Tema 7: Procesador Central

HISTORIAL DE REVISIONES

NÚMERO	FECHA	MODIFICACIONES	NOMBRE
v1.0.0	2018 September 12		CA

Índice

1. Temario

1. Conjunto de instrucciones
    a. Arquitecturas CISC, RISC y VLIW
    b. Fases de ejecución de una instrucción
    c. Ruta de datos

2. Refs

- Apuntes : Tema 2: Arquitectura von Neumann (unidad de control)
- Libro de Texto: Estructura y Organización de Computadores .William Stalling. Capítulo 12.

3. Introducción

- El objetivo principal de la CPU es la implementación del *ciclo de instrucción*. Es el soporte hardware para poder llevar a cabo todas las operaciones que conllevan las intrucciones de un programa.
- Unidad Central de Proceso (CPU) o Procesador.
    - Nombres: Procesador, microprocesador (El componente electrónico básico es el transistor con un tamaño en sus origenes del orden de la micra), ..
    - Central porque la computadora tiene varios procesadores: Por ejemplo el controlador de la memoria y los controladores de los periféricos.
- Arquitectura Von-Neumann.
    - La CPU es una de las unidades básicas que conforman la arquitectura Von-Neumann (CPU-MP-IO) y Buses.
- *Microarquitectura*
    - Las unidades básicas de la CPU son: Unidad de Control (UC), y la Ruta de Datos (ALU,FPU,MMU,Registros y circuitos de enrutamiento como multiplexores, conmutadores, etc).
- El ciclo de instrucción puede ser secuencial o segmentado, permitiendo el solapamiento en el tiempo de la ejecución de más de una instrucción (técnicas de paralelismo a nivel de instrucción, ILP)
- La CPU se puede ver desde el punto de vista del programador o desde el punto de vista del diseñador electrónico.
- Desde el punto de vista del *programador* interesa conocer:
    - La Arquitectura del Repertorio de Instrucciones (ISA)
    - Registros: registros de propósito general accesibles por el programador (acumulador, registro índice, punteros pila, etc), registro de estado, registros de coma flotante, registros multimedia, resgistros de segmentación de memoria, registros no accesibles como el contador de programa, tamaño de los registros, etc
    - Modos de Funcionamiento de la CPU: modo superusuario, modo usuario, modo interrupción
- Técnicas HW de optimización de la ejecución de un programa **(Performance)**
    - Segmenación-Pipelining: organizar el ciclo de instrucción en fases o segmentos y ejecutarlos en paralelo.
    - Ejecución fuera de Orden OoO: Run time
    - Renombre de Registros: Compiler & Run time
    - Branch Predictor: Run time

4. Conjunto de Instrucciones

4.1. Arquitectura (ISA)

- Recordatorio de la primera parte de la asignatura:
  - Temas: arquitectura von neumann, representación de datos, operaciones aritmetico-lógicas, representación de las instrucciones y programación en lenguaje ensamblador.
- Instruction Set Arquitecture (ISA)
  - La arquitectura del repertorio de instrucciones define: códigos de operación, tipos de operando, modos de direccionamiento, etc
  - Son las instrucciones máquina ejecutables directamente por la CPU en código binario.: *lenguaje máquina*
  - La instrucción a ejecutar está almacenada en código binario en el registro RI de la Unidad de Control.
- El repertorio de instrucciones está especificado en el manual del programador de la CPU:
  - Programamos en *lenguaje Ensamblador* en lugar de en *lenguaje máquina*
  - El manual contiene la definición de la Arquitectura del Repertorio de Instrucciones.
    - el listado y descripción de todas las instrucciones ejecutables por el microprocesador
      - categorías de las instrucciones: transferencia(mov), control(jmpz,loop),aritméticas(add), lógicas(xor), i/o (in/oout) [Error: itemizedlist too deeply nested]
      - Modos de direccionamiento: inmediato, directo, indirecto, desplazamiento
      - Tipos de datos: entero, real, alfanumérico
    - Formatos binarios
      - De las instrucciones: campos de operación, operando, modo direccionamiento
      - De los datos: complemento a 2, coma flotante

