CC3301 Programación de software de sistemas

Arquitectura lógica y física de una CPU, implementaciones básicas y avanzadas, memoria caché

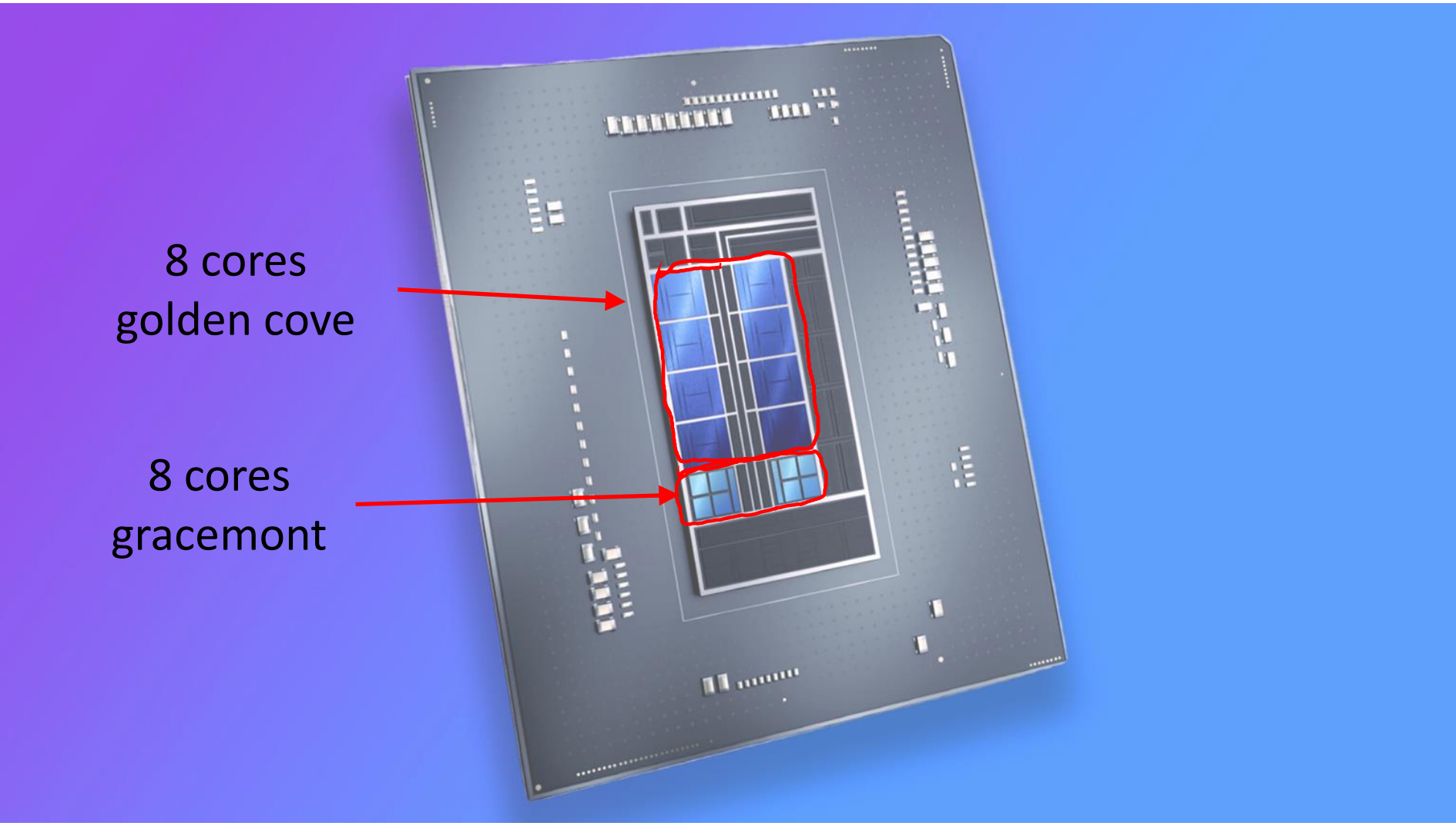
La arquitectura lógica de una CPU

- Definición informal: Una *CPU* (*central processing unit*) es la componente del computador encargada de ejecutar los programas
- Definición: La *arquitectura* (o *arquitectura lógica*) de una CPU es una *especificación* de la codificación binaria de las instrucciones de máquina, de qué hace exactamente cada instrucción, los registros, etc.
- Dos CPUs con la misma arquitectura pueden ejecutar el mismo sistema operativo y los mismos binarios compilados para ese sistema operativo, exitosamente
- Por ejemplo las CPUs AMD Ryzen e Intel Core (i7, i5 o i3), Pentium, Celeron poseen la misma arquitectura
- Las CPUs de Qualcomm, Apple y Samsung para celulares poseen la misma arquitectura (Arm v8) pero distinta de las CPUs de Intel y AMD
- La arquitectura + el sistema operativo se denomina la *plataforma*: determina qué programas se pueden ejecutar en formato binario *directamente*
