Introducción a la Arquitectura y Organización de Computadores

Alejandro Furfaro

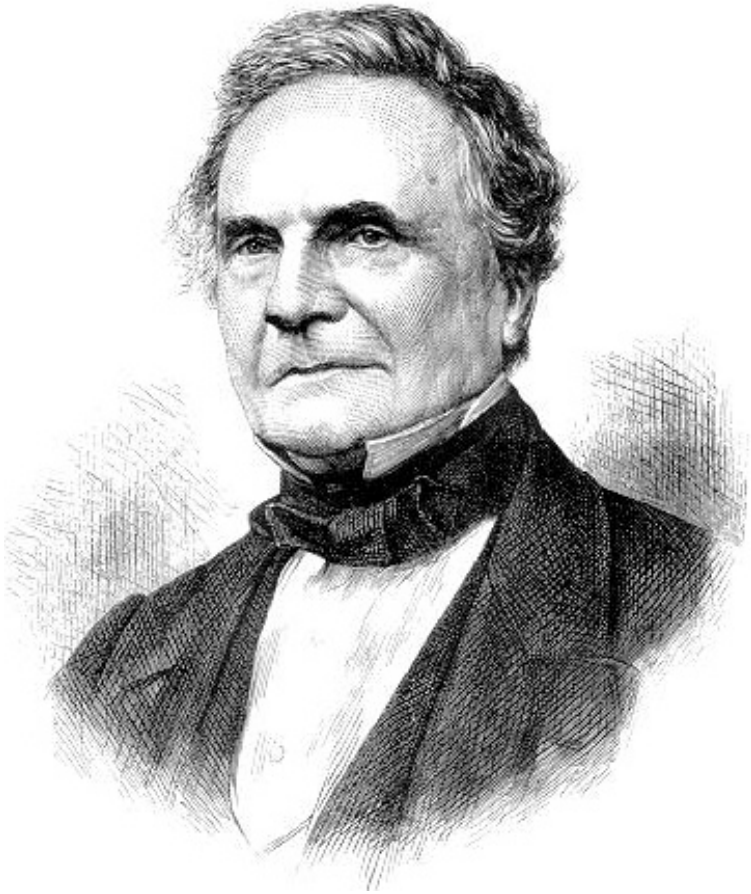
23 de marzo de 2020

Agenda

¿Quien diseñó el primer computador de propósito general?

- ¿Fue Alan Turing durante los años 30?
- ¿Fue Von Newmann en los años 40?
- ¿Fueron los alemanes durante la 2da. Guerra Mundial?
- .... Frio ....

Fue en el sigo XIX....



Charles Babbage 26/12/1791 - 18/10/1871

Máquina Analítica (1837)

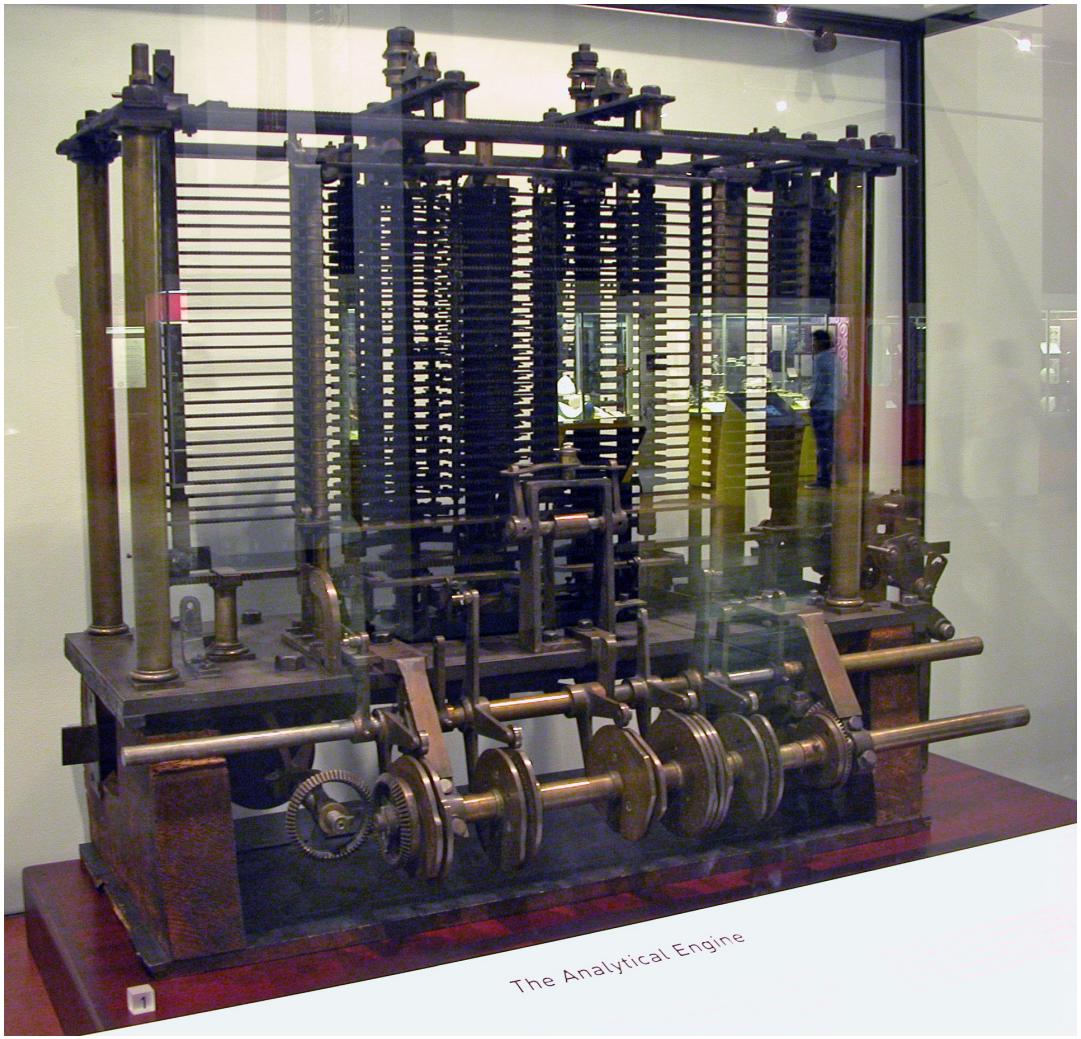


Figura: Analitical Machine. Foto: ©De Bruno Barral (ByB), CC BY-SA 2.5, https://commons.wikimedia.org/w/index.php?curid=6839854

Diferential Engine (14 de Junio de 1821)

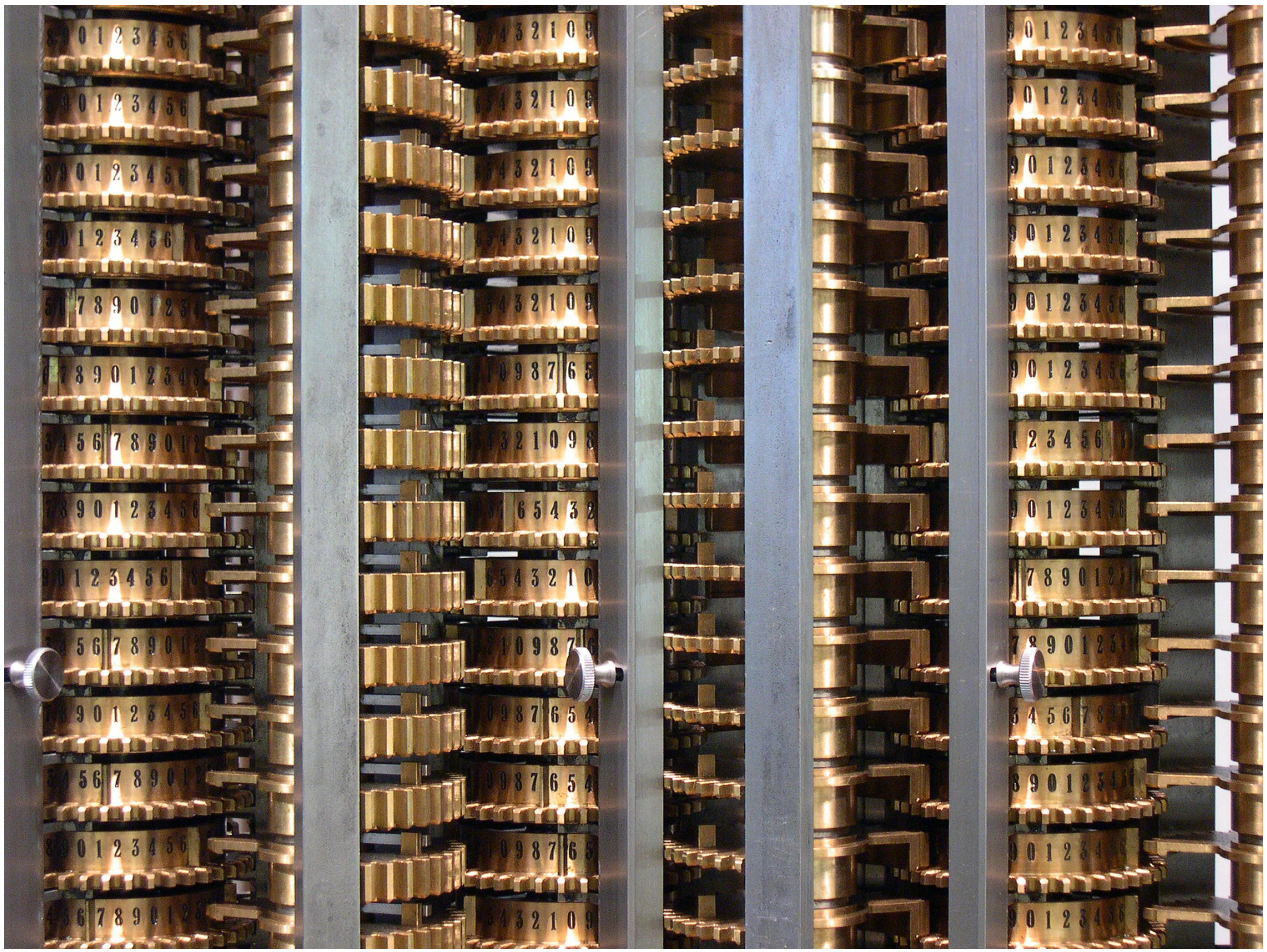


Figura: Diferential Engine. Foto: ©De Carsten Ullrich - Trabajo propio, CC BY-SA 2.5, https://commons.wikimedia.org/w/index.php?curid=851017

