**RISC-V**

RISC-V es un ISA (Instruction Set Architecture) disponible para el libre desarrollo del mismo (open source), que nació en 2010 en la universidad de California en Berkeley con un propósito esencialmente educativo, y ha estado creciendo en popularidad gracias entre otras cosas a la flexibilidad que aporta por la gran variedad de extensiones que se pueden montar sobre la misma, y a la oportunidad de explotarla libremente a diferencia de otras arquitecturas como x86 o ARM.

**Implementaciones Actuales**

Como esta arquitectura se construye a partir de las múltiples extensiones de la misma, se vuelve bastante flexible en sus posibles aplicaciones, desde sistemas empotrados como microcontroladores hasta supercomputadores y servidores. Para dar un ejemplo de las posibles implementaciones dejo una lista con las más recientes y significativas, tratando de dar la mejor visualización de su utilidad actual.

**ESP32-C61:** Este procesador le da conexión inalámbrica (conexión wifi 6), bluetooth a dispositivos empotrados donde tiene un único core RISC-V microcontrolador. Este dispositivo salió al mercado el 8 enero del 2024 marcando un claro ejemplo de la implementación de RISC-V para microcontroladores.

Referencia: <https://www.espressif.com/en/products/socs/esp32-c61>

**SiFive Performance P870-D:** Este procesador es de la marca SiFive, compatible con RVA23, perfil de implementación de RISC-V, este es un procesador planteado para centros de datos donde se necesita una alta cantidad de paralelización enfocándose en la eficiencia y tamaño, y no necesariamente en la potencia máxima de cómputo. Su salida está planteada para finales de 2024.

Referencia: <https://www.sifive.com/cores/performance-p870d>

**NV-RISCV:** NVIDIA se ha vuelto un gran impulsor de RISC-V estos últimos años, desde el 2015 aproximadamente NVIDIA ha ido sustituyendo los controladores “falcom” por cores RISC-V desarrollando a lo largo de los años hasta 20 extensiones especializadas sobre este estándar, y ahora se estima que cada chip incluye entre 10 y 40 cores RISC-V.

Referencia: <https://www.tomshardware.com/pc-components/gpus/nvidia-to-ship-a-billion-of-risc-v-cores-in-2024>

**Menciones adicionales:**

- Alibaba con T-Head XuanTie C910 y la CPU Sophon SG2042

- Western Digital's con su procesador SweRV Core

**Opciones Base**

A pesar de que RISC-V este montado sobre varias extensiones, este comparte algunas partes independientemente de las extensiones que se monten. Para empezar, esta arquitectura es una arquitectura load-store donde los accesos a memoria se solo hacen desde estas dos instrucciones y el resto de las operaciones se hace directamente en CPU, como por ejemplo operaciones aritméticas y lógicas. La longitud general de las instrucciones es de 32 bits, aunque soporta otros tipos de longitudes en sus extensiones, esto siempre que estén alineadas a media palabra (16 bits). Esto es decisión de diseño, de manera que las instrucciones quedan divididas en parcelas de media palabra. Para dar soporte a los dos tipos posibles de alineaciones (big-endian y little-endian) las parcelas se almacenan en memoria con alineación little-endian independientemente de la alineación del procesador. En la figura 1 podemos ver un formato para diferenciar la longitud de una instrucción de antemano.

Figura 1

Tabla

Descripción generada automáticamente con confianza baja

Continuando con el diseño general de esta arquitectura tenemos un total de 31 registros para propósito general (x1-x31), donde contamos con un registro x0 que estará puesto siempre a 0 independientemente de lo que se cargue en él (forzado por hardware). La longitud de estos registros depende de la versión base: para RV32 estos tienen una longitud de 32 bits mientras que para RV64 tienen una longitud de 64 bits. Si comparamos la versión de 32 bits nos encontramos con una situación muy parecida a la del procesador mc88110. Luego nos encontramos con 32 registros de coma flotante (f0-f31) que de forma general tendrán que tener tantos bits como sea necesario para el soporte que se necesite de las extensiones que se vayan a implementar, por ejemplo, de forma general tenemos simple precisión (32 bits), doble precisión (64 bits). Aunque cabe resaltar que hay extensiones que soportan la media precisión y la cuádruple precisión, como se podrá ver en el próximo apartado. Para registros de carácter específico tenemos el PC (program counter) junto al registro FSR (floating point status register) que es un registro para el modo de operación, redondeo, y el estado de excepciones de la unidad de coma flotante. De forma adicional tenemos una serie de formatos de instrucción base como se puede ver en la figura 2.