- Existen *emuladores* que traducen las instrucciones de una arquitectura foránea, a las instrucciones de la arquitectura anfitriona, *on the fly*, como por ejemplo *qemu*, pero dependiendo de su calidad, el sobrecosto puede ser importante (4x para *qemu-riscv32* bajo x86)

La arquitectura física de una CPU

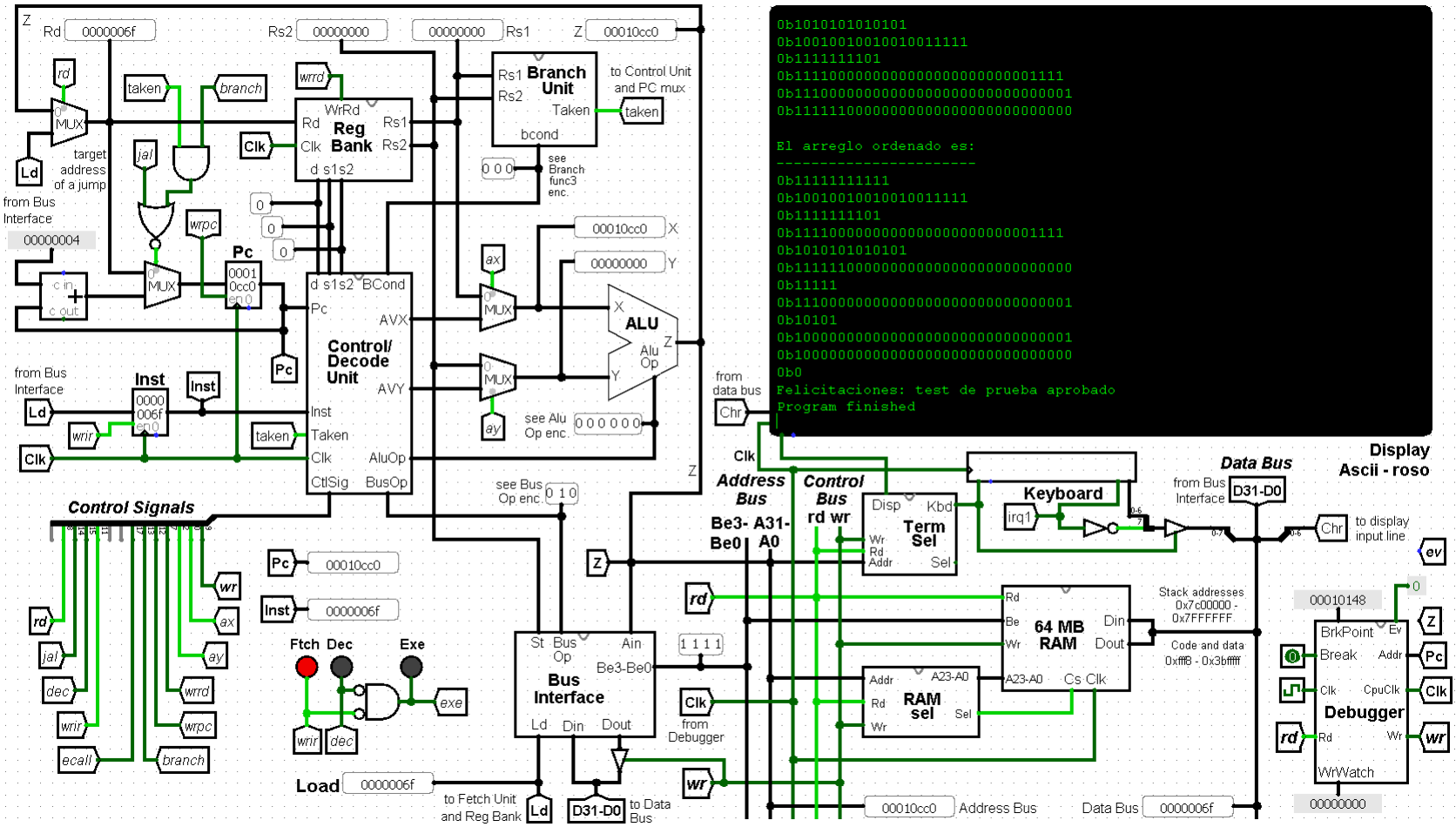
- Definición: La *arquitectura física* de una CPU es su implementación a nivel de circuitos
- Definición: un *core* es una implementación específica de una CPU que es capaz de ejecutar un thread (a veces 2 threads, gracias al *hyperthreading* de Intel)
- Parámetros relevantes de un core:
  - Area que ocupa en el chip (mm^2): incide en el costo (no linealmente)
  - Frecuencia del reloj (GHz): incide en la velocidad y negativamente en el consumo
  - Potencia (watts): incide en el consumo energético
  - IPC (*instructions per clock*): incide en la velocidad de ejecución de un thread
  - Eficiencia energética: trabajo realizado por joule
- *Conocer la arquitectura física de una CPU ayuda a programar eficientemente para esa CPU*
- Problema: a mayor velocidad, mayor área y menor eficiencia energética
- El nuevo *Alder Lake* de Intel tiene 8 cores de alto desempeño *golden cove* (con *hyperthreading*) y 8 cores eficientes energéticamente *gracemont*