Z3: (1941) Primer Computador Programable

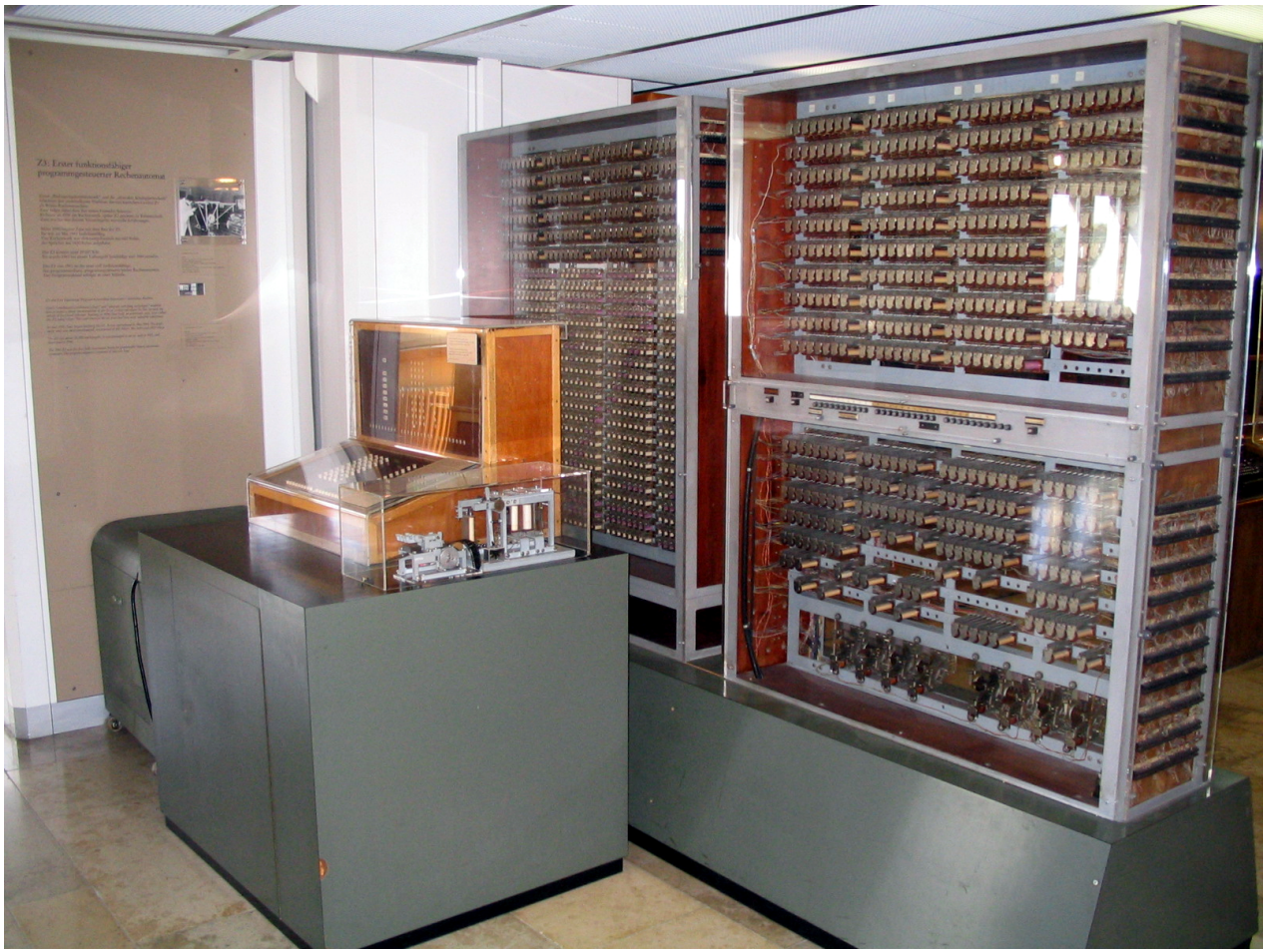


Figura: Réplica del Computador Z3. Foto: ©De Venusianer, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=3632073

La Máquina de Turing

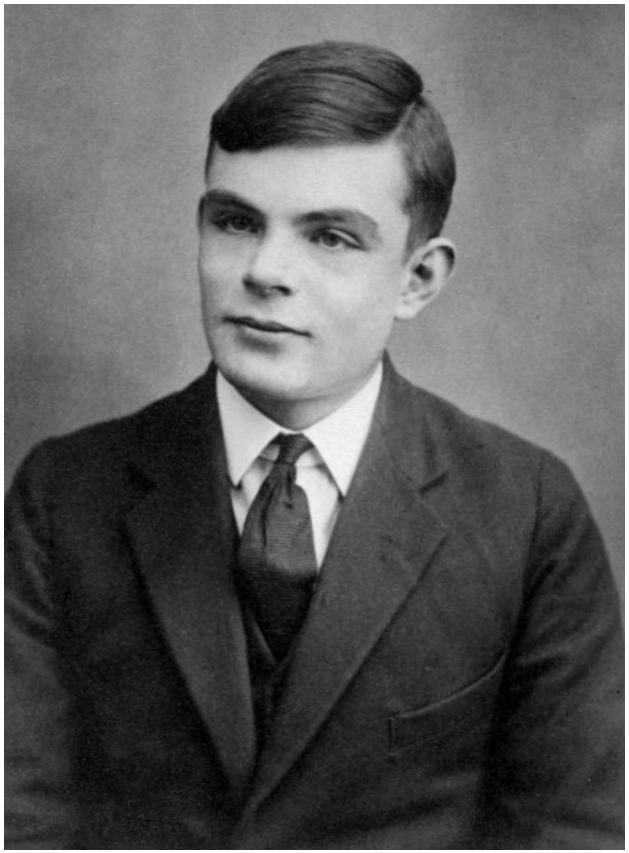


Figura: Alan Turing. Foto: ©De Desconocido - http://www.turingarchive.org/viewer/?id=521&title=4, Dominio público, https://commons.wikimedia.org/w/index.php?curid=22828488

- Hasta que Alan Turing enunció en 1936 su modelo teórico conocido como Máquina de Turing, no se disponía de un marco formal para trabajar en ciencias de la computación.
- La Máquina de Turing es un modelo matemático implementable mediante un autómata, que tiene la capacidad de implementar cualquier problema matemático, que pueda ser expresado a través de un algoritmo.