4.2. Ejemplos: Intel x86, Motorola 68000, MIPS, ARM

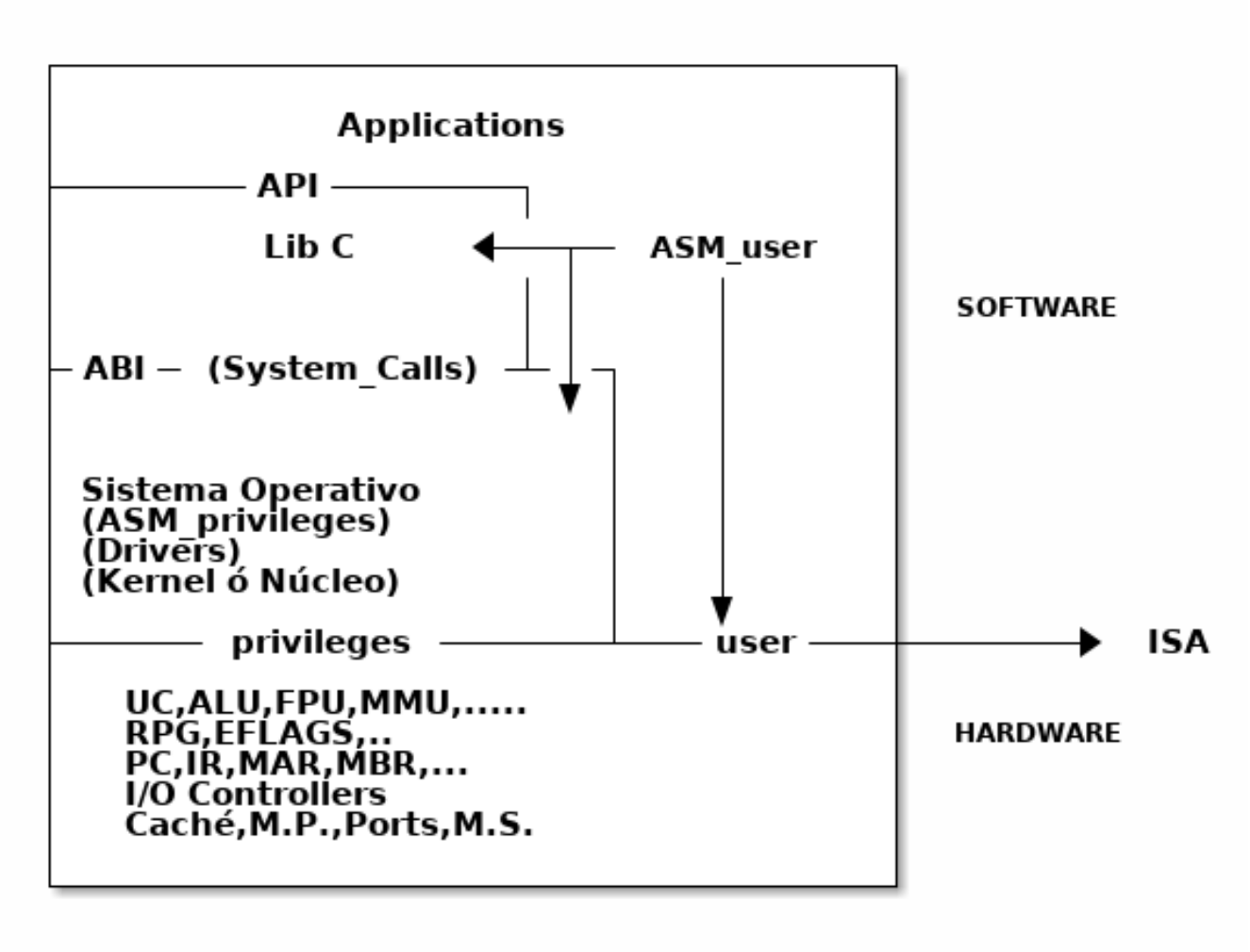
- Ver Apéndice Lenguajes Ensamblador

5. La Computadora desde el punto de vista del programador

5.1. Niveles o Capas de Abstracción

- El programador se abstrae (en parte) de la implementación del Hardware gracias al Sistema Operativo. El programador interactua con el Sistema Operativo para acceder a los recursos HW de la computadora.
- Abstracción de la Máquina : mediante las instrucciones ISA / especificaciones ABI
  - **ABI** : "Application Binary Interface" . Es un documento que especifica las características binarias del software, es decir, el nivel más bajo del software. Por ejemplo especifica el convenio de llamadas a subrutinas, cómo está estructurada la pila, cómo realizar las llamadas al sistema, el formato binario del módulo objeto ejecutable, etc . . . El compilador, linker y loader han de conocer la interfaz ABI con todo detalle.
  - Desde el punto de vista del programador de aplicaciones de bajo nivel: La interfaz con la máquina son las llamadas al sistema (ABI) y el repertorio "user ISA"
- Abstracción a niveles superiores
  - La interfaz con la librería son los prototipos de las funciones de la librería (Application Programming Interface - **API**)