Figura 2

Escala de tiempo

Descripción generada automáticamente

**Extensiones**

A partir de este punto voy a introducir algunas de las posibles extensiones junto con las instrucciones y partes que me han parecido curiosas o diferentes de las instrucciones del procesador motorola 88110, procesador RISC.

Empezado primero en que hay dos extensiones base sobre las que se monta el resto, siendo estas RV32 y RV64 que se diferencian sobre todo en la longitud de los registros de propósito general, y en el soporte que dan sobre distintas longitudes de direcciones de memoria, hasta 32 bits para RV32 y 64 bits para RV64. También está la versión RV128 para direcciones de memoria de 128 bits. Esta, al igual que RV64, se monta sobre su versión anterior, en este caso RV64, para este habría que incluir instrucciones para el acceso y calculo con direcciones y registros de 128 bits, sin embargo, como esta extensión es todavía muy prematura en su implementación en este documento me voy a centrar en las extensiones RV64 y RV32.

**RV32I**

Esta es la extensión base de instrucciones de enteros de 32 bits, en este caso se da soporte a direcciones de 32 bits asumiendo como se ha mencionado antes registros de 32 bits. Tenemos en este caso instrucciones básicas de operaciones con números enteros como: operaciones lógicas (AND, OR, XOR), operaciones aritméticas (ADD, SUB), de desplazamientos (SLL, SRL, SRA) junto con operaciones de carga (LW, SW, LBU, LB, SB, LH, LHU …) e instrucciones de control, tanto saltos incondicionales (JAL, JALR), como saltos condicionales (BEQ, BNE, BLT, BGE) también se cuenta con instrucciones de ordenación de memoria (FENCE) y llamadas al sistema (ECALL, EBREAK). Como punto a resaltar es que los datos inmediatos en la mayoría de las instrucciones son de 12 bits de signo extendido, por supuesto hay excepciones como es el caso de JAL donde el offset es de 20 bits. También se han encontrado algunas partes que me han parecido curiosas en la decisión de diseño, para empezar en los accesos a memoria; en las instrucciones de carga y almacenamiento la dirección de destino se calcula a partir del valor de un registro con un offset inmediato de 12 bits con signo extendido, a diferencia del procesador mc88110 donde para el cálculo de la dirección destino el offset se podía encontrar en un registro. La mayoría de las operaciones tanto lógicas como aritméticas cuentan con una versión que soporta datos inmediatos, sin embargo, para el caso de la operación resta (SUB) no se tiene una versión que permita restar a un registro datos inmediatos, al igual que no se cuentan con versiones ni para la suma ni para la resta para operar con datos sin signo. También para el caso de los saltos condicionales a diferencia del procesador mc88110 no hay que hacer una llamada previa a una instrucción de comparación, sino que la comparación se realiza de manera interna en la propia operación.

**RV64I**

Esta es la extensión base para las operaciones con números enteros de 64 bits. Para este caso, podríamos decir que esta extensión se monta sobre la anterior puesto que todas las instrucciones que teníamos en la extensión RV32I las tenemos en esta extensión. Para empezar ahora se tienen registros con 64 bits, con ello el tamaño natural de las operaciones pasa a ser de 64 bits. Sin embargo, se han añadido una serie de instrucciones adicionales para dar soporte a las instrucciones a nivel de palabra (32 bits) de la extensión anterior, para diferenciarlas se les ha añadido w al final de la instrucción (word). Por ejemplo: addiw para sumar los 32 bits menos significativos de un registro con un dato inmediato y cargar el resultado en un registro destino. Inclusive para las instrucciones de carga y almacenamiento donde ya se había implementado este formato, ahora contamos con operaciones como ld para cargar 64 bits, y como lwu para cargar en los 32 bits menos significativos un dato que se encuentre en memoria sin extender el signo, instrucciones ilegales en la extensión RV32I. Aún cuando se ha extendido la longitud máxima de los registros de propósito general el tamaño y formato de las instrucciones se mantiene con respecto al anterior, esto puesto que no hay una necesidad clara de la misma para las características de estas funciones.