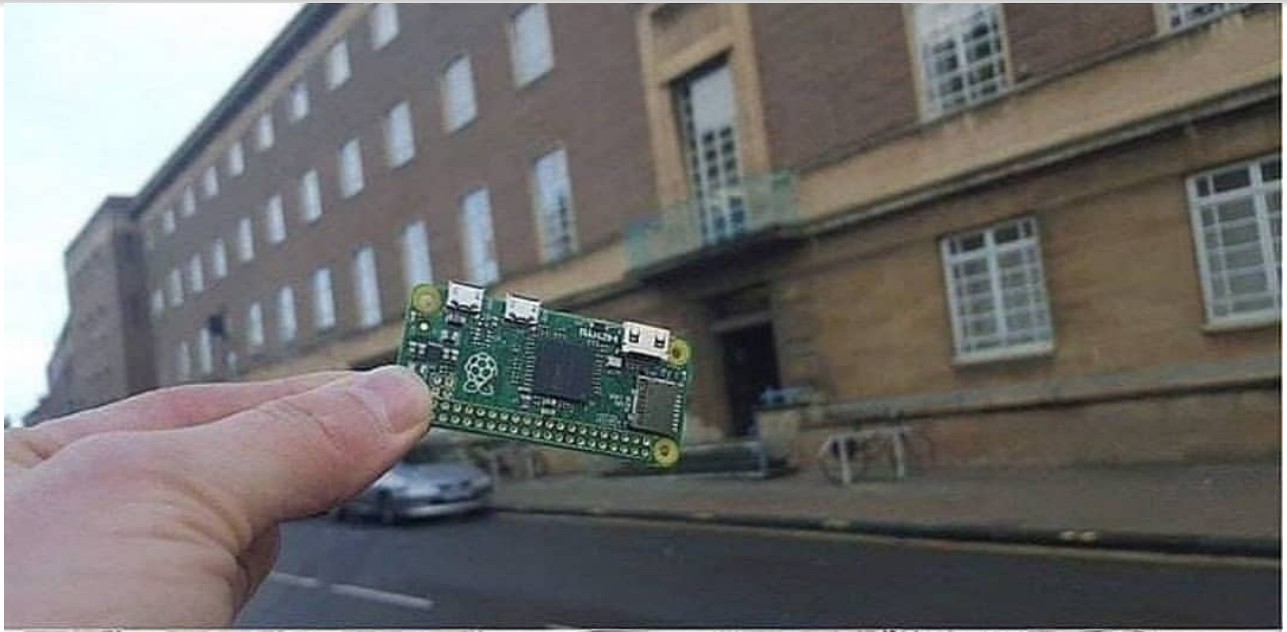
Componentes La Máquina de Turing

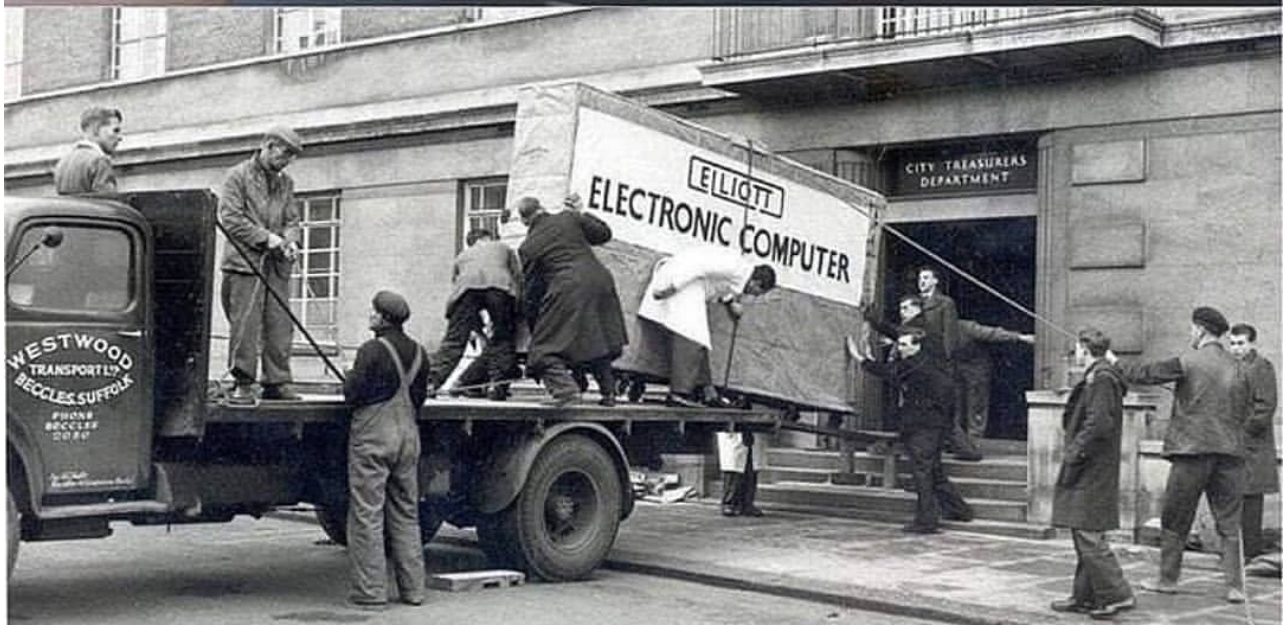
- Una cinta de longitud infinita dividida en celdas contiguas
- Un cabezal que puede avanzar o retroceder la cinta en un paso discreto (un paso = una celda), y que además puede leer o escribir en la cinta.
- Un registro de estado, que describe el estado en que se encuentra la máquina. (Turing lo asimilaba al estado mental de una persona cuando realiza un cálculo mental)
- Una tabla de instrucciones, que permite en función del valor leo y eventualmente de alguna acción ingresada por el usuario escribir un resultado, avanzar, la cinta, retroceder la cinta, dejar la cinta en su mismo lugar, y actualizar el nuevo estado en el registro de estados.

Alan Turing

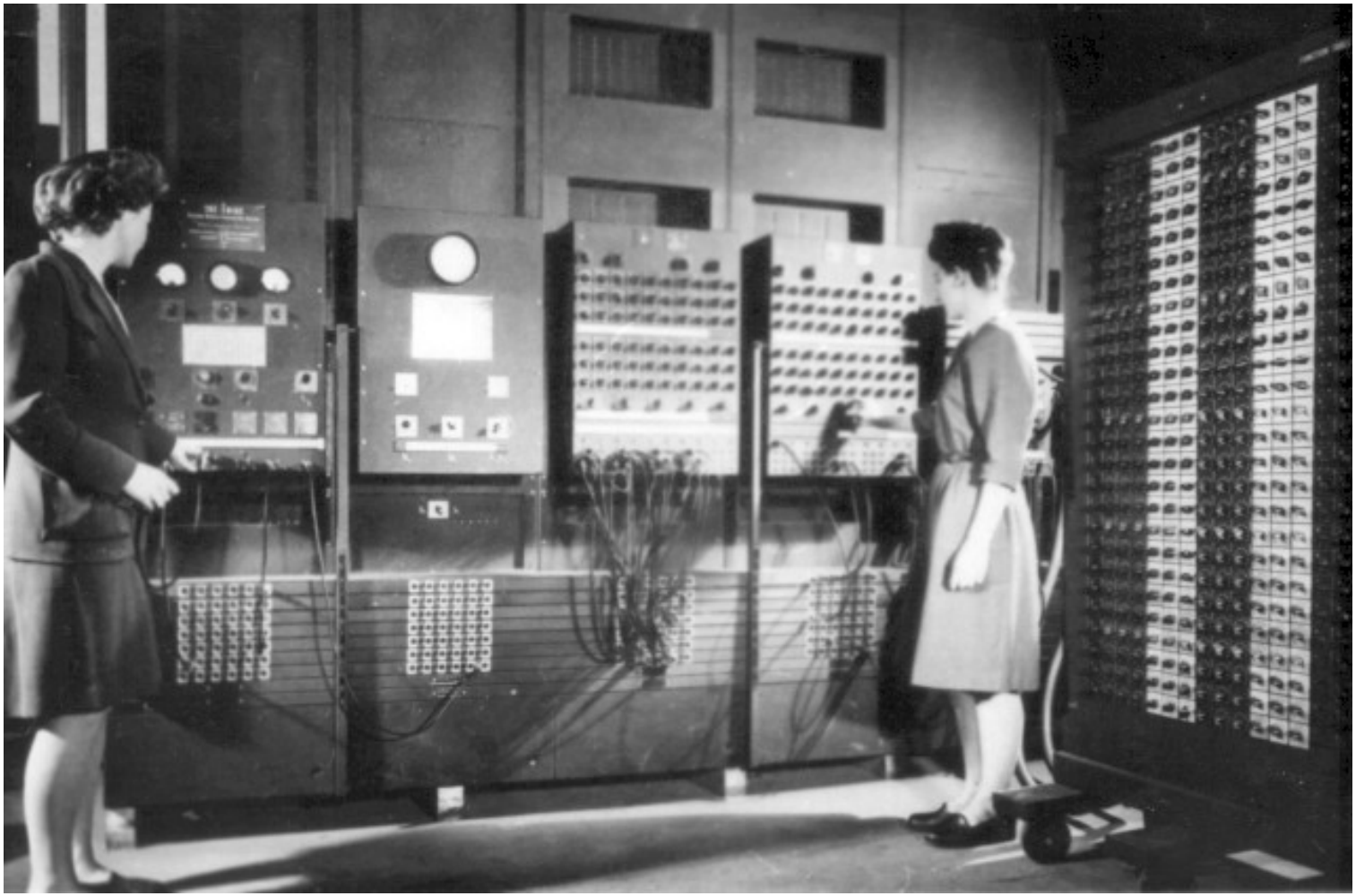
- De éste modo, Turing proporcionó la descripción matemática de un dispositivo muy simple con capacidades de cómputo arbitrarias, capaz de probar capacidades de computabilidad tanto generales como particulares.
- La *Completitud Turing* de un sistema de instrucciones es la medida de su capacidad de simular una máquina de Turing.
- En la actualidad los lenguajes de computación utilizados son Turing Completos siempre que no se considere que ejecutan sus programas en memoria finita.

En solo algo menos de 80 años





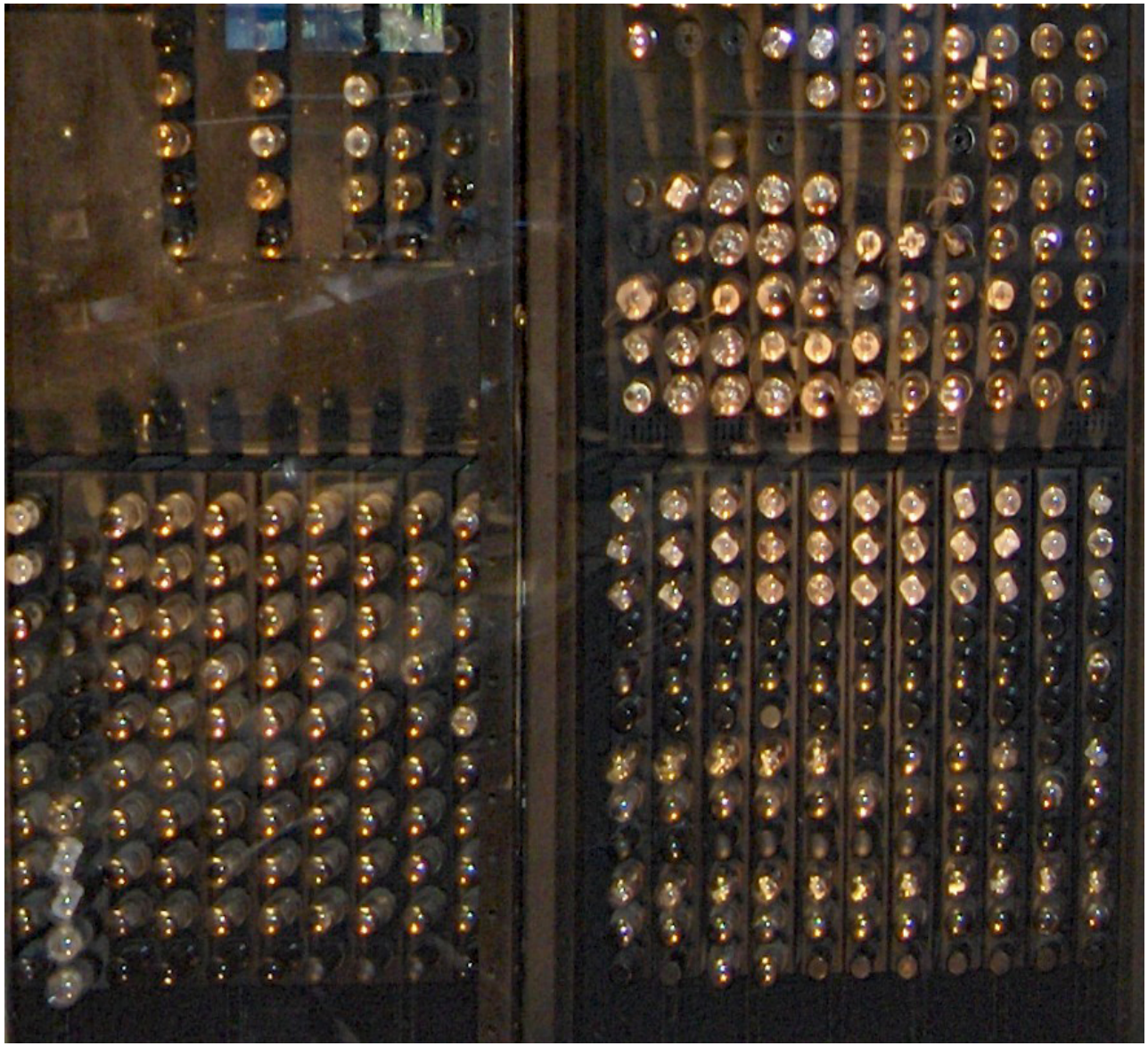
ENIAC “primer”Computadora de Propósito general