- Esquema con las Interfaces de las aplicaciones desarrolladas en **lenguajes de bajo nivel**:



  - Desde el punto de vista del sistema operativo S.O.:
    - La interfaz con la máquina es **ISA (system isa y user isa)**
    - La interfaz con el programador es **ABI**
  - Desde el punto de vista del programador
    - si no hay S.O. la interfaz con la máquina será equivalente a la del S.O.
    - si hay S.O. y librerías la interfaz con la máquina:
      - ◇ en lenguaje C : **API** y **ABI** específicos de C
      - ◇ en lenguaje ASM: **API ©** y **ABI** específico de asm
- La programación de bajo nivel requiere tener algunos conocimientos del Hardware de la máquina no siendo posible su completa abstracción. Por lo tanto es necesario estudiar la CPU desde el punto de vista del programador.
- ¿Cual sería el esquema de niveles o capas visto por los siguientes niveles de abstracción superiores?
  - Escritorio
  - Lenguaje de Programación Java

5.2. Compatibilidad Software

5.2.1. Compatibilidad

- Cada procesador tiene su repertorio de instrucciones
- Si dos procesadores tienen el mismo repertorio de instrucciones, es decir, la misma arquitectura, el módulo fuente en lenguaje ensamblador será compatible para los dos procesadores aunque la estructura interna de la CPU sea diferente: Ejemplo: Intel IA64 y AMD64

5.2.2. Ejemplos

- El programador necesita conocer el trío ARCH-KERNEL-LIBC
  - Arch se refiere a la arquitectura de la computadora → ISA
  - Kernel: núcleo del sistema operativo. Implementa las llamadas del sistema
  - Libc: librería para el programador de aplicaciones. Implementa las llamadas al sistema
  - Tanto el Kernel como la Librería tienen asociados sus interfaces de nivel alto (API) como de nivel bajo (ABI)
- Ejemplos arch/kernel/libc
  - amd64-linux-gnu
  - arm-linux-gnueabi

6. Fases de Ejecución de una Instrucción

6.1. Estructura

./images/von_neumann/ias_architecture.png

Figura 1: IAS_Architecture

6.2. Ciclo / Diagrama / Fases

./images/cpu/12_5.jpg

Figura 2: Diagrama de Estados

- Fases del ciclo de instrucción
  1. Fetch Instruction : FI
     - Inicialmente hay que volcar al bus de direcciones de memoria el contenido del Contador de Programa (PC)
     - Captar la instrucción
     - PC ← PC+1
  2. Instruction Decode : ID
     - interpretar la instrucción
  3. Fetch Operand : OF
     - captar datos, captar los operandos

- resolver la dirección efectiva

4. Execute Instruction : EI
   - procesar la instrucción con los datos
5. Write Operand: WO
   - almacencar el resultado
   - resolver la dirección efectiva
6. Interruption : II
   - El programa puede ser interrumpido por la prioridad de ejecutar otro programa de atención a periféricos, etc..Una vez atendida la interrupción el programa continua con el siguiente ciclo de instrucción.
7. Next Instruction : NI

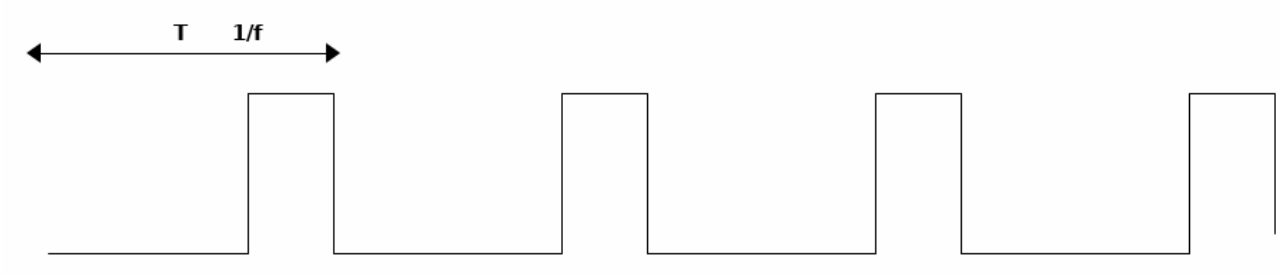
- Ciclo de instrucción
  - Después de la fase de captación de la instrucción (FI) le sigue la fase de Ejecución (EI) ó la Fase de determinación de la Dirección Efectiva del Operando y Obtención del operando (OF)
  - Después de la fase de ejecución puede haber un ciclo de atención a una interrupción.

