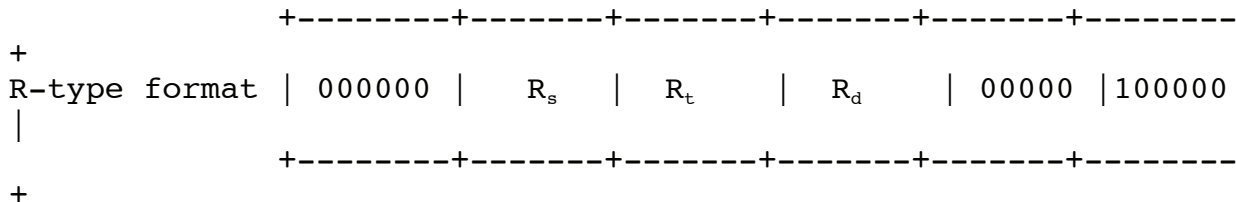


MIPS Instructions: 32-bit Core Subset

General notes:

- a. R_s , R_t , and R_d specify general purpose registers
- b. f_s , f_t , and f_d specify floating point registers
- c. C_d specifies coprocessor 0 registers
- d. **PC** (**P**rogram **C**ounter) specifies the instruction address register and contains address of instruction in execution
- e. Square brackets ([]) indicate "the contents of"
- f. **I** specifies part of instruction and its subscripts indicate bit positions of sub-fields
- g. || indicates concatenation of bit fields
- h. << indicates shift left
- i. >> indicates shift (arithmetic or logical)right
- j. Superscripts indicate repetition of a binary value
- k. $M\{i\}$ is contents of the word at the memory address i
- l. $m\{i\}$ is contents of the byte at the memory address i
- m. all integers are in 2's complement representation if not indicated as unsigned

1. Add Word: ADD



Effects: $R_d \leftarrow [R_s] + [R_t]$; $PC \leftarrow [PC] + 4$
(If overflow then exception)

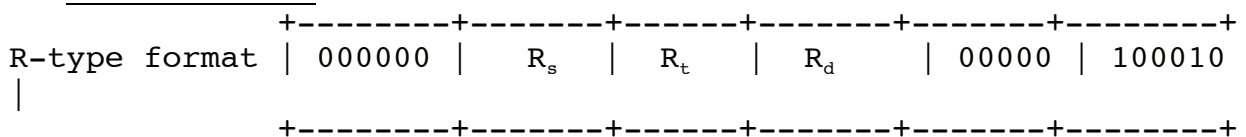
Assembly format: **ADD** R_d, R_s, R_t

2. Add Unsigned Word: ADDU

As **ADD** instruction, except:

- Funct=33_{dec}
- overflow ignored

3. Subtract Word: SUB



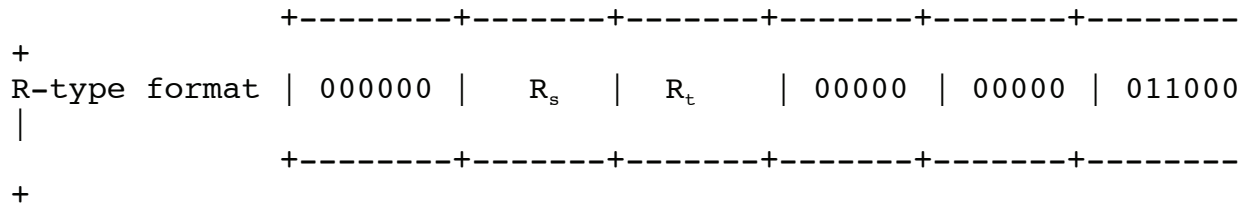
Effects: $R_d \leftarrow [R_s] - [R_t]$; $PC \leftarrow [PC] + 4$
(If overflow then exception)
 Assembly format: **SUB** R_d, R_s, R_t

4. Subtract Unsigned Word: SUBU

As **SUB** instruction, except:

- Funct=35_{dec}
- overflow ignored

5. Multiply Word: MULT



Effects: $Hi || Lo \leftarrow [R_s] * [R_t]$; $PC \leftarrow [PC] + 4$

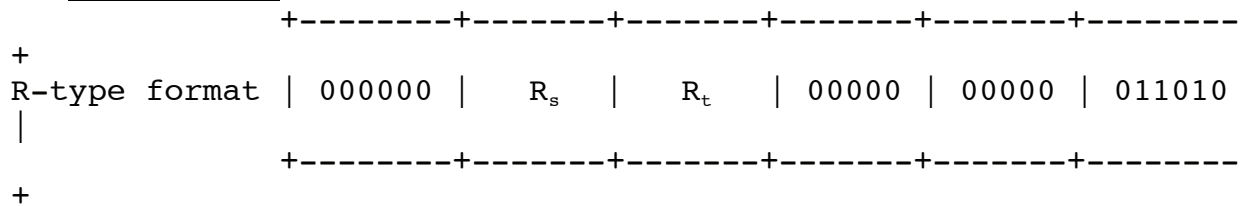
Assembly format: **MULT** R_s, R_t

6. Multiply Unsigned Word: MULTU

As **MULT** instruction, except:

- Funct = 25_{dec}
- contents of R_s and R_t are considered as unsigned integers

7. Divide Word: DIV



Effects: $Lo \leftarrow [R_s] / [R_t]$; $Hi \leftarrow [R_s] \bmod [R_t]$; $PC \leftarrow [PC] + 4$
 (Note: No exception on zero divide, and results is unpredictable)

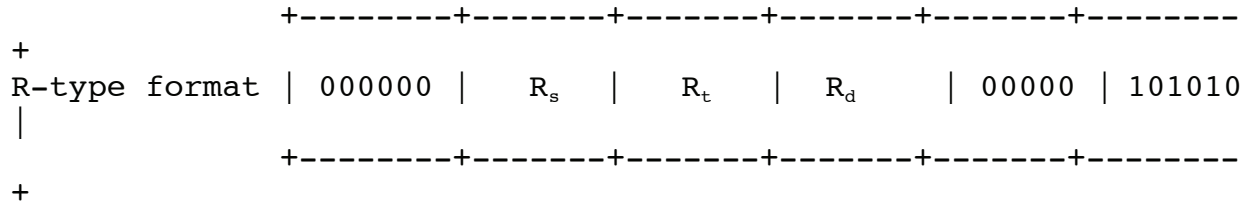
Assembly format: **DIV** R_s, R_t

8. Divide Unsigned Word: DIVU

As **DIV** instruction, except:

- Funct = 27_{dec}
- contents of R_s and R_t are considered as unsigned integers

9. Set on Less Than: SLT



Effects: if [R_s] < [R_t] then R_d <-- 0³¹ || 1 else R_d <-- 0³²;
PC <-- [PC] + 4

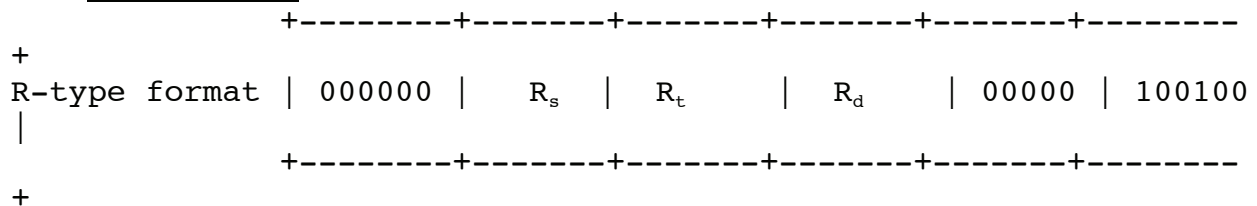
Assembly format: **SLT R_d,R_s,R_t**

10. Set on Less Than Unsigned: SLTU

As **SLT** instruction, except:

- Funct = 43_{dec}
- contents of R_s and R_t are considered as unsigned integers.