ENIAC “primer”Computadora de Propósito general

- Tal como dijimos la Z3 en Alemania fue algo anterior,
- Pero fue destruida por los aliados en el bombardeo de Berlín en 1943,
- A poco de terminar la guera aparece ENIAC y, como la historia la escriben los que ganan.... está considerada la primera.
- En rigor es la primer computadora Electrónica. Eso si: **E**lectronic **N**umerical **I**ntegrator **A**nd **C**omputer.
- No obstante, no utiliza programa almacenado. Hay que recablear todo para cambiar el programa.
- Utilizaba 6000 interruptores para programarse.
- 17468 valvulas, 7200 diodos, 10000 condensadores, 1500 relés, 70000 resistencias, y 5 millones de soldaduras.
- Consumo 160 KW, 27 Toneladas, 167 m^2

ENIAC BackPlane. Vávulas termoiónicas



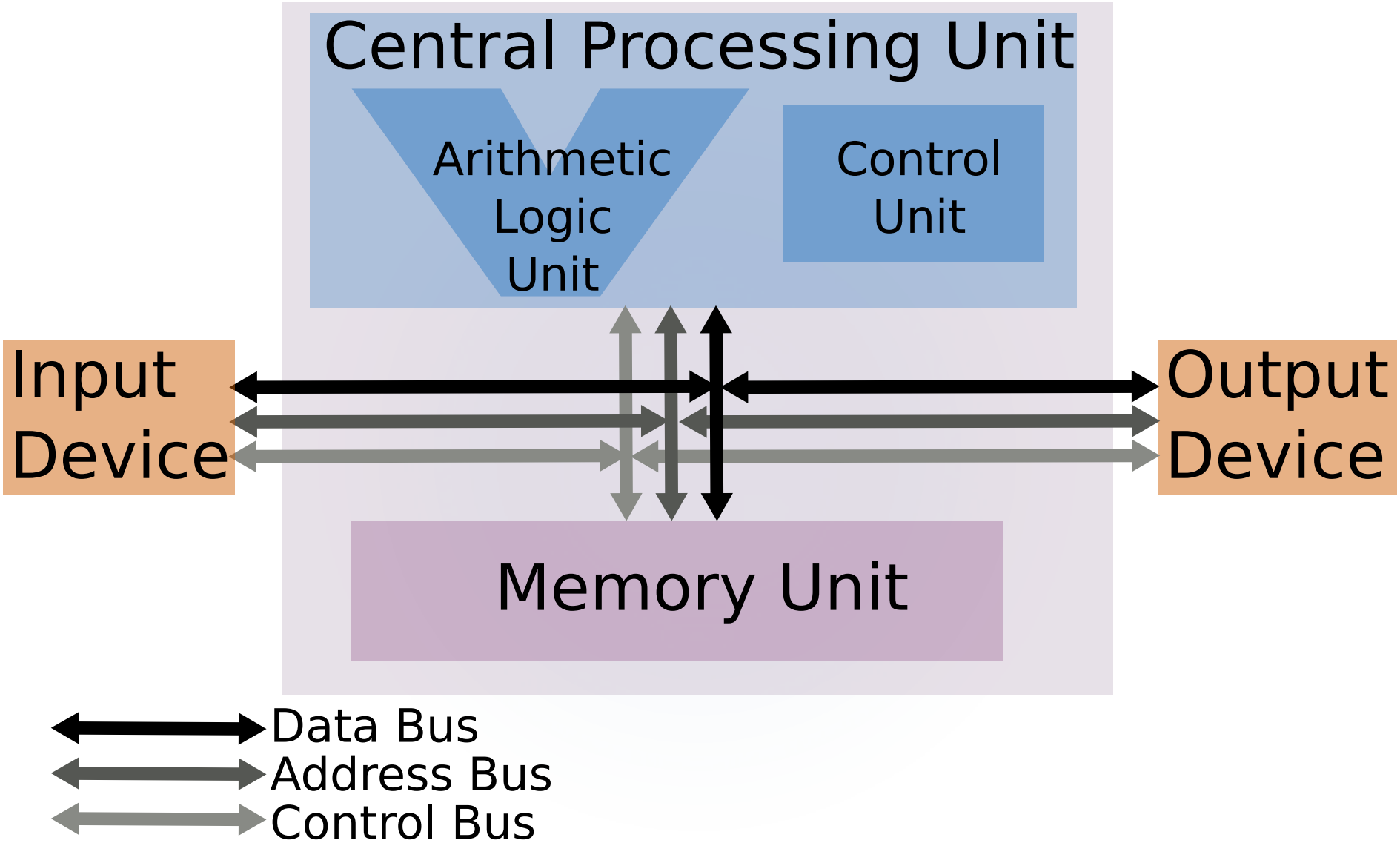
Modelo de Von Mewmann

- 1945, “First Draft of a Report on the EDVAC. John Von Newmann”.
- Este reporte borrador describe la organización y arquitectura de una máquina de cómputo binaria y electrónica en la que se detallan las características y recursos de arquitectura de los componentes principales.
- Tal vez su mérito principal sea el de haber sido la primer publicación de la arquitectura y organización de un computador basado en los postulados de Turing de una década antes. Obviamente la **EDVAC** (por **E**lectronic **D**iscrete **V**ariable **A**utomatic **C**omputer) era una máquina Turing completa.

Modelo de Von Mewmann

- Inaugurando el lema “publish or perish”, esta arquitectura por ser la primera bien documentada, se convirtió en sinónimo de programa almacenado (recordemos que la máquina de Turing había postulado las bases teóricas de este concepto 9 años antes).
- De modo que sin perjuicio de los valiosos aportes de Von Newmann, estos conceptos salieron de otra mente brillante: la de Alan Turing.
- No obstante, los avances relativos de la EDVAC debidos a los trabajos de Von Newmann, y su publicación en el trabajo antes mencionado fue fundamental para la difusión del modelo de programa almacenado.
- Esta publicación fue muy influyente durante décadas en el diseño y concepción de computadores, y éste es su significativo aporte.

Modelo de Von Mewmann



Modelo de Von Mewmann

Esta organización fue dominante durante la primer y segunda generación de Computadores

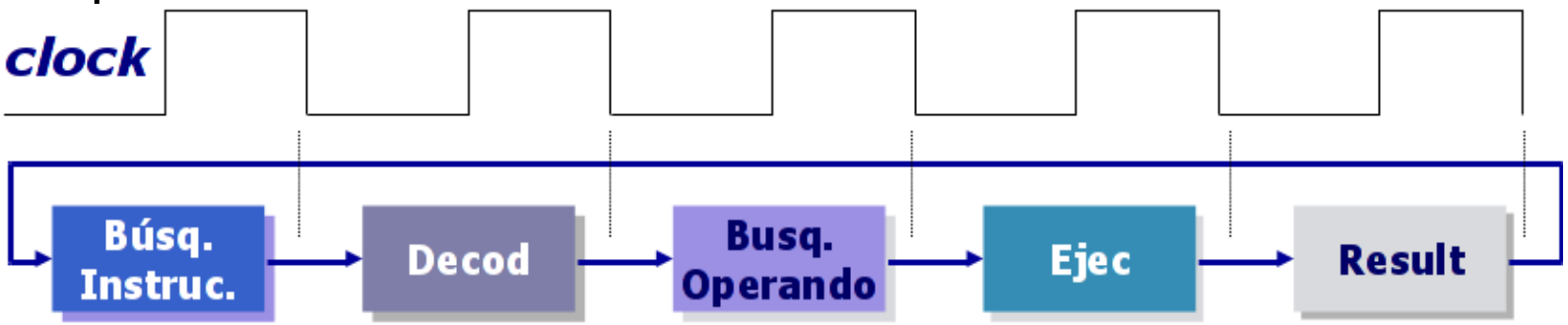
- ***Central Arithmetic (CA)***: Una Unidad de Procesamiento Aritmético Lógica y Registros.
- ***Central Control (CC)***: Implementación de la máquina de estados, con registro de instrucciones y registro Contador de Programa.
- ***Memory (M)***: Almacena el código y los Datos. Dos cuestiones: (1) Esta idea ya estaba en la **ENIAC** y en otros diseños anteriores, y (2) Con el tiempo se advirtió que este escenario limita la performance ya que no permite leer una instrucción y un dato a la vez, escenario conocido como “Von Newmann bottleneck”.

