## ARITHMETIC CORE INSTRUCTION SET

| NAME | MNE-MON-IC | FOR-MAT | OPERATION (in Verilog) | | OPCODE/ FMT/FT/ FUNCT |
|---|---|---|---|---|---|
| Divide | `div` | R | Lo=R[rs]/R[rt]; Hi=R[rs]%R[rt] | | 0/–/–/1a |
| Divide Unsigned | `divu` | R | Lo=R[rs]/R[rt]; Hi=R[rs]%R[rt] | (6) | 0/–/–/1b |
| Multiply | `mult` | R | {Hi,Lo}=R[rs]∗R[rt] | | 0/–/–/18 |
| Multiply Unsigned | `multu` | R | {Hi,Lo}=R[rs]∗R[rt] | (6) | 0/–/–/19 |
| Branch On FP True | `bc1t` | FI | if(FPCond) PC=PC+4+BranchAddr | (4) | 11/8/1/– |
| Branch On FP False | `bc1f` | FR | if(!FPCond) PC=PC+4+BranchAddr | (4) | 11/8/0/– |
| FP Compare Single | `c.`*x*`.s`* | FR | FPCond=(F[fs] *op* F[ft])?1:0 | | 11/10/–/*y* |
| FP Compare Double | `c.`*x*`.d`* | FR | FPCond=({F[fs],F[fs+1]} *op* {F[ft],F[ft+1]})?1:0 | | 11/11/–/*y* |
| | | | *(*x* is `eq`, `lt` or `le`) (*op* is ==, < or <=) (*y* is 32, 3c or 3e) | | |
| FP Add Single | `add.s` | FR | F[fd]=F[fs]+F[ft] | | 11/10/–/0 |
| FP Divide Single | `div.s` | FR | F[fd]=F[fs]/F[ft] | | 11/10/–/3 |
| FP Multiply Single | `mul.s` | FR | F[fd]=F[fs]∗F[ft] | | 11/10/–/2 |
| FP Subtract Single | `sub.s` | FR | F[fd]=F[fs]-F[ft] | | 11/10/–/1 |
| FP Add Double | `add.d` | FR | {F[fd],F[fd+1]}={F[fs],F[fs+1]}+{F[ft],F[ft+1]} | | 11/11/–/0 |
| FP Divide Double | `div.d` | FR | {F[fd],F[fd+1]}={F[fs],F[fs+1]}/{F[ft],F[ft+1]} | | 11/11/–/3 |
| FP Multiply Double | `mul.d` | FR | {F[fd],F[fd+1]}={F[fs],F[fs+1]}∗{F[ft],F[ft+1]} | | 11/11/–/2 |
| FP Subtract Double | `sub.d` | FR | {F[fd],F[fd+1]}={F[fs],F[fs+1]}-{F[ft],F[ft+1]} | | 11/11/–/1 |
| Move From Hi | `mfhi` | R | R[rd]=Hi | | 0/–/–/10 |
| Move From Lo | `mflo` | R | R[rd]=Lo | | 0/–/–/12 |
| Move From Control | `mfc0` | R | R[rd]=CR[rs] | | 16/0/–/0 |
| Load FP Single | `lwc1` | I | F[rt]=M[R[rs]+SignExtImm] | (2) | 31/–/–/– |
| Load FP Double | `ldc1` | I | F[rt]=M[R[rs]+SignExtImm]; F[rt+1]=M[R[rs]+SignExtImm+4] | (2) | 35/–/–/– |
| Store FP Single | `swc1` | I | M[R[rs]+SignExtImm]=F[rt] | (2) | 39/–/–/– |
| Store FP Double | `sdc1` | I | M[R[rs]+SignExtImm]=F[rt]; M[R[rs]+SignExtImm+4]=F[rt+1] | (2) | 3d/–/–/– |

## ASSEMBLER DIRECTIVES

| | |
|---|---|
| `.data` [*addr*]* | Subsequent items are stored in the data segment |
| `.kdata` [*addr*]* | Subsequent items are stored in the kernel data segment |
| `.ktext` [*addr*]* | Subsequent items are stored in the kernel text segment |
| `.text` [*addr*]* | Subsequent items are stored in the text |
| | * starting at [*addr*] if specified |
| `.ascii` *str* | Store string *str* in memory, but do not null-terminate it |
| `.asciiz` *str* | Store string *str* in memory and null-terminate it |
| `.byte` $b_1, \ldots, b_n$ | Store the $n$ values in successive bytes of memory |
| `.double` $d_1, \ldots, d_n$ | Store the $n$ floating-point double precision numbers in successive memory locations |
| `.float` $f_1, \ldots, f_1$ | Store the $n$ floating-point single precision numbers in successive memory locations |
| `.half` $h_1, \ldots, h_n$ | Store the $n$ 16-bit quantities in successive memory halfwords |
| `.word` $w_1, \ldots, w_n$ | Store the $n$ 32-bit quantities in successive memory words |
| `.space` $n$ | Allocate $n$ bytes of space in the current segment |
| `.extern` *symsize* | Declare that the datum stored at *sym* is *size* bytes large and is a global label |
| `.globl` *sym* | Declare that label *sym* is global and can be referenced from other files |
| `.align` $n$ | Align the next datum on a $2^n$ byte boundary, until the next `.data` or `.kdata` directive |
| `.set at` | Tells SPIM to complain if subsequent instructions use `$at` |
| `.set noat` | prevents SPIM from complaining if subsequent instructions use `$at` |

## SYSCALLS

| SERVICE | $v0 | ARGS | RESULT |
|---|---|---|---|
| `print_int` | 1 | integer `$a0` | |
| `print_float` | 2 | float `$f12` | |
| `print_double` | 3 | double `$f12/$f13` | |
| `print_string` | 4 | string `$a0` | |
| `read_int` | 5 | | integer (in `$v0`) |
| `read_float` | 6 | | float (in `$f0`) |
| `read_double` | 7 | | double (in `$f0`) |
| `read_string` | 8 | buf `$a0`, buflen `$a1` | |
| `sbrk` | 9 | amount `$a` | address (in `$v0`) |
| `exit` | 10 | | |

## EXCEPTION CODES

| Number | Name | Cause of Exception |
|---|---|---|
| 0 | Int | Interrupt (hardware) |
| 4 | AdEL | Address Error Exception (load or instruction fetch) |
| 5 | AdES | Address Error Exception (store) |
| 6 | IBE | Bus Error on Instruction Fetch |
| 7 | DBE | Bus Error on Load or Store |
| 8 | Sys | Syscall Exception |
| 9 | Bp | Breakpoint Exception |
| 10 | RI | Reserved Instruction Exception |
| 11 | CpU | Coprocessor Unimplemented |
| 12 | Ov | Arithmetic Overflow Exception |
| 13 | Tr | Trap |
| 15 | FPE | Floating Point Exception |

[1] Patterson, David A; Hennessy, John J.: Computer Organization and Design, 3rd Edition. Morgan Kaufmann Publishers. San Francisco, 2005.