**RV32E y RV64E**

Estas dos extensiones nacen para soportar microcontroladores en sistemas empotrados, donde la memoria y el espacio es un recurso preciado. Estas extensiones son extensiones base como RV32I y RV64I donde la única diferencia con estas se encuentra en la cantidad de registros de propósito general disponibles, que se ha reducido a 16 para ganar espacio. Entones las instrucciones son las mismas con la diferencia clave de que las que usan registros del x16 al x31 pasan a estar ahora reservadas.

**M**

La extensión M se monta sobre alguna de las extensiones base y, lo que permite es usar operaciones aritméticas sobre enteros de mayor complejidad como son las multiplicaciones y divisiones. Entonces para esta extensión tenemos tres tipos de operaciones principales: la operación multiplicación (MUL), la operación división (DIV) y la operación módulo (REM). Cada una de estas operaciones se puede hacer sobre una longitud específica (word, half, byte, …) y tienen versiones para trabajar con datos sin signo. Sin embargo, como punto a resaltar, no se han implementado en esta extensión operaciones con datos inmediatos, de forma que todas las operaciones, tanto divisiones, como multiplicaciones, como el resto se tienen que hacer entre registros.

**Zmmul**

Esta extensión está pensada para tener un subconjunto de las instrucciones de la extensión M, en este caso son solo las instrucciones que dan soporte a la operación de multiplicación, y además solo una parte de las mismas: MUL, MULW, MULH, MULHU, MULHSU. De manera que no se implementen ni la operación división ni la operación resta. Esta pensada entonces para casos específicos en los que la operación de división, que es altamente costosa y compleja, no se vaya a usar con frecuencia o no sea necesaria.

**F**

Esta extensión añade las operaciones principales para el manejo general de números de coma flotante en simple precisión (32 bits), por tanto, para implementar esta extensión la longitud de los registros de coma flotante (f0-f31) necesita ser de al menos 32 bits. Para el control de algunas partes de estas operaciones como el tipo de redondeo dinámico o los flags de las excepciones, tenemos, como se había mencionado en la introducción, el registro de propósito específico floating-point control and status register (fcsr) que se puede leer y escribir para ajustar las operaciones. Entrando en las instrucciones, tenemos primero las instrucciones de carga y almacenamiento (FLW y FSW), luego tenemos las operaciones clásicas como son la suma, la resta, la multiplicación, la división (FADD, FSUB, FMUL, FDIV), y adicionalmente tenemos una serie de instrucciones más complejas como son la raíz cuadrada (FSQRT), o la operación mínimo y máximo (FMIN, FMAX) junto con operaciones compuestas (FNMADD.S, FMADD.S, FNMSUB.S, FMSUB.S) que juntan dos operaciones como son la multiplicación y la suma en una única instrucción, el típico daxpy. Para complementar las funciones básicas de los números de coma flotante tenemos instrucciones de conversión, tanto de entero a coma flotante como viceversa junto con instrucciones move que nos permiten copiar y mover los datos de un registro de coma flotante a otro. Contamos también con instrucciones que nos permiten hacer comparaciones con los números de coma flotante (FCMP), así como clasificarlos (FCLASS).

**D**

Esta extensión de forma similar a la anterior nos permite implementar las operaciones básicas con coma flotante de doble precisión (64 bits), necesitando entonces registros de coma flotante de 64 bits de longitud para poder trabajar. Entrando en el apartado de las instrucciones, si comparamos esta extensión a la anterior, nos encontramos las mismas instrucciones solo cambiando una parte para diferenciarlas de las instrucciones reservadas para simple precisión, es decir donde antes se añadía en las instrucciones .S ahora se añade .D, por ejemplo: cuando en simple precisión para la suma teníamos FADD.S ahora tenemos FADD.D, y si hablamos de las instrucciones de carga y almacenamiento tenemos ahora las dos instrucciones: FLD y FSD.

**Continuar…….**