Modelo de Von Mewmann - Resumen

- ***Entrada (I)***: Se trata de mecanismos para que se puedan ingresar al computador comandos y datos desde el exterior.
- ***Salida (O)***: Se trata de mecanismos para que el computador pueda enviar resultados hacia el mundo exterior.
- ***Memoria Externa (R)***: Se trata de un medio de almacenamiento de mayor capacidad que la Unidad de Memoria en donde solo están el código y los datos del programa en ejecución. En la Memoria externa podemos tener una colección de programas y datos para copiar de a una vez en la Unidad de Memoria y lanzar a ejecutar. Es lo que hoy conocemos como Storage. En ese entonces se trataba de Cintas perforadas por ejemplo.

Modelo de Von Meumann

1. Modelo de programa almacenado.
2. Datos y programa en el mismo (único) banco de memoria.
3. Máquina de estados



4. No está claro cual fue la primera implementación, pero Manchester Baby, IBM-SSEC, EDVAC, BINAC, entre otras fueron de las primeras (alrededor de 1948 todas ellas)

Modelo o Arquitectura Harvard

- En 1945, cuando Von Newmann publica el reporte de la EDVAC, tal vez era difícil imaginar que en solo cuestión de algo mas de dos décadas alguien mas iba a intentar leer desde memoria una operación y los datos necesarios para llevarla adelante **al mismo tiempo** (o al menos que todo se comporte como si así fuese).

- El postulado de Von Newmann “una única Unidad de Memoria para datos y código”, definitivamente no encaja en el nuevo paradigma.

- En este nuevo escenario, una memoria que almacene ambas entidades se transforma lisa y llanamente en un cuello de botella.

- Surge lo que se conoce como “Von Newmann bottleneck”.

Modelo o Arquitectura Harvard

- Recién por entonces y ante ese problema concreto, algunos diseñadores repararon en el diseño de la **Mark I**, un computador electromecánico desarrollado por IBM y la Universidad de Harvard entre 1939 y 1944. Es decir ni siquiera un computador de la primer generación, y obviamente anterior a la **EDVAC**.

- Mas aun: 760.000 engranajes, 800km de cables, basada en la maquina analítica de Babagge, y para colmo ¡¡¡trabajaba en Decimal!!!

- ¿Que podía tener de interesante esta pieza de museo que insumía 0.3 a 10 segundos por cálculo? (no obstante lo cual se utilizó hasta 1959...)

Modelo o Arquitectura Harvard

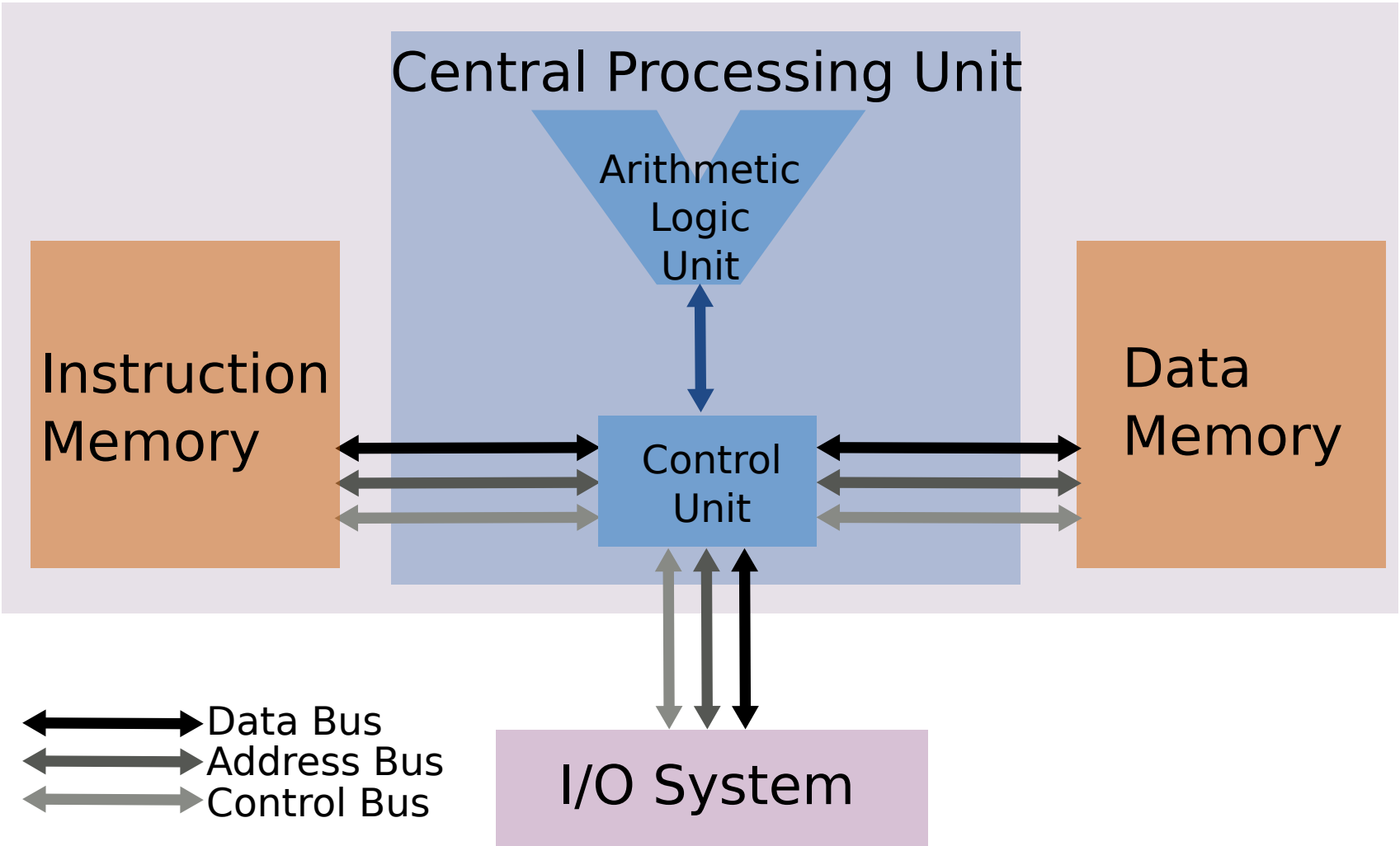
- Esta reliquia ni electrónica ni binaria tenía en sus entrañas la solución al “Von Newmann bottleneck”. Así son las cosas. . .
- Su código ingresaba mediante una cinta de papel perforada, y los operandos mediante contadores electromecánicos. Es decir, las instrucciones y los datos ingresan por data paths diferentes.
- Moraleja: ¡Viejos son los trapos!.

Modelo o Arquitectura Harvard

Conclusiones

- Un computador tiene organización o Arquitectura Harvard **cuando las instrucciones y los datos se leen desde memorias físicamente diferentes, y por caminos de señal físicamente diferentes**.
- Esto permite tener memorias de tecnología y organización diferente para instrucciones y para datos, y mapas de direcciones particulares en cada caso, y **caminos de señal (buses) independientes**.
- O sea podemos tener dos direcciones 0x00000000 de memoria: una para instrucciones y otra para datos. La memoria de instrucciones podría estar organizada en palabras de 32 bits y la de datos en palabras de 8 bits. Y todo funciona.

Modelo o Arquitectura Harvard



Arquitectura Harvard. Mapas y buses de memoria diferentes para Instrucciones y Datos

Modelo o Arquitectura Harvard

- La inmensa mayoría de los microprocesadores actuales parecieran presentar al usuario una organización acorde al modelo de Von Newman.
- Sin embargo como veremos en pocas clase mas, en su interior por razones de performance utilizan desde la década del 90 una tecnología denominada split cache: dividir los primeros niveles de memoria cache en una de Instrucciones y otra de datos, con sendos buses independientes para accederlas en paralelo. Esto es Harvard.
- Sin embargo, los niveles de memoria mas externos almacenan instrucciones y datos juntos. Esto es Von Newmann.

Modelo o Arquitectura Harvard

- El sub-sistema de memoria cache está diseñado y dimensionado de modo que la mayoría de los accesos a memoria de parte de las CPUs se producen en el cache, es decir, en la memoria organizada de acuerdo al modelo de Harvard.

- La única salvedad por la que no termina siendo “estrictamente Harvard” ni siquiera el primer nivel de cache está dado por el hecho que el programador “ve” una sola dirección de memoria en lugar de ver una de datos y otra de código. Esta división la realiza la CPU (en las MMU por ejemplo según veremos).

Modelo o Arquitectura Harvard

- Los primeros modelos de microprocesadores basados en arquitectura Harvard se dieron en la década del 80 con el advenimiento de los DSP (Procesadores de Señales Digitales), que requerían un alto nivel de paralelismo.
- Estos a la fecha son los únicos procesadores puramente Harvard.
- Los procesadores de propósito general como Intel x86, ARM CORTEX A, CORTEX-R, y últimamente algún que otro CORTEX-M, Power, entre otros trabajan con split cache en el Nivel 1 de su jerarquía de memoria, y juntan código y datos en los niveles mas externos de memoria.
- Todos presentan al programador un mapa de direcciones de memoria único. Razón por la cual se los llama “casi - Von Newmann”.