6.3. Ejemplo: máquina IAS de Von-Neumann

- Tema 2: Arquitectura Von-Neumann
- Cada instrucción de la computadora IAS se ejecuta siguiendo una secuencia de fases. Dicha secuencia se repite para cada instrucción y se conoce como el ciclo de instrucción de la unidad central de proceso (CPU).
- La unidad de control es la unidad de la CPU que implementa cada fase del ciclo de instrucción.
- La unidad de control controla la ruta de datos de la CPU mediante microordenes.
- Internamente está formada por el circuito generador de microordenes y por los registros : contador de programa y registro de instrucción.

6.3.1. Diagrama de Microoperaciones

- Microoperaciones: operaciones realizadas por la CPU internamente, al ejecutar una Instrucción Máquina.
  - Ejemplos: escribir en el registro MAR, orden de lectura a la MPrincipal, leer de MBR, interpretar de IR, incrementar PC, etc
  - Ejecución Síncrona con el reloj de la CPU:



    - Flancos de reloj: Cambio de nivel 0→1 (positivos) o 1→0 (negativos)
    - IAS no es síncrona: una microoperación no comienza con ningún patrón de tiempos.

- Descripción de las micro-operaciones: Register Transfer Language (RTL)

./images/von_neumann/ias_operation.png

Figura 3: IAS Operation

- Operación de la máquina IAS:
  - El ciclo de instrucción tiene dos FASES
  - La primera fase es común a todas las instrucciones.
- Ejemplos de instrucciones
  - X: referencia del operando
  - AC ← M(X)
  - GOTO M(X,0:19): salto incondicional a la dirección X. X apunta a dos instrucciones. X,0:19 es la referencia de la Instrucción de la izda.
  - If AC>0 goto M(X,0:19): salto condicional
  - AC ← AC+M(x).

7. Microarquitectura: Unidades Funcionales

7.1. Introducción

- Se conoce con el nombre microarquitectura a la arquitectura interna del microprocesador.
  - La microarquitectura es el diseño e implementación del ciclo de instrucción del conjunto de instrucciones definido por ISA.
  - Ejemplos
    - Intel: 8051, x86
    - AMD: x86
    - ARM: Cortex

7.2. Implementación del ciclo de instrucción

- ¿Cómo implementar el ciclo de instrucción?
  - Mediante un Circuito Electrónico Digital secuencial: Máquina de estados finitos FSM que implementa la secuencia del diagrama de estados y que recibe el nombre de Unidad de Control.
  - La Unidad de Control es una secuencia de estados que van realizando las distintas fases del ciclo de instrucción.
  - Las distintas fases del ciclo de instrucción utilizan distintas unidades funcionales como: registros, ALU, etc
  - La interpretación de distintas instrucciones máquina dará lugar a diferentes secuencias de estados en la Unidad de Control.

7.3. Estructura de la CPU

- Tres recursos básicos: Unidad de Control, **Unidad de Ejecución** y Registros.
- Dos Bloques básicos de la CPU
  - Unidad de Control (UC) y la Ruta de Datos (DataPath).
- La unidad de control esta formada por

- generador de las microoperaciones que implementan el ciclo de instrucción
- registros: registro de instrucción IR, registro contador de programa PC

- La Ruta de Datos esta formada por
  - **Unidad de Ejecución UE :**
    - Unidad Aritmetico Lógica ALU: cálculos números enteros
    - Unidad de Punto Flotante FPU: cálculos números reales
    - Unidad Load/Store LSU: cálculos de la dirección efectiva y acceso a la memoria principal
      - Memory Management Unit (MMU): cálculo de la dirección efectiva FISICA de la MP. Traduce las direcciones virtuales de memoria utilizadas por la cpu en direcciones físicas de la memoria principal.
  - los Registros
    - Registros de propósito general GPR accesibles por el programador
    - Registros de estado SR