11. Logical AND: AND



Effects: R_d <-- [R_s] AND [R_t]; PC <-- [PC] + 4

Assembly format: **AND R_d,R_s,R_t**

12. Logical Or: OR

As **AND** instruction, except:

- Funct=37_{dec}
- or function performed instead of logical and

13. Logical Not Or: NOR

As **AND** instruction, except:

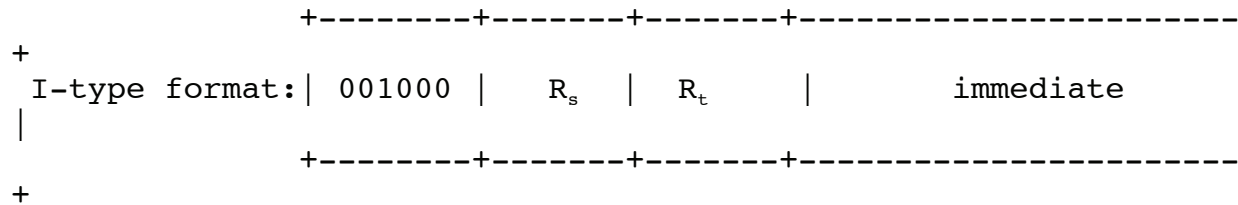
- Funct=39_{dec} for **NOR** instruction
- not or function performed instead of logical and

14. Logical Exclusive Or: XOR

As **AND** instruction, except:

- Funct=38_{dec}
- exclusive or function performed instead of logical and

15. Add Immediate Word: ADDI



Effects: $R_t \leftarrow [R_s] + ([I_{15}]^{16} || [I_{15..0}]); PC \leftarrow [PC] + 4$
(If overflow then

exception)

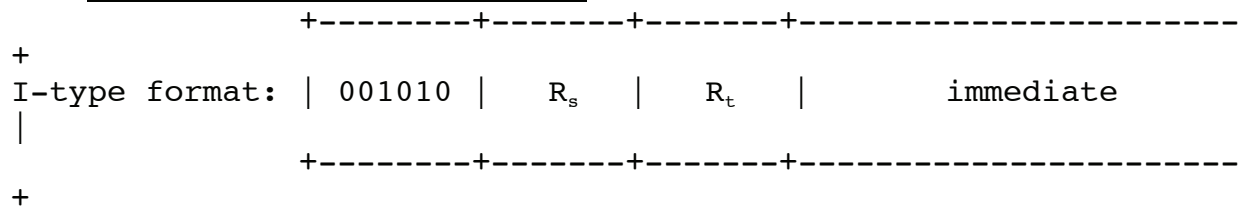
Assembly format: **ADDI R_t,R_s,immediate**

16. Add Immediate Unsigned Word: ADDIU

As **ADDI** instruction, except:

- Op-code=9_{dec}
- overflow ignored

17. Set on Less Than Immediate: SLTI



Effects: if $[R_s] < ([I_{15}]^{16} || [I_{15..0}])$ then $R_t \leftarrow 0^{31} || 1$ else $R_t \leftarrow 0^{32}$
 $PC \leftarrow [PC] +$

4

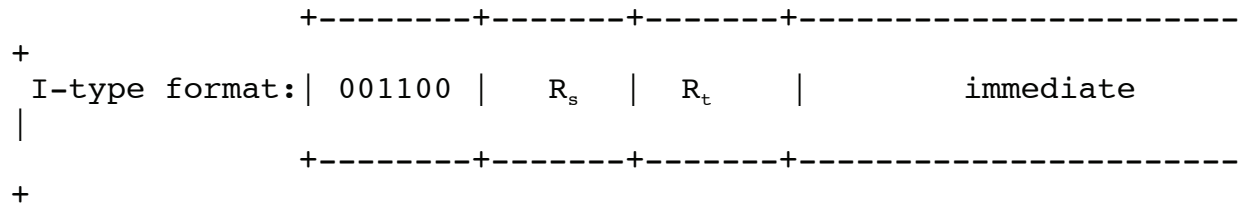
Assembly format: **SLTI R_t,R_s,immediate**

18. Set Less Than Immediate Unsigned: SLTIU

As **SLTI** instruction, except:

- Op-code = 11_{dec}
- contents in the comparison are considered as unsigned integers.