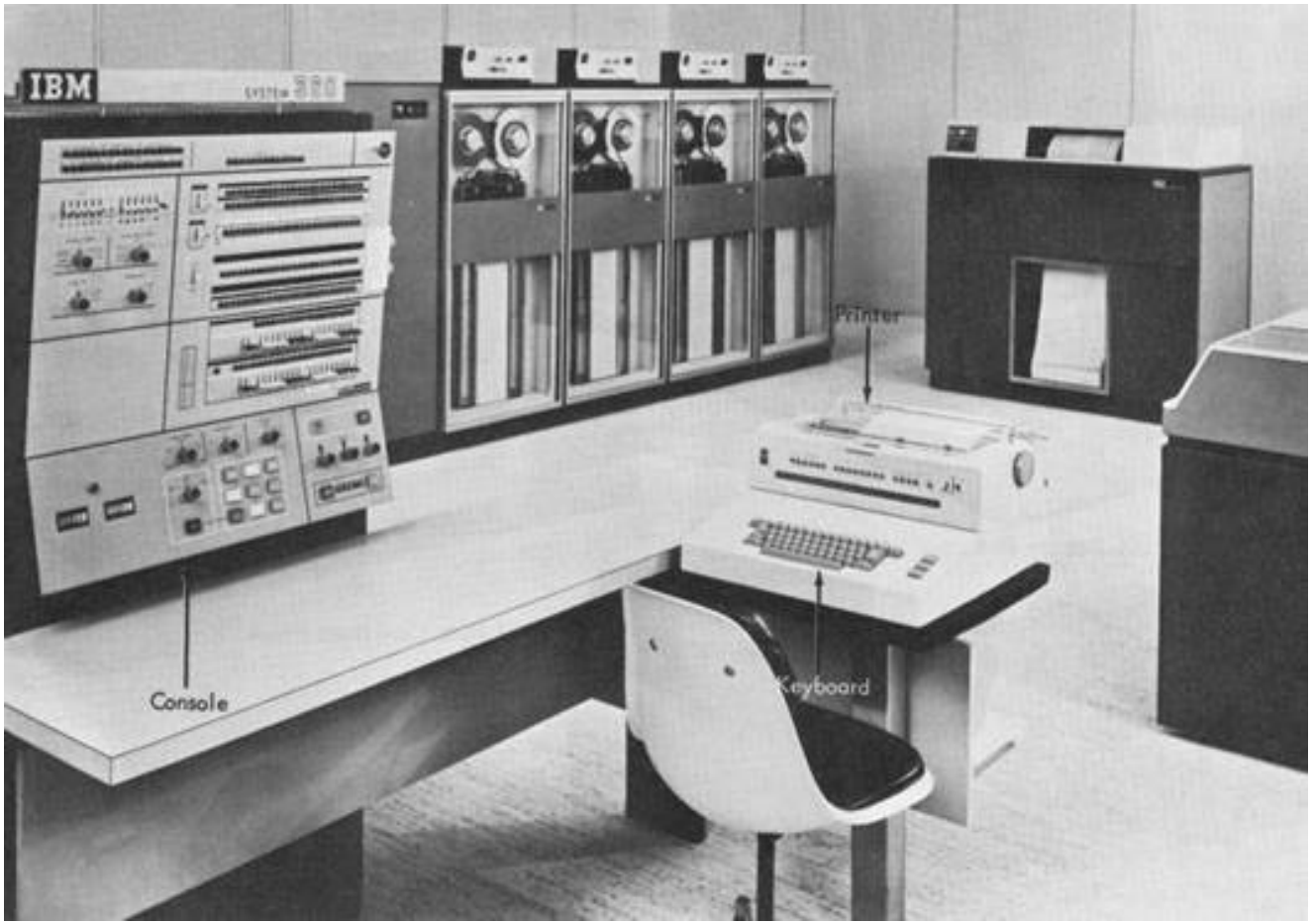
2da. Generación

- Son los computadores transistorizados.
- El transistor se implementa en 1947 en Laboratorios Bell. (patente en Canadá en 1925)
- Los primeros equipos aparecen durante fines de la década del 50.
- Ventajas: Menor consumo, menor espacio, y (a largo plazo) mejor desempeño.
- Coincide con los lenguajes de Alto Nivel. En 1957 se escribe el primer compilador de FORTRAN (**FOR**mula **TRAN**slator).
- Bell Labs presenta en 1958 la TRADIC (**TRA**nsistor, **DI**gital **C**omputer
- Se consolida IBM, aparece XEROX y DEC.

3er. Generación

- Computadores basados en circuitos integrados. Jack Kirby (TI) lo patenta junto con una implementación en 1959. (Jacobi, Siemens AG había patentado también en 1958 solo el dispositivo)
- En 1957 Robert Noyce funda Fairchild, y luego del patentamiento de Kirby le agrega una capa metálica que permite mejorar su conexionado.
- Igual que para la segunda generación, transcurriría un tiempo desde el descubrimiento del CI y la implementación de un computador de esta nueva generación.
- Los primeros equipos ven la luz en los años 60. El paisaje de un salón de cómputos comienza a verse muy diferente.

IBM 360. Minicomputador de 1964.



Cabe en una mesa. Los demás módulos son periféricos.

PDP-8. Minicomputador de 1964

Por su parte DEC lanza en el mismo año los PDP-8 que también podían ubicarse sobre una mesa pequeña.

Se consolida la Tecnología CMOS

- En los años 60 Los transistores de Efecto de Campo ya son una realidad y los MOSFET comienzan lentamente a impulsar la industria de los semiconductores.
- La miniaturización de los componentes de un circuito integrado avanza rápidamente, de modo que el tamaño de computadores de esta generación es significativamente menor.
- En 1968 Robert Noyce deja Fairchild, y funda Intel junto a Gordon Moore y Andrew Grove. Fue su primer CEO.

Se consolida la Tecnología CMOS

- En que se transformó Intel y su contribución a la industria y la ciencia de computación, junto con Texas, Fairchild, y otras empresas, es historia conocida.
- Esto genera un desarrollo impresionante en el Valle de Santa Clara en California del Norte, zona que pasa a conocerse como Silicon Valley.
- La figura de Robert Noyce fue tan crucial en este proceso que se lo suele referir como el Alcalde del Silicon Valley.
- Ingresamos a una vorágine tecnológica que transformará la forma de vida de los próximos 40 años de manera sustancial.

3er. Generación = Minicomputadores

Minicomputadores

Costaban un orden de cantidad menos que las enormes computadores transistorizados. Sus fabricantes podían fabricar en serie algunos miles de equipos y así se puede empezar a desarrollar la industria.

3er. Generación = Programación estructurada

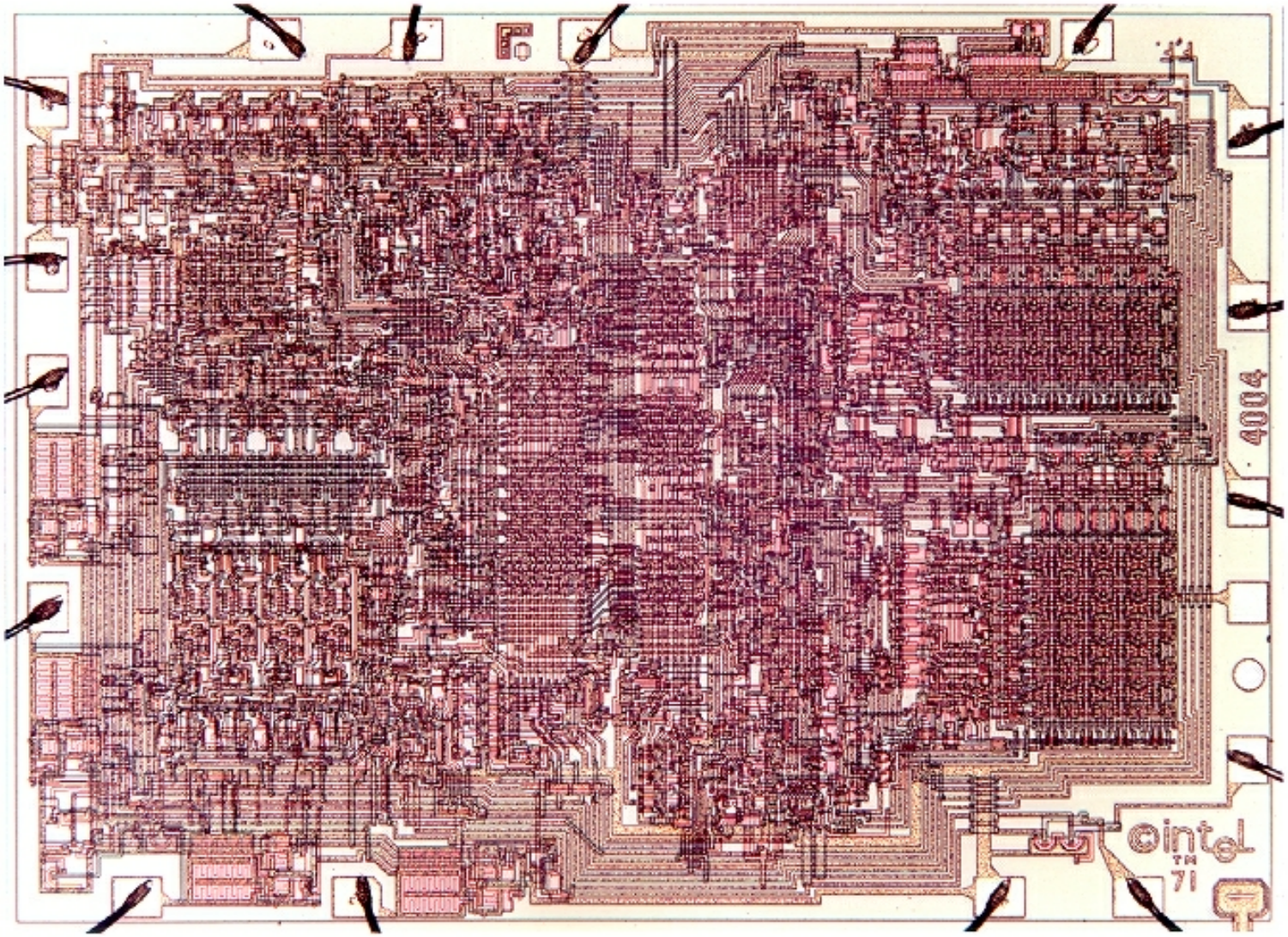
Programación Estructurada y Lenguaje C

- Se impone el paradigma de programación estructurada durante los años 60.
- En 1969 Dennis Ritchie y Ken Thompson presentan en Bell Labs un sistema operativo multitarea y multiusuario llamado UNIX, en una PDP-7.
- A principios de los años 70 Ritchie libera el primer compilador de lenguaje C, documenta la sintaxis del lenguaje,
- En 1970 Ritchie y Thompson Thompson escriben la primer versión de **UNIX** completamente portable cuyo código está escrito prácticamente en su totalidad en C.
- El C se impone para escribir sistemas operativos, introduce el concepto de portabilidad y también para desarrollar aplicaciones escalables y portables.