El silicio de un chip: *die*



Intel Alder Lake
(lanzamiento: octubre 2021)

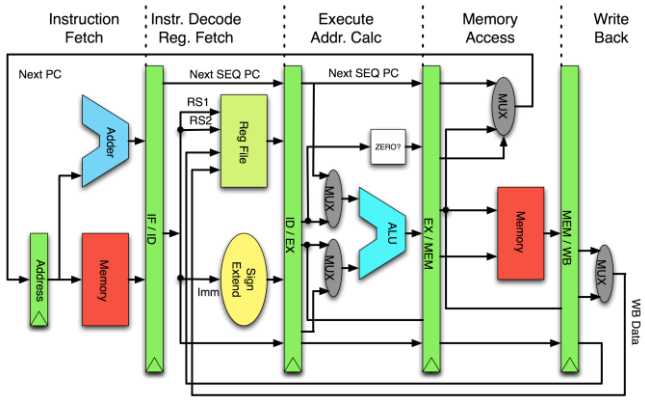
Diagrama de bloques



Implementación de Risc-V en Logisim

Implementaciones básicas

- Hasta fines de los 80: *microprogramación*
  - Cada instrucción requiere varios ciclos del reloj: un ciclo para decodificar una instrucción, otro para leer los argumentos, otro para realizar la operación, otro para guardar el resultado
  - Se ocupaba en los *mainframes* de IBM, minis de Digital, cpus de Intel y Amd hasta la Intel 386 (1985)
- A principios de los 80: pipeline, 1 instrucción x ciclo
  - Similar a la cadena de ensamblaje de automóviles de Henry Ford



  - Especialmente adaptadas para las CPUs RISC
  - Ejemplo de pipeline: carga de la instrucción, decodificación, ejecución, acceso a memoria y escritura
  - A mayor número de etapas del pipeline, mayor frecuencia del reloj y por lo tanto la velocidad del core
  - Intel lo ocupó por primera vez en la 486 (1989)
  - Problema: una instrucción lenta como una división o lectura de memoria bloquea todo el pipeline