7.4. Fase de Captación

- Ejemplo: Microoperaciones de la Fase de captación del ciclo de instrucción.
  - Se realiza la lectura de una instrucción mediante las siguientes acciones que son activadas por la Unidad de control:
    - El Contador de Programa (PC) o Instruction Pointer (IP) contiene la dirección de referencia de la instrucción a captar
    - El Memory Address Register (MAR) se carga con el contenido del (PC)
    - El bus de direcciones del sistema se carga con el contenido de MAR
    - Se vuelca el contenido de la dirección apuntada al Buffer i/o de memoria, de ahí al bus de datos transfiriendose así al Memory Buffer Register (MBR)

./images/cpu/12_6.jpg

Figura 4: Flujo de Datos. Ciclo de Captación

- Secuencia de las microordenes en el ejemplo:
  a. MAR $\rightarrow$ address bus
  b. UC $\rightarrow$ control bus
  c. data bus $\rightarrow$ MBR
  d. MBR $\rightarrow$ IR y UC $\rightarrow$ PC
  e. al finalizar la ejecución: PC $\rightarrow$ MAR

7.5. Perspectiva de la CPU

- Divimos la CPU en 5 unidades:
  - Unidad de Control (UC)
  - Unidad de Ejecución (UE)
  - Registros : de Propósito General (rax,mmx,sse,xmm,..), control (usuario,superusuario,paginación,interrupción,...) y status (rflags, ..).
    - Los registros de control no son accesibles por el usuario, son accesibles por el sistema operativo.
  - Memoria Cache L0
  - Memory Management Unit (MMU)
  - Reloj para sincronizar las tareas: facilita el diseño del Hardware.

7.5.1. Unidad de Control

- The control unit (sometimes called the fetch / decode unit) is responsible for retrieving individual instructions from their location in memory, then translating them into commands that the CPU can understand. These commands are commonly referred to as machine-language instructions, but are sometimes called **micro-operations**, or UOPs. When the translation is complete, the control unit sends the UOPs to the execution unit for processing.
- Señales de control de la UC
  - Señales digitales binarias

7.5.2. Unidad de Ejecucion (EU)

- The execution unit is responsible for performing the third step of the instruction cycle, namely, executing, or performing the operation that was specified by the instruction.
- Incluye: ALU+FPU+LSU+RPG
  - Operaciones: Aritméticas, Lógicas, Transferencia,

7.5.3. Ruta de Datos

- Es la ruta que realizan los datos (instrucciones, campos del formato de instrucciones, operando, dirección, etc . . .) a través del procesador, internamente al procesador, dirigidos por la Unidad de Control.
- Es necesario interconectar las distintas unidades y subunidades de la CPU para poder transferir y procesar los bits y conjuntos de bits entre ellas.
- Los microcomandos de la UC en forma de señal transportan y procesan dichos datos.
  - Ejemplos de microcomandos: abrir puerta, conectar bus, multiplexar datos, etc . . . microordenes de control del hardware
  - Dicho transporte y procesamiento dependerá de la interpretación de la instrucción en ejecución y del diseño de la microarquitectura.
- Los componentes básicos de la Ruta de Datos son :
  - Unidades de transporte: BUS, conmutador, multiplexor, etc
  - Unidad de memoria: cálculo de la dirección efectiva, interfaz con la memoria externa
  - Unidades de procesamiento: ALU
  - Unidades de almacenamiento: registros
- RTL: Register Transfer Language
  - Lenguaje para indicar las acciones de transporte, procesamiento y almacenamiento.
    - AC $\leftarrow$ [PC]+ M[CS:SP]
- Esquema de la Ruta de Datos

./images/cpu/datapath.jpg

Figura 5: Datapath

  - Líneas gruesas: bus de datos
  - Líneas finas: bus de control $\rightarrow$ chip select, microorden sumar, cargar registro, etc ..
- Ver *applet* de la ruta de datos del apartado Imágenes

- Diseño del datapath
  - determinar que microunidades son necesarias
  - cómo conectarlas
  - Qué microseñales accionar y cuándo en cada microoperación. Paralelismo a nivel de microoperaciones
  - ubicación y temporización de los datos según la secuencia del diagrama de estados de la UC
    - AC ← [PC]+ M[CS:SP] ⇒ microoperaciones asociadas y diagrama de tiempos