Aparecen los primeros microprocesadores

- Intel fabricaba CI de memorias.
- En 1971 presenta el CI 4004. Primer Microprocesador de la historia.
- Desarrollado a pedido para Busicom una compañía Japonesa fabricante de Calculadoras.
- Arquitectura de 4 bits Turing Completa (16 registros de 4 bits).
- Noyce contrata a Federico Faggin de Fairchild autor de la Silicon Gate Technology (SGT) tecnología crucial para obtener la integración de los 2300 transistores que componían el 4004.
- Tecnología de 5 μm (o 15 micras según la jerga de esa época), clock de 740KHz., y 12 bits para direcciones (4KB de memoria).
- Espacios de direcciones separados para instrucciones y datos
- Se le atribuye en forma errónea en ciertos documentos arquitectura Harvard
- 16 terminales. Multiplexa la dirección de 12 bits con los mismos 4 bits de datos.

El 4004 de Intel



Primer microprocesador de la historia.

Se empieza a acelerar la industria

- Con la tecnología SGT Intel lanza los procesadores 8008, 4040 y 8080, consolidándose como fabricante de Microprocesadores y memorias.
- En 1974 Faggin funda ZILOG y lanza el Z80, que supera claramente al 8080 y al 8085 con el que Intel intentará contrarrestarlo.
- Si IBM no hubiese seleccionado un proyecto de procesador de 16 bits en curso de Intel (el 8086) como procesador de su primer computadora personal otra sería la historia tal vez.
- Primeros computadores personales. Altair, IBM PC, Apple, Commodore, Sinclair, MSX, etc. utilizaron generaciones de procesadores de 8 o 16 bits.
- Hasta mitad de los 80 estamos en esta etapa. Aparecen Microsoft y Apple como actores nuevos de esta historia. El DOS se impone como Sistema Operativo y dominaría la década del 80. Aun entrada la cuarta generación.

Y UNIX ya había llegado en 1969

Esta máquina es a la que se portó la primer versión escrita en C.
16 Kbytes de memoria. 512 Kbytes de almacenamiento...

4ta. Generación

- El desarrollo de la tecnología de integración evolucionó de acuerdo con la predicción de Gordon Moore.
- Al momento de escribir este texto, la Ley de Moore está llegando a su fín, sencillamente porque se han alcanzado dimensiones físicas de niveles atómicos, de modo que ya no es posible reducir mucho mas las dimensiones de una compuerta de un transistor MOSFET.
- Alrededor de 1980 se comienza a hablar de Circuitos Integrados VLSI (Very Large Scale Integration), en referencia a circuitos integrados que contienen cientos de miles de transistores integrados.
- La verdad es que de lo que se ha producido en esta era es de lo que vamos a hablar en el resto de ésta asignatura y la siguiente.
- Lo divertido es que estamos transitándola aún

¿Porqué esta clase de historia?

- Actualmente la preocupación central de los fabricantes de microprocesadores es el consumo de energía eléctrica. todos investigan y trabajan diariamente para presentar chips que consuman el mínimo de energía para el mismo trabajo.

- Durante el corto lapso que lleva esta industria, la memoria fue durante muchas décadas el recurso mas escaso (y en rigor sigue vigente la máxima de Von Neumann *"la cantidad de memoria siempre será insuficiente"*). Pero como pudimos apreciar, en los albores esta escasez fue dramática.

- 16 Kbytes de RAM (8 para el sistema operativo y 8 para las aplicaciones) en 1969,. . . huelgan las palabras.

Uso mínimo de memoria: un “must”

Por lo tanto así como hoy se diseñan procesadores con foco puesto en el mínimo consumo de energía, hace 4 o 5 décadas, se diseñaban pensando en que los programas consuman la mínima cantidad de memoria posible.

Instrucciones complejas

1. Los diseñadores de la época apuntaban a microprocesadores con instrucciones muy complejas
2. En ese contexto se han diseñado Microprocesadores capaces de integrar código de alto nivel.
3. Ejemplos extremos: POLY (VAX computer)

Listado1

```
1       ; To compute P(x) = C0 + C1*x + C2*x**2
2       ; where C0 = 1.0,  C1 = .5, and C2 = .25
3
4               POLYF    X,#2,PTABLE
5               .
6               .
7               .
8   PTABLE:  .FLOAT   0.25      ; C2
9            .FLOAT   0.5       ; C1
10           .FLOAT   1.0       ; C0
```

Instrucciones complejas: Conclusión

Con los bloques de datos no hay mucho que hacer para ahorrar memoria. Pero en el caso del código, si podemos utilizar menos instrucciones para el mismo trabajo, se ahorra memoria. Este paradigma estuvo presente en todos los diseños durante décadas. Intel desarrolló su arquitectura actual con este paradigma. Y se comprometió a mantener **compatibilidad**.

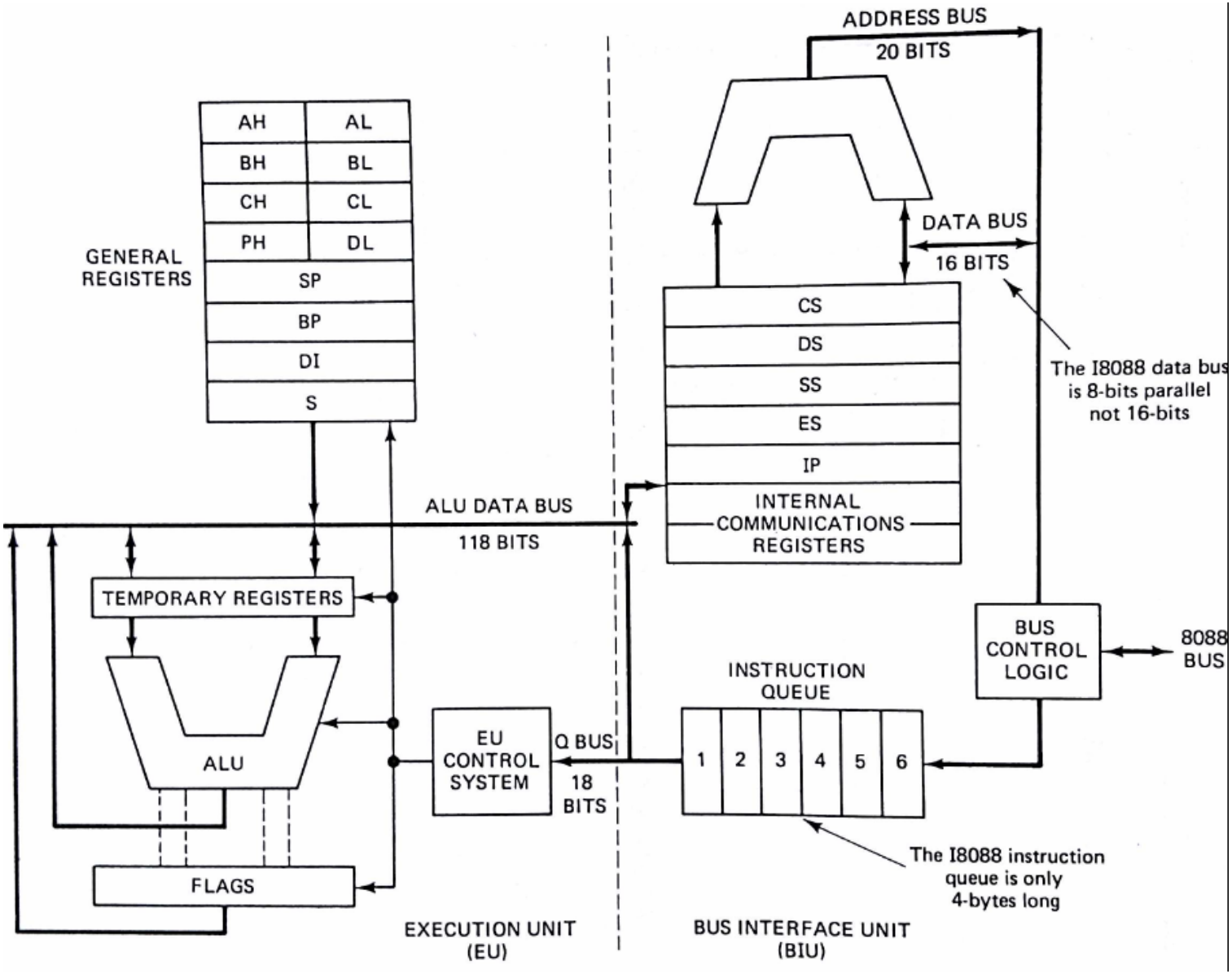
Otros problemas a resolver

- En los ’70, los usuarios de computadores demandaban disminuir el costo de actualización de sus equipos
- Cada upgrade obligaba a reemplazar no solo el hardware sino además el sistema operativo y las aplicaciones.
- Ésta dinámica dificultaba a los proveedores establecer un cash flow que permitiese incrementar inversiones en I+D.