Implementaciones avanzadas

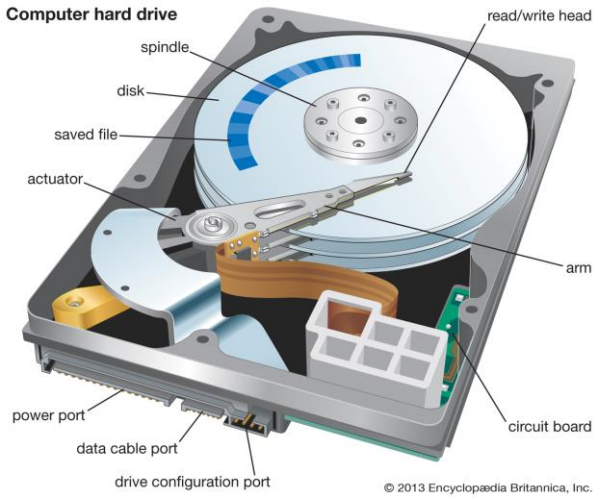
- A principios de los 90: superescalar, *n* inst. x ciclo
  - Varios pipelines síncronos trabajando en paralelo
  - Ejemplos: primer Pentium (1993), Cortex A55
  - Ventajas: área pequeña, alta eficiencia energética, alta eficiencia por área de silicio
  - Problemas: la instrucciones con dependencias no se pueden ejecutar en paralelo, si se bloquea un pipeline, se bloquean todos los demás
- Mediados de los 90: ejecución especulativa y fuera de orden (Intel Pentium Pro, 1995)
  - Múltiples unidades de ejecución que operan en paralelo: alus, coprocesadores de punto flotante, lectura y escritura de memoria, saltos condicionales
  - Ejemplos: Pentium Pro (1995), Cortex A73, …, Cortex X1
  - Instrucciones se ejecutan cuando están listos sus operandos
  - En una división o lectura de memoria se continúa ejecutando las instrucciones siguientes, a menos que usen los resultados de la división o lectura
  - En un salto condicional se especula (se predice) si ocurre o no
  - Ventaja: alta velocidad de ejecución de un thread
  - Problemas: extremadamente complejo de diseñar, baja eficiencia energética, gran área de silicio

Problema: la velocidad de la memoria

- Los computadores almacenan los programas en ejecución en memorias *DRAM* (*Dynamic RAM*)
- Son baratas por bit y densas
- Un bit se almacena en un condensador
- Son veloces de leer y escribir, pero no lo suficiente para alimentar la CPU con instrucciones y datos a la velocidad requerida

- Las memorias flash (SSDs, pendrives, microSD) son más baratas por bit y más densas que las DRAM pero *no pueden almacenar un programa en ejecución*, la lectura y escritura es más lenta, se deterioran después de unas 1000 escrituras y los SSDs requieren controladores complejos para nivelar el desgaste de los bits

- Los discos son aún más baratos por bit y más densos pero son aún más lentos que las memorias flash, especialmente en acceso directo (*fseek*) porque deben mover físicamente el cabezal consiguiendo apenas unos 100 accesos directos por segundo



La memoria caché

- Las memorias *SRAM* (*Static RAM*) son más rápidas que las DRAM pero son menos densas porque requieren 6 transistores por bit y por lo tanto más caras
- El dilema es que a mayor tamaño de la SRAM, más lento resulta leerla o escribirla
- Además acceder a datos que están fuera del chip de la CPU toma más tiempo y energía que leer datos que se encuentran dentro del mismo chip
- Solución: la *memoria caché*
- Es muy veloz porque es una SRAM de pequeño tamaño y va dentro del chip de la CPU
- Almacena una copia de una pequeña parte de la DRAM, pero que tiene una alta probabilidad de ser usada en el futuro cercano porque el acceso a la memoria de un programa exhibe estas 2 propiedades:
  - ✓ *Localidad temporal*: si un programa accedió a una dirección de memoria, es altamente probable que acceda a la misma dirección prontamente
  - ✓ *Localidad espacial*: si un programa accedió a una dirección de memoria, es altamente probable que acceda a las direcciones adyacentes prontamente

