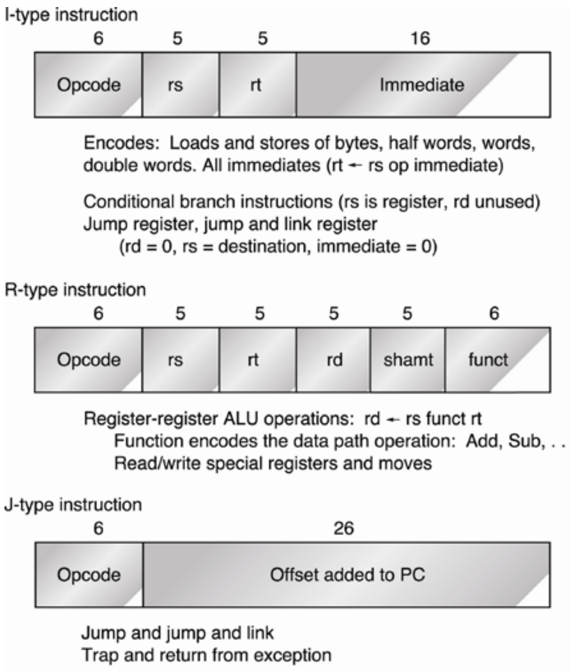
MIPS Subset for EGCP 4210 Projects



<u>Loads/Stores (8, 16, and 64-bit widths not required) – I type</u>

LW R1, 30(R2) - 32-bit integer load
    Mapping: rt =R1   rs=R2
SW R3, 500(R4) – 32-bit integer store
    Mapping: rt=R3   rs=R4

<u>Arithmetic – I-type (immediate) or R-type</u>

ADD, SUB, MUL, DIV, AND, OR, XOR
    e.g., ADD R3, R4, R5   (R3 = R4+R5)
ADDI, ANDI, ORI, XORI
    e.g., ADDI R2, R4, 4   (R2 = R4+4)
SLL, SRL, SRA
    e.g., SLL R4, R5, 3   (R4 = R5 << 3   or *8)

<u>Control-flow – I-type or J-type</u>

BEQ, BNE
    e.g., BEQ R4, R5, LABEL   (branch is R4=R5)
BLTZ, BLEZ, BGEZ, BGTZ

e.g., BLTZ R4, LABEL (branch R4 < 0)

J, JAL

e.g., J Label

JR, JALR

e.g, JALR R6 (jump and link to address in R6, relative to PC+4)

Note: JAL and JALR automatically store the current PC+4 in R31; to return from the subroutine, merely JR R31 (there is no explicit RETURN instruction). Note also that to allow recursion or nested subroutines, R31 would have to be saved prior to a second subroutine call. Return values will be placed in R1

NOP, HALT (neither has arguments) (done as J type)

<u>Floating-point ops (totally optional for project)</u>

LDC1 F4, 200(R4) - single load

SWC1 F6, 40(R2) - single store

ADD.S, SUB.S, MUL.S, DIV.S

MTC1, MFC1

CVT.W.S, CVT.S.W

C.**.S, where ** = LT, GT, LE, GE, EQ, NE

BC1T, BC1F

Assembly and Object Code Format

```
Begin Assembly
ADD R2, R1, R3
End Assembly
Begin Data 4000 16
45
56
4.3
0.34   (note: floats MUST contain a decimal point)
End Data
Begin Data 5000 100    (only two data sections allowed; second optional
End Data               (you don’t have to supply data if you don’t want to initialize
```

```
Org 0
45
65
32
11
15
56
76
87
Org 4000 16
23
24
56
54
76
78
76
88
Org 5000 100
```

Sample Program – computes sum of *n* numbers

```
-- A program to find the sum of a list of numbers
-- The program uses a subroutine to add 2 numbers, as a demo
-- It also sets up a stack frame, although not needed for this program
-- 4000 = # of nums to sum
-- 4004  = location for sum to be put
-- 4008 = beginning of array of nums
--
-- R20, R21 - parameter passing regs
-- R30 = SP
-- R31 = Ret Addr Reg
-- R3 = size of array, in bytes
-- R4 = Address of beginning of array (4008)
-- R5 = first address past array, for loop termination
-- R6 = current address being worked on (loop i variable)
-- R7 = sum
-- R8 = current array data value
--
Begin Assembly
-- Stack will be at Org5000 - R30 is SP
ADDI R30, R0, 5000
-- Data is at Org 4000
ADDI R4, R0, 4000
-- Load number of elements
LW R2, 0(R4)
-- Multiply this by 4, since each element is 4 bytes
SLL R3, R2, 2
-- R4 is address of beginning of array of numbers
ADDI R4, R4, 8
-- R5 now points to first address past array
ADD R5, R4, R3
-- initialize loop variable to first address (4008)
ADD R6, R4, R0
-- sum = 0
ADD R7, R0, R0
LABEL LoopStart
BEQ R6, R5, PostLoop
-- load current value
LW R8, 0(R6)
-- pass parameters (curr value and curr sum)
ADD R20, R8, R0
ADD R21, R7, R0
JAL AddThem
-- move sum from return reg to R7
```

```
ADD R7, R1, R0
-- increment address (by 4 bytes)
ADDI R6, R6, 4
J LoopStart
LABEL PostLoop
-- store answer
SW R7, -4(R4)
HALT
-- subroutine to add 2 numbers
LABEL AddThem
-- if doing recursion, must save R31
SW R31, 0(R30)
-- post incr the SP
ADDI R30, R30, 4
-- Since subroutine uses R5, must save
SW R5, 0(R30)
ADDI R30, R30, 4
-- get nums from parameter regs and sum
ADD R5, R20, R21
-- move result to return reg
ADD R1, R5, R0
-- now put stack back the way it was
-- and restore return address and R5
ADDI R30, R30, -4
LW R5, 0(R30)
ADDI R30, R30, -4
LW R31, 0(R30)
NOP
-- return from subroutine
JR R31
NOP
End Assembly
Begin Data 4000 44
10
0
23
71
33
5
93
82
34
13
111
23
End Data
```

Begin Data 5000 100
End Data