# Overview

CS01 is a thirty-two-bit processor. The processor features 32, 32-bit integer registers and 32, 80-bit floating-point registers. The processor has the most prominent features from real processors with some of the more complex details left out.

Memory is 80-bits wide to accommodate floating point values. There are no unaligned memory accesses. Thirty-two-bit values are fetched from the lowest 32 bits of a memory cell.

Instruction memory is 32-bits wide and separate from data memory.

# Programming Model

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Registers | | | |
| S2/T2 | 31 0 |  | S2/T2 | 79 0 |
| 0 | x0 |  | 1 | f0 |
| 0 | x1 |  | 1 | f1 |
| 0 | … |  | 1 | … |
| 0 | x30 |  | 1 | f30 |
| 0 | x31 / sp |  | 1 | f31 |
|  | pc |  |  |  |

x? refers to an integer register. f? refers to a floating-point register

Registers x0 and f0 are always zero. f0 is positive zero.

x31 / sp is the stack pointer used by the JSR and RTS instructions.

pc is the program counter.

# Nomenclature

The size of values are referred to as the following:

|  |  |
| --- | --- |
| Size |  |
| 1 | byte |
| 2 | wyde |
| 4 | tetra |
| 5 | penta |
| 8 | octa |
| 10 | deci |
| 16 | hexi |