7.6. Unidad de Control Microprogramada

- Unidad de Control Microprogramada vs Cableada
- Microcableada: El secuenciados o FSM de la unidad de control ejecuta *directamente* las instrucciones en código máquina almacenadas en la memoria principal
- Microprogramada:
  - Las instrucciones máquina (ISA) almacenadas en la memoria principal y cuya secuencia consituye el **código máquina** del programa del usuario no son ejecutadas directamente por la UC. En su lugar cada instrucción en código máquina es traducida en una secuencia de **microinstrucciones** y cada microinstrucción genera las microoperaciones o microseñales de la unidad de control que conforman el ciclo de instrucción.
  - La secuencia de microinstrucciones asociadas a una microinstrucción constituye el **microcódigo** que se encuentra almacenada en una memoria de sólo lectura (Read Only Memory ROM) interna de la Unidad de control.
  - Cambiando o añadiendo microcódigo a nuestra Unidad de Control conseguimos nuevas arquitecturas ISA de una manera más flexible que con la unidad de control cableada.
    - microcode

8. Arquitecturas CISC/RISC

8.1. Introducción

- CISC: Complex Instruction Set Computer
- RISC: Reduced Instruction Set Computer
- CISC y RISC son dos filosofías de diseño de un computador, dos arquitecturas.

8.1.1. CISC

- Ejemplos: Motorola 68k, Intel x86.
- Instrucciones de varios bytes y no uniformes.
- Necesita un HW complejo que ocupa mucho espacio y necesita muchos ciclos de reloj.
- La arquitectura del lenguaje ensamblador está próxima a un lenguaje de alto nivel cómo el lenguaje C por lo que facilita la tarea a los compiladores y a los programadores de lenguaje ensamblador.
- En cambio complica el diseño e implementación de elementos hardware como la CPU.

8.1.2. RISC

- Ejemplos: PowerPC, ARM, MIPS and SPARC
- Apuesta por un Hardware sencillo por lo que las instrucciones han de ser sencillas, regulares.
  - Un HW sencillo es rápido y ocupa poca área del chip.
- Inconveniente: Gran número de accesos a memoria para capturar las instrucciones, los operandos y el resultado.
  - Solución: incrementar la memoria interna: el número de Registros internos y la memoria caché. Para lo cual hay espacio debido al HW sencillo.

8.1.3. Cuestiones

- Qué arquitectura optimiza el tamaño de bytes del programa
- Qué arquitectura optimiza el tiempo de ejecución del cada instrucción
- Qué arquitectura optimiza el tamaño y coste de fabricación de la CPU
- Qué arquitectura optimiza el consumo
- Qué arquitectura optimiza el número de capturas a memoria. ¿Existe indepencia entre captura y ejecución de instrucciones?

8.1.4. SW

- Un programa ensamblador de una arquitectura RISC tiene más instrucciones que un CISC
- Cada instrucción RISC se ejecuta en menor tiempo que una CISC.

8.2. Tabla Comparativa

- RISC vs CISC

9. Instruction Level Parallelism (ILP)

- wikipedia
  - Instruction-level parallelism (ILP) es la medida de cuantas instrucciones de un programa pueden ser ejecutadas simultáneamente. El solapamiento de la ejecución de las instrucciones recibe el nombre de instruction level parallelism (ILP)
  - Son dos los mecanismos para conseguir el ILP
    - Hardware
    - Software
- Técnicas de diseño de microarquitecturas que persiguen un solape ILP
  - VLIW
  - Superscalar
  - Pipelining (Segmentación)
  - Out-of-order execution
  - etc

9.1. VLIW vs Superscalar

9.1.1. VLIW

- Very Long Instruction Words
- La CPU contiene múltiples Unidades de Ejecución
- Una palabra contiene tantas instrucciones como unidades de ejecución.
  - A la palabra se le denomina Instruction Word, la cual contiene múltiples instrucciones máquina.
  - El *compilador* crea las Instrucciones Word con las múltiples instrucciones **asignando** a cada una de ellas una Unidad de Ejecución distinta.
    - Múltiples Instrucciones en Paralelo

9.1.2. Superscalar

- La arquitectura superescalar significa que la CPU tiene múltiples Unidades de Ejecución (UE), no confundir con múltiples núcleos (core), y es la *propia CPU* la que **asigna** en tiempo de ejecución los recursos de la máquina a las distintas instrucciones .
- Dicha arquitectura permite la ejecución simultánea de múltiples instrucciones.
- Una CPU superscalar n-way significa que puede ejecutar simultáneamente n instrucciones.
- Superscalar no significa multinúcleo. Un único núcleo es superscalar.