Diseño de un caché de 64 KB de 1 grado de asociatividad con líneas de 16 bytes

- La memoria DRAM se descompone en líneas de 16 bytes c/u. Decimos que M[L] es la L-ésima línea de la DRAM.
- El caché es una pequeña SRAM ultra rápida que puede contener copias de apenas C=4096 líneas de la DRAM. Decimos que C[LC] es la LC-ésima línea del caché. C siempre es una potencia de 2.
- Un dato con dirección D se ubica en M[L] con L=D/16: L es la etiqueta de la línea. Si esa línea está en el caché solo puede estar en C[LC] con LC=L%C *¡los 12 bits menos significativos de L!*
- Una SRAM adicional de tamaño C almacena las etiquetas de las líneas contenidas en el caché y un bit adicional que dice si esa línea fue alterada en el caché. Decimos que E[LC] es la etiqueta de la línea almacenada en C[LC] y A[LC] dice si fue alterada.
- Al acceder un dato de la línea L, será un acierto en el caché si y solo si E[LC] ≡L, con LC=L%C.
- Si es un desacierto C[LC] contiene M[L'], con L'=E[LC]. Solo si A[LC] es verdadero, se escribe C[L'] en M[L']. En un desacierto se *reemplaza* L' por L escribiendo M[L] en C[LC], L en E[LC] y dejando A[LC] en falso. Esto puede tomar unos 100 ciclos del reloj.
- El acceso se concreta leyendo el dato de C[LC]. Si es una escritura se coloca A[LC] en verdadero. Esto toma unos 3 ciclos del reloj.
- Este caché podría alcanzar un 80 a 90% de aciertos.
- Se le llama caché de acceso directo (*direct mapped*) o de un grado de asociatividad

Diseño de un caché de 256 KB de 4 grados de asociatividad con líneas de 16 bytes

- Se construye a partir de 4 cachés de 64 KB de acceso directo
- Una línea de la DRAM puede estar solo en uno de estos cachés
- Un acceso a la línea L es un acierto si y solo si L es un acierto en alguno de los 4 cachés. La revisión de los 4 cachés se hace en paralelo. Típicamente el acceso toma un ciclo más que en un caché de acceso directo.
- En caso de desacierto, se elige pseudo aleatoriamente uno de los 4 cachés y se hace el *reemplazo* en el caché elegido. Toma unos 100 ciclos del reloj
- La tasa de aciertos en este caché es mejor que en un caché de 256 KB de acceso directo con un área de silicio marginalmente superior, logrando un mejor desempeño promedio
- A mayor tamaño del caché, menor número de desaciertos, pero un solo caché demasiado grande (por ejemplo 512 KB) tiene un tiempo de acceso mayor, exhibiendo un desempeño promedio inferior a un caché de menor tamaño: un cache de 64 KB de 4 grados de asociatividad es mejor que el de 256 KB
- Cuadruplicar el tamaño del caché disminuye los desaciertos solo a la mitad, pero también hace significativamente más lentos los aciertos en el caché

Jerarquía de cachés

- Típicamente se usa una *jerarquía de cachés*: un caché de nivel 1 de 32 KB solo para las instrucciones (L1I) con 3 o 4 ciclos de latencia y otro caché de nivel 1 de 32 KB solo para los datos (L1D). En caso de desacierto se busca en un caché L2 de 512 KB con unos 10 ciclos de latencia y si no está ahí, se busca en un caché L3 de 8 MB con unos 30 ciclos de latencia.
- A igual tamaño de caché, a mayor número de grados de asociatividad, mejor desempeño debido a la disminución de los desaciertos, con un incremento marginal del área y la latencia del caché
- A partir de 16 grados de asociatividad, no se obtiene ninguna mejora en el desempeño
- Para cachés L1 se usan 2 o 4 grados de asociatividad, 8 para L2 y 16 para L3, típicamente
- Cada core tiene su propio caché L1 y L2, pero el caché L3 es compartido por grupos de 4 u 8 cores
- *El conocimiento del funcionamiento del caché permite escribir programas que hagan un uso eficiente de la jerarquía de cachés y la memoria*
- *El diseño del cache en los procesadores modernos es crucial para su buen desempeño*
- *Una alta tasa de aciertos en el caché también contribuye a disminuir el consumo energético porque ir a buscar datos a la memoria consume considerablemente más energía que recuperarlos del caché*