# Instruction Set

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Opcode  6 bits | Target Reg  5 bits | | Source Reg  5 bits | | Source Reg  5 bits | | 5 bits | | | | | | Funct  6 bits | |
| ADD | 2 | Rd | | Rs1 | | Rs2 | | ~ | | | | | | 4 | |
| SUB | 2 | Rd | | Rs1 | | Rs2 | | ~ | | | | | | 5 | |
| MUL | 2 | Rd | | Rs1 | | Rs2 | | ~ | | | | | | 11 | |
| CMP | 2 | Rd | | Rs1 | | Rs2 | | ~ | | | | | | 6 | |
| CMPU | 2 | Rd | | Rs1 | | Rs2 | | ~ | | | | | | 7 | |
| AND | 2 | Rd | | Rs1 | | Rs2 | | ~ | | | | | | 8 | |
| OR | 2 | Rd | | Rs1 | | Rs2 | | ~ | | | | | | 9 | |
| EOR | 2 | Rd | | Rs1 | | Rs2 | | ~ | | | | | | 10 | |
| Shift |  |  | |  | |  | |  | | | | | |  | |
| ASL | 2 | Rd | | Rs1 | | Rs2 | | 0 | | | | | | 3 | |
| ASR | 2 | Rd | | Rs1 | | Rs2 | | 1 | | | | | | 3 | |
| SHL | 2 | Rd | | Rs1 | | Rs2 | | 2 | | | | | | 3 | |
| SHR | 2 | Rd | | Rs1 | | Rs2 | | 3 | | | | | | 3 | |
| ROL | 2 | Rd | | Rs1 | | Rs2 | | 4 | | | | | | 3 | |
| ROR | 2 | Rd | | Rs1 | | Rs2 | | 5 | | | | | | 3 | |
|  |  |  | |  | | Constant  16 bits | | | | | | | | | |
| ADD # | 4 | Rd | | Rs1 | | imm | | | | | | | | | |
| SUB # | 5 | Rd | | Rs1 | | imm | | | | | | | | | |
| MUL # | 11 | Rd | | Rs1 | | imm | | | | | | | | | |
| CMP # | 6 | Rd | | Rs1 | | imm | | | | | | | | | |
| CMPU # | 7 | Rd | | Rs1 | | imm | | | | | | | | | |
| AND # | 8 | Rd | | Rs1 | | imm | | | | | | | | | |
| OR # | 9 | Rd | | Rs1 | | imm | | | | | | | | | |
| EOR # | 10 | Rd | | Rs1 | | imm | | | | | | | | | |
| ASL # | 3 | Rd | | Rs1 | | ~ | | 0 | | | | | | | imm |
| ASR # | 3 | Rd | | Rs1 | | ~ | | 1 | | | | | | | imm |
| SHL # | 3 | Rd | | Rs1 | | ~ | | 2 | | | | | | | imm |
| SHR # | 3 | Rd | | Rs1 | | ~ | | 3 | | | | | | | imm |
| ROL # | 3 | Rd | | Rs1 | | ~ | | 4 | | | | | | | imm |
| ROR # | 3 | Rd | | Rs1 | | ~ | | 5 | | | | | | | imm |
| MOV | 12 | FRd / Rd | | FRs1 / Rs1 | | ~ | | ~ | T2 | | | S2 | | | ~ |
|  |  |  | |  | |  | | | | | | | | | |
| FTOI | 15 | FRd / Rd | | FRs1 | | ~ | | ~ | | | T | |  | | 2 |
| ITOF | 15 | FRd | | FRs1 / Rs1 | | ~ | | ~ | | |  | | S | | 3 |
| FADD | 15 | FRd | | FRs1 | | FRs2 | | ~ | | Rm | | | | | 4 |
| FSUB | 15 | FRd | | FRs1 | | FRs2 | | ~ | | Rm | | | | | 5 |
| FMUL | 15 | FRd | | FRs1 | | FRs2 | | ~ | | Rm | | | | | 11 |
| FDIV | 15 | FRd | | FRs1 | | FRs2 | | ~ | | Rm | | | | | 12 |
| FSQRT | 15 | FRd | | FRs1 | | FRs2 | | ~ | | Rm | | | | | 13 |
| FCMP | 15 | Rd | | FRs1 | | FRs2 | | ~ | | Rm | | | | | 6 |
|  |  |  | |  | |  | |  | | | | | | | |
| NOP | 16 | ~ | | ~ | | | | | | | | | | | |
| BEQZ | 24 | Rs1 | | Displacement21 | | | | | | | | | | | |
| BNEZ | 25 | Rs1 | | Displacement21 | | | | | | | | | | | |
| BLTZ | 26 | Rs1 | | Displacement21 | | | | | | | | | | | |
| BLEZ | 27 | Rs1 | | Displacement21 | | | | | | | | | | | |
| BGTZ | 28 | Rs1 | | Displacement21 | | | | | | | | | | | |
| BGEZ | 29 | Rs1 | | Displacement21 | | | | | | | | | | | |
| BRA | 30 | Rs1 | | Displacement21 | | | | | | | | | | | |
| JMP | 31 | Address26 | | | | | | | | | | | | | |
| JSR | 32 | Address26 | | | | | | | | | | | | | |
| JMP | 33 | Rs1 | | imm | | | | | | | | | | | |
| JSR | 34 | Rs1 | | imm | | | | | | | | | | | |
| RTS | 35 | 31 | | 31 | | Amount16 | | | | | | | | | |
|  |  |  | | | | | | | | | | | | | |
| LDT | 48 | Rd | Rs1 | | imm | | | | | | | | | | |
| LDF | 49 | FRd | Rs1 | | imm | | | | | | | | | | |
| POP | 50 | Rd | ~ | | ~ | | | | | | | | | | |
| POPF | 51 | FRd | ~ | | ~ | | | | | | | | | | |
| STT | 56 | imm | Rs1 | | Rs2 | | imm | | | | | | | | |
| STF | 57 | imm | Rs1 | | FRs2 | | imm | | | | | | | | |
| PUSH | 58 | ~ | ~ | | Rs2 | | ~ | | | | | | | | |
| PUSHF | 59 | ~ | ~ | | FRs2 | | ~ | | | | | | | | |
|  |  |  | | | | | | | | | | | | | |

# Supervisor Programming Model

The supervisor programming model includes an additional set of registers and several instructions that may only be used in supervisor mode.