9.1.3. Comparativa Superscalar-VLIW

- One of the great debates in computer architecture is static vs. dynamic. **static** typically means "let's make our compiler take care of this", while **dynamic** typically means "let's build some hardware that takes care of this". Each side has its advantages and disadvantages. the compiler approach has the benefit of time: a compiler can spend all day analyzing the heck out of a piece of code. however, the conclusions that a compiler can reach are limited, because it doesn't know what the values of all the variables will be when the program is actually run. As you can imagine, if we go for the hardware approach, we get the other end of the stick. there is a limit on the amount of analysis we can do in hardware, because our resources are much more limited. on the other hand, we can analyze the program when it actually runs, so we have complete knowledge of all the program's variables.
- **VLIW** approaches typically fall under the "static" category, where the compiler does all the work.
- **Superscalar** approaches typically fall under the "dynamic" category, where special hardware on the processor does all the work. consider the following code sequence:

```
sw $7, 4($2)
lw $1, 8($5)

$ significa direccionamiento directo
() direccionamiento indirecto indexado
```

- suppose we can run two memory operations in **parallel** [but only if they have **no dependencies**, of course]. are there dependencies between these two instructions? well, it depends on the values of $5 and $2. if $5 is 0, and $2 is 4, then they depend on each other: we must run the store before the load.
  - in a VLIW approach, our compiler decides which instructions are safe to run in parallel. there's no way our compiler can tell for sure if there is a dependence here. so we must stay on the safe side, and dictate that the store must always run before the load. if this were a bigger piece of code, we could analyze the code and try to build a proof that shows there is no dependence. [modern parallelizing compilers actually do this!]

- if we decide on a SUPERSCALAR approach, we have a piece of hardware on our processor that decides whether we can run instructions in parallel. the problem is easier, because this dependence check will happen in a piece of hardware on our processor, as the code is run. so we will know what the values of $2 and $5 are. this means that we will always know if it is safe to run these two instructions in parallel.
  - Hopefully you see some of the tradeoffs involved. dynamic approaches have more program information available to them, but the amount of resources available for analysis are very limited. for example, if we want our superscalar processor to search the code for independent instructions, things start to get really hairy. static approaches have less program information available to them, but they can spend lots of resources on analysis. for example, it's relatively easy for a compiler to search the code for independent instructions.

9.2. Pipeline (Segmentacion)

- Pipeline: cauce o tubería.
- Ejemplo de Lavado de coches
  - Fases: Humedecer - Enjabonar - Cepillar - Aclarar - Secar - Abrillantar
- Máquina Secuencial
  - Cola de coches ante la máquina
  - Si un coche está en cualquiera de las fases no entra el siguiente coche.
  - El intervalo de tiempo de salida de coches será la suma de todas las fases. ¿Cada cuanto tiempo sale un coche del lavadero?
  - **Througput (Producción)**: Número de coches de salida por unidad de tiempo
- Segmentación frente a Secuencial.
  - En lugar de tener una máquina que realice todas la fases tenemos máquinas independientes que realizan cada fase.
  - El intervalo de tiempo de salida de coches será el de la duración de la fase de mayor duración.
  - El througput, del número de coches atendidos por unidad de tiempo, aumenta.
- Flujo de Instrucciones con segmentación en 2 etapas

./images/cpu/12_9.jpg

Figura 6: Segmentación en 2 etapas

- En caso de que los tiempos de cada etapa sean distintos o halla penalización por saltos en el flujo , se producirán tiempos de espera.

./images/cpu/12_10.jpg

Figura 7: Diagrama de tiempos con segmentación de 6 etapas

./images/cpu/12_11.jpg

Figura 8: Diagrama de tiempos. Salto incondicional

./images/cpu/12_12.jpg

Figura 9: Flujo de instrucciones con segmentación de 6 etapas

10. Ejercicios

- Capítulo 12 del libro de texto William Stalling.
- Capítulo 13 del libro de texto William Stalling

11. Imagenes

- Imagenes