19. Logical And Immediate: ANDI



Effects: R_t <-- [R_s] AND (0¹⁶ || [I_{15..0}]); PC <-- [PC] + 4

Assembly format: **ANDI R_t,R_s,immediate**

20. Logical Or Immediate: ORI

As **ANDI** instruction, except:

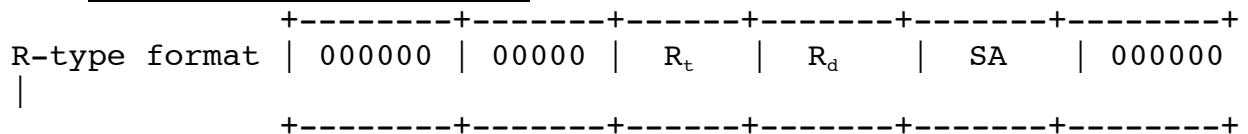
- Op-code=13_{dec} for **ORI** instruction
- or function performed instead of logical and

21. Exclusive Or Immediate: XORI

As **ANDI** instruction, except:

- Op-code=14_{dec} for **XORI** instruction
- exclusive or function performed instead of logical and

22. Shift Word Left Logical: SLL



Effects: R_d <-- [R_t] << SA; PC <-- [PC] + 4

Assembly format: **SLL R_d,R_t,SA**

23. Shift Word Left Logical Variable: SLLV

```

+-----+-----+-----+-----+-----+
R-type format | 000000 | Rs | Rt | Rd | SA | 000100
|
+-----+-----+-----+-----+-----+

```

Effects: $R_d \leftarrow [R_t] \ll [R_s]$; $PC \leftarrow [PC] + 4$

Assembly format: **SLLV R_d, R_t, R_s**

24. Shift Word Right Logical: SRL

```

+-----+-----+-----+-----+-----+
R-type format | 000000 | 00000 | Rt | Rd | SA | 000010
|
+-----+-----+-----+-----+-----+

```

Effects: $R_d \leftarrow [R_t] \gg SA$ (logical); $PC \leftarrow [PC] + 4$

Assembly format: **SRL R_d, R_t, SA**

25. Shift Word Right Logical Variable: SRLV

As **SRL** instruction, except:

- Funct=6_{dec}
- shift amount in R_s

26. Shift Word Right Arithmetic: SRA

```

+-----+-----+-----+-----+-----+
R-type format | 000000 | 00000 | Rt | Rd | SA | 000011
|
+-----+-----+-----+-----+-----+

```

Effects: $R_d \leftarrow [R_t] \gg SA$ (arithmetic); $PC \leftarrow [PC] + 4$

Assembly format: **SRA R_d, R_t, SA**

27. Shift Word Right Logical Variable: SRAV

As **SRA** instruction, except:

- Funct=7_{dec}
- shift amount in R_s

28. Load Word: **LW**

I-type format:	100011	R_s	R_t	offset
----------------	--------	-------	-------	--------

Effects: $R_t \leftarrow M\{[R_s] + [I_{15}]^{16} \parallel [I_{15..0}]\};$ PC $\leftarrow [PC] + 4$
(If illegal address then exception)

Assembly format: **LW R_t ,offset(R_s)**

29. Store Word: **SW**

I-type format:	101011	R_s	R_t	offset
----------------	--------	-------	-------	--------

Effects: $M\{[R_s] + [I_{15}]^{16} \parallel [I_{15..0}]\} \leftarrow [R_t];$ PC $\leftarrow [PC] + 4$
(If illegal address then

exception)

Assembly format: **SW R_t ,offset(R_s)**

30. Load Byte Unsigned: **LBU**

I-type format:	100100	R_s	R_t	offset
----------------	--------	-------	-------	--------

Effects: $R_t \leftarrow 0^{24} \parallel m\{[R_s] + [I_{15}]^{16} \parallel [I_{15..0}]\};$ PC $\leftarrow [PC] + 4$
(If illegal address then

exception)

Assembly format: **LBU R_t ,offset(R_s)**

31. Load Byte: **LB**

As **LBU** instruction, except:

- leftmost 24 bits of R_t set to a value of leftmost bit of byte
- Op-code = 32_{dec}

32. Load Halfword Unsigned: **LHU**

Similar as **LBU** instruction, except:

- 16 bits from memory loaded into R_t plus 16-bit zero-extend
- Op-code = 37_{dec}

33. Load Halfword: **LH**

Similar as **LB** instruction, except:

- 16 bits from memory loaded into R_t plus 16-bit sign-extend
- Op-code = 33_{dec}

34. Store Byte: **SB**

I-type format:

+	-----	+	-----	+	-----	+	-----	+	
I-type format:		101000		R_s		R_t		offset	
	+	-----	+	-----	+	-----	+	-----	+

Effects: $m\{[R_s] + [I_{15}]^{16} \mid [I_{15..0}]\} \leftarrow [R_t]_{7..0}; PC \leftarrow [PC] + 4$
(If illegal address then
exception)

Assembly format: **SB R_t ,offset(R_s)**

35. Store Halfword: **SH**

Similar as **SB** instruction, except:

- rightmost 16 bits from R_t stored into memory
- Op-code = 41_{dec}

36. Load Word to Floating Point: **LWC1**

I-type format:

+	-----	+	-----	+	-----	+	-----	+	
I-type format:		110001		R_s		f_t		offset	
	+	-----	+	-----	+	-----	+	-----	+

Effects: $f_t \leftarrow M\{[R_s] + [I_{15}]^{16} \mid [I_{15..0}]\}; PC \leftarrow [PC] + 4$
(If illegal address then
exception)

Assembly format: **LWC1 f_t ,offset(R_s)**

37. Store Word from Floating Point: **SWC1** instruction

I-type format:

+	-----	+	-----	+	-----	+	-----	+	
I-type format:		111001		R_s		f_t		offset	
	+	-----	+	-----	+	-----	+	-----	+

Effects: $M\{[R_s] + [I_{15}]^{16} \mid [I_{15..0}]\} \leftarrow [f_t]; PC \leftarrow [PC] + 4$
(If illegal address then
exception)

Assembly format: **swc1 f_t ,offset(R_s)**

38. Load Upper Immediate: LUI

```
+-----+-----+-----+-----+
I-type format: | 001111 | 00000 | Rt |           immediate           |
+-----+-----+-----+-----+
```

Effects: $R_t \leftarrow [I_{15..0}] \parallel 0^{16}$; $PC \leftarrow [PC] + 4$

Assembly format: **LUI R_t,immediate**

39. Branch on Equal: BEQ

```
+-----+-----+-----+-----+
I-type format: | 000100 | Rs | Rt |           offset           |
+-----+-----+-----+-----+
```

Effects: if $[R_s]=[R_t]$ then $PC \leftarrow [PC]+4 + ([I_{15}]^{14} \parallel [I_{15..0}] \parallel 0^2)$
else $PC \leftarrow [PC] + 4$

Assembly format: **BEQ R_s,R_t,offset**

40. Branch on Not Equal: BNE

```
+-----+-----+-----+-----+
+
I-type format: | 000101 | Rs | Rt |           offset           |
|
+-----+-----+-----+-----+
+
```

Effects: if $[R_s] \neq [R_t]$ then $PC \leftarrow [PC]+4 + ([I_{15}]^{14} \parallel [I_{15..0}] \parallel 0^2)$
else $PC \leftarrow [PC] + 4$

Assembly format: **BNE R_s,R_t,offset**

41. Branch on Less Than or Equal Zero: BLEZ

```
+-----+-----+-----+-----+
+
I-type format: | 000110 | Rs | 00000 |           offset           |
|
+-----+-----+-----+-----+
+
```

Effects: if $[R_s] \leq 0$ then $PC \leftarrow [PC] + 4 + ([I_{15}]^{14} \parallel [I_{15..0}] \parallel 0^2)$

else PC <-- [PC] + 4

Assembly format: **BLEZ** R_s ,offset

42. Branch on Greater Than Zero: BGTZ

As **BLEZ** instruction, except:

- branch if $[R_s] > 0$
- Op-code = 7_{dec}

43. Branch on Less Than Zero: BLTZ

As **BLEZ** instruction, except:

- branch if $[R_s] < 0$
- Op-code = 1_{dec}

44. Jump: J



Effects: PC <-- $[PC_{31..28}] || [I_{25..0}] || 0^2$

Assembly format: **J** jump_target

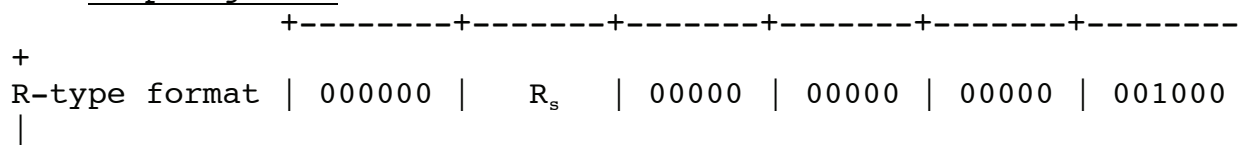
45. Jump and Link: JAL



Effects: $R_{31} <-- [PC] + 4$ PC <-- $[PC_{31..28}] || [I_{25..0}] || 0^2$

Assembly format: **JAL** jump_target

46. Jump Register: JR



```

+-----+-----+-----+-----+-----+-----+
+
Effects: PC <-- [Rs]
Assembly format: JR Rs

```

47. Jump and Link Register: JALR

```

+-----+-----+-----+-----+-----+-----+
+
R-type format | 000000 | Rs | 00000 | Rd | 00000 | 001001
|
+-----+-----+-----+-----+-----+-----+
+

```

Effects: R_d <-- [PC] + 4; PC <-- [R_s]
Assembly format: **JALR R_d, R_s**

48. Move From HI Register: MFHI

```

+-----+-----+-----+-----+-----+-----+
+
R-type format | 000000 | 00000 | 00000 | Rd | 00000 | 010000
|
+-----+-----+-----+-----+-----+-----+
+

```

Effects: R_d <-- [Hi]; PC <-- [PC] + 4
Assembly format: **MFHI R_d**

49. Move From LO Register: MFLO

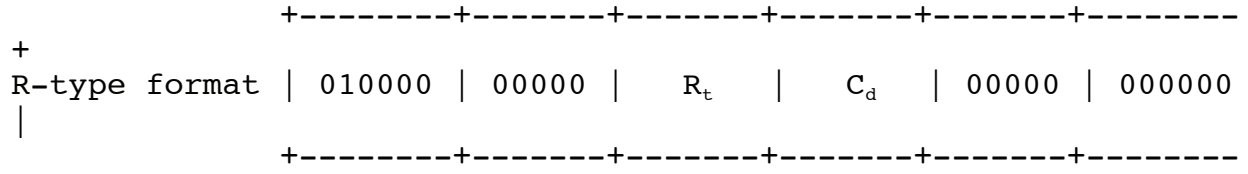
```

+-----+-----+-----+-----+-----+-----+
+
R-type format | 000000 | 00000 | 00000 | Rd | 00000 | 010010
|
+-----+-----+-----+-----+-----+-----+
+

```

Effects: R_d <-- [Lo]; PC <-- [PC] + 4
Assembly format: **MFLO R_d**

50. Move from Coprocessor 0 Register: MFC0

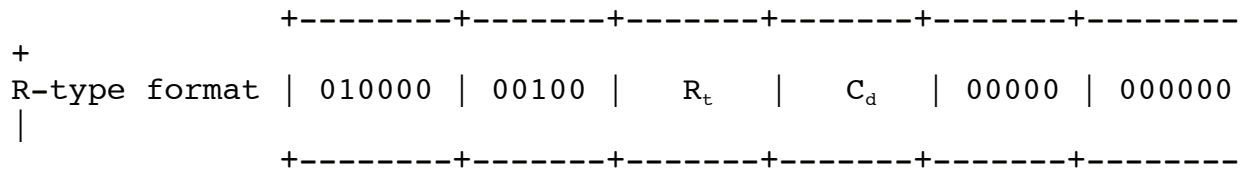


Effects: $R_t \leftarrow [C_d]$; $PC \leftarrow [PC] + 4$

Assembly format: **MFC0** R_t, C_d

C_{12} = Status Register; C_{13} = Cause Register; C_{14} = EPC Register;