## Supervisor Register Set

|  |  |  |
| --- | --- | --- |
| S2/T2 | Supervisor Mode Registers | |
| 2 | x0 |  |
| 2 | x1 |  |
| 2 | … |  |
| 2 | x30 |  |
| 2 | x31 / sp |  |
|  | pc |  |
|  |  |  |

# Supervisor Instructions

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Opcode  6 bits | Target Reg  5 bits | | Source Reg  5 bits | | Source Reg  5 bits | | 5 bits | | Funct  6 bits | | | |
|  |  |  | |  | | Constant  16 bits | | | | | | | |
| CSR | 13 | Rd | | Rs1 | | CSR Regno10 | | | | | ~ | | Op2 |
|  |  |  | |  | |  | | | | | | | |
| SYS | 0 | 0 | 0 | | 0 | | ~ | | Cause8 | | | | |
| SEI | 0 | Rd | Rs1 | | 1 | | ~ | | | | | imm4 | |
| PFI | 0 | Rd | Rs1 | | 2 | | ~ | | | | | imm4 | |
| RTI | 0 | 0 | 0 | | 3 | | ~ | | | | | | |
|  |  |  | | | | | | | | | | | |

## CSR – Control and Status Register Access

**Description**:

There are four operations possible with this instruction.

1. Read a control or status register. The register’s value is transferred to Rd.
2. Read and write a control or status register. This operation reads the current value of the control register into Rd then sets a new value for the register from register Rs1. The swap takes place atomically.
3. Read the control register into Rd then set individual bits.
4. Read the control register into Rd then clear individual bits.

## PFI – Poll For Interrupt

**Description**:

This instruction checks for an interrupt at the point in time the instruction is executed. If an interrupt is present then the interrupt service routine is entered. Interrupts do not have to be enabled in order for the PFI instruction to transfer the program execution to the interrupt subroutine. The interrupt level checked for is greater than or equal to the bitwise or of the value in Rs1 and an immediate constant in the instruction. Interrupts below this level will not be recognized. The interrupting level is returned in register Rd.

## RTI – Return from Interrupt

**Description**:

This instruction returns from a system routine entered via the SYS instruction. The interrupt level is reset to the level active when the SYS instruction was invoked.

## SEI – Set Interrupt Mask

**Description**:

This instruction sets the interrupt mask level to a value specified by the bitwise or of Rs1 and an immediate constant in the instruction. The previous interrupt setting is returned in register Rd.

## SYS – Invoke System

**Description**:

This instruction generates an interrupt which switches the processor to supervisor mode. The cause field of the instruction is stored in the cause CSR.

Contrast with “real” Computing Hardware

Most machines have 32 or fewer registers.

* Some experimentation of the number of available registers shows there isn’t much benefit in most applications to having more than 32 registers. Effectively having 32 registers is almost as good as having an infinite number of registers. Many small functions use only a handful of registers.
* Machine code readability is not a primary focus of most instruction sets. Encoding space takes precedence.

Most machines have separate floating-point and integer register sets.

* Separate register sets make it easier to make floating-point hardware optional.
* Instructions can be encoded using fewer bits with separate register sets.

Many instructions sets are much narrower to conserve memory space.

* Memory has a cost associated with it. Instruction sets are organized to make the best use of available memory.
* Conserving memory space is important enough that there are compressed versions of instructions sets like the Arm or RiSCV compressed instructions. Or the instruction set length varies (x86, 68k).

Some questions for the reader

How to make the instruction set more compact?

* There are a lot of ways to do this. Fewer registers, smaller opcode and funct encodings.
* Shift operations can be encoded more compactly.
* See if the student can come up with relative branch addressing.

Why isn’t there a divide operation?

* The standard division method is quite slow, takes a lot of machine cycles. It’s much faster to multiply by the reciprocal or use Newton-Raphson iterations.

How could loads and stores be improved?

* There is currently only load and store tetra-byte values. It would be better if there were loads and stores for other sized values as well.