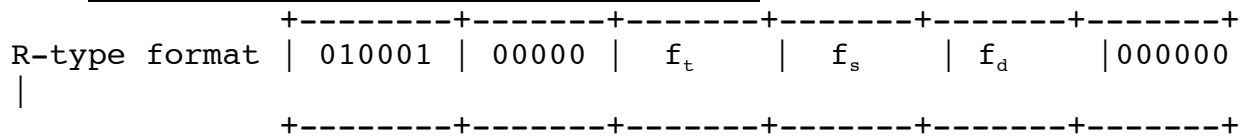
51. Move to Coprocessor 0 Register: MTC0



Effects: $C_d \leftarrow [R_t]$; $PC \leftarrow [PC] + 4$

Assembly format: **MTC0** C_d, R_t

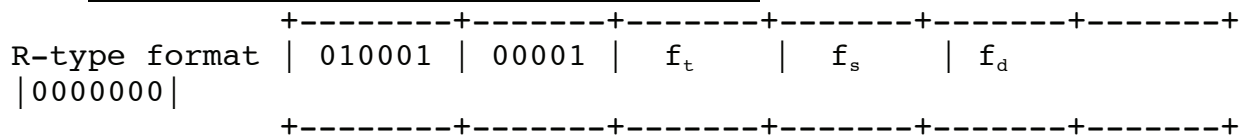
52. Floating Point Add Single Precision: ADD.S



Effects: $f_d \leftarrow [f_s] + [f_t]$; $PC \leftarrow [PC] + 4$
(Exception possible)

Assembly format: **ADD.S** f_d, f_s, f_t

53. Floating Point Add Double Precision: ADD.D



Effects: $f_d | f_{d+1} \leftarrow [f_s] | [f_{s+1}] + [f_t] | [f_{t+1}]$; $PC \leftarrow [PC] + 4$
(Exception possible)

Assembly format: **ADD.D** f_d, f_s, f_t

54. Floating Point Subtract Single Precision: SUB.S

Similar as ADD.S but with funct=1

55. Floating Point Subtract Double Precision: SUB.D

Similar as ADD.D but with funct=1

56. Floating Point Multiply Single Precision: MUL.S

Similar as add.s but with funct=2

57. Floating Point Multiply Double Precision: MUL.D

Similar as add.d but with funct=2

58. Floating Point Divide Single Precision: div.s

Similar as add.s but with funct=3

59. Floating Point Divide Double Precision: DIV.D instruction

Similar as add.d but with funct=3

60. Floating Point Move Single Precision: MOV.S

R-type format	010001	00000	00000	f_s	f_d	000110
---------------	--------	-------	-------	-------	-------	--------

Effects: $f_d \leftarrow [f_s]$; $PC \leftarrow [PC] + 4$

Assembly format: **MOV.S** f_d, f_s

61. Floating Point Move Double Precision: MOV.D

R-type format	010001	00001	00000	f_s	f_d	0000000
---------------	--------	-------	-------	-------	-------	---------

+-----+-----+-----+-----+-----+-----+

Effects: $f_d || f_{d+1} \leftarrow [f_s] || [f_{s+1}]$; $PC \leftarrow [PC] + 4$

Assembly format: **MOV.D** f_d, f_s, f_t

62. System Call: SYSCALL

+-----+-----+-----+-----+-----+-----+

R-type format | 000000 | XXXXX | XXXXX | XXXXX | XXXXX | 0011000 |

+-----+-----+-----+-----+-----+-----+

Effects: Exception

Assembly format: **SYSCALL**

Exception Handling

When a condition for any exception (overflow, illegal op-code, division by zero, etc.) occurs the following hardware exception processing is performed:

```

EPC <-- [PC]      /
Cause_Reg <--    | 028 || 1010  if illegal op-code (10)
                  | 028 || 1100  if integer overflow (12)
                  | 029 || 100   if illegal memory address (4)
                  | 028 || 1111  if floating point exception (15)
                  | 028 || 1000  is SYSCALL instruction (8)
                  \ .....      etc.
PC <-- 80000180hex

```

Also, mode of CPU operation changes to kernel.