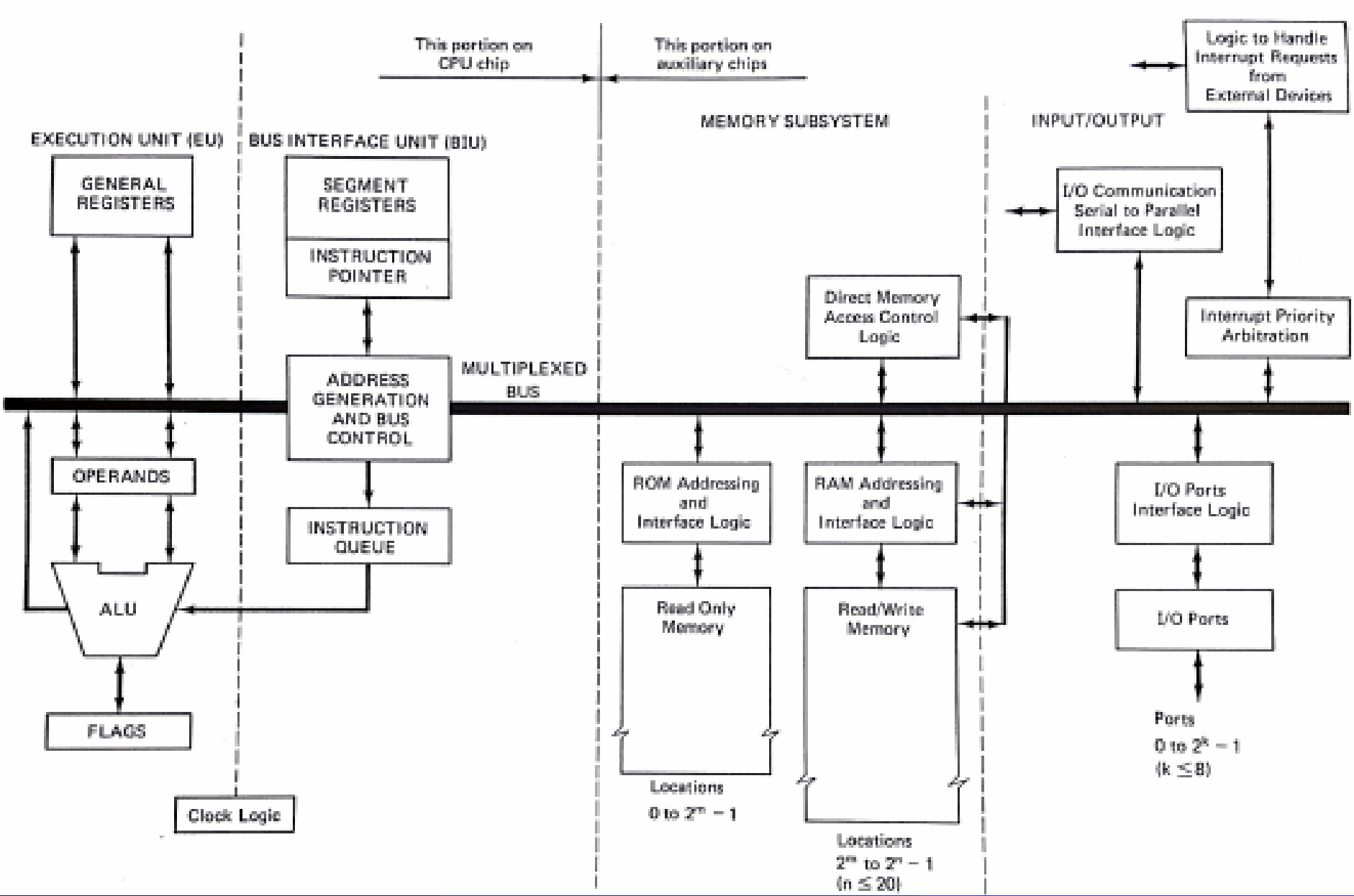
Intel dá el golpe

- En 1978 presenta la familia iAPx86 de la mano de su nave insignia procesador 8086,
- Se compromete a mantener compatibilidad ascendente en los modelos subsiguientes de procesadores para satisfacer esta demanda.
- Esto fue uno de los motivos de la decisión de IBM de adoptar esta familia de procesadores como base para su PC.

8086 Organización



Sistema básico de 8086



Arquitectura vs Microarquitectura

Arquitectura

Es el conjunto de recursos accesibles para el programador, que por lo general se mantienen a lo largo de los diferentes modelos de procesadores de esa arquitectura (puede evolucionar pero la tendencia es mantener compatibilidad hacia los modelos anteriores).

- Registros
- Set de instrucciones
- Estructuras de memoria (descriptores de segmento y de página p. ej.)

Micro Arquitectura

Es la implementación en el silicio de la arquitectura. Lo que está detrás del set de registros y del modelo de programación. Puede ser muy simple o sumamente robusta y poderosa. Cambia de un modelo a otro dentro de una misma familia.

Definición de la Arquitectura de un Computador

- Es sin dudas la tarea mas ardua de un diseñador.
- Se trata de definir los aspectos mas relevantes en la arquitectura de un computador que maximicen su rendimiento, sin dejar de satisfacer otras limitaciones impuestas por los usuarios, como un costo económico que lo haga accesible, o un consumo de energía moderado.
- Comprende el diseño del set de instrucciones, el manejo de la memoria y sus modos de direccionamiento, los restantes bloques funcionales que componen el CPU, el diseño lógico, y el proceso de implementación
- La implementación comprende el diseño del circuito integrado, su encapsulado, montaje, alimentación y refrigeración.

Definición de la Arquitectura de un Computador

skills necesarios

No es posible realizar esta tarea con éxito sin tener manejar de manera solvente varias tecnologías diferentes:

- Diseño lógico.
- Tecnología de encapsulado
- Funcionamiento y diseño de compiladores y de Sistemas Operativos.

Definiendo un ISA

Nos referimos a *Instruction Set Architecture*, como el set de instrucciones visibles por el programador. Es además el límite entre el software y el hardware.

Clases de ISA. ISAs con Registros de Propósito general vs. Registros dedicados. ISAs registro-memoria vs. ISAs Load Store.
Direccionamiento de Memoria. Alineación obligatoria de datos vs. administración de a bytes.
Modos de Direccionamiento. Como se especifican los operandos.
Tipos y tamaños de operandos. Enteros, Punto Flotante, Punto Fijo.Diferentes tamaños y precisiones.
Operaciones. Pocas Simples (RISC) o Muchas Complejas (CISC).
Instrucciones de control de flujo Saltos condicionales, calls.
Longitud del código Instrucciones de tamaño fijo vs. variable.

Microarquitectura = Organización + Hardware

Organización

Se refiere a los detalles de implementación de la ISA. Es decir:

- Organización e interconexión de la memoria.
- Diseño de los diferentes bloques de la CPU y para implementar el set de instrucciones.
- Implementación de paralelismo a nivel de Instrucciones, y/o de datos.
- Podemos así encontrar procesadores que poseen la misma ISA pero una organización muy diferente.

Ejemplo:

Los procesadores AMD FX y los Intel Core i7, tienen la misma ISA, sin embargo organizan su caché y su motor de ejecución de maneras diferentes.

Microarquitectura = Organización + Hardware

Hardware

- Se refiere a los detalles de diseño lógico y tecnología de fabricación.
- Existirán por lo tanto, procesadores con la misma ISA y la misma organización, pero absolutamente distintos a nivel de hardware y diseño lógico detallado.

Ejemplo:

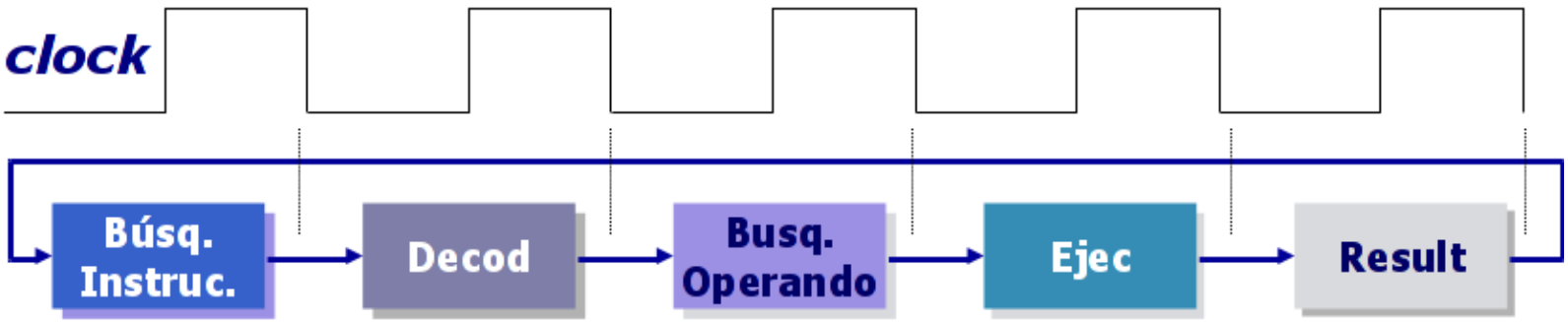
El Pentium 4, diseñado para desktop, y el Pentium 4 M para notebooks. Este último tiene una cantidad de lógica para control del consumo de energía, siendo su hardware muy diferente del del Pentium 4 desktop.

¿Porque necesitamos saber todo esto?

- Para comprender lo que ocurre debajo del software.
- Comprender como los diseños de hardware impactan en el software y en el programador
- Estar en condiciones de cruzar verticalmente las capas que separan el silicio de la aplicación.
- Comprender las nociones básicas que rigen el diseño de un sistema de cómputo moderno

Máquina de estados elemental

- En la década del 40 Von Newman definió un modelo básico de CPU, que a esta altura está mas que superado.
- Sin embargo algunos conceptos de ese modelo viven en los mas modernos procesadores.
- Uno de ellos es la máquina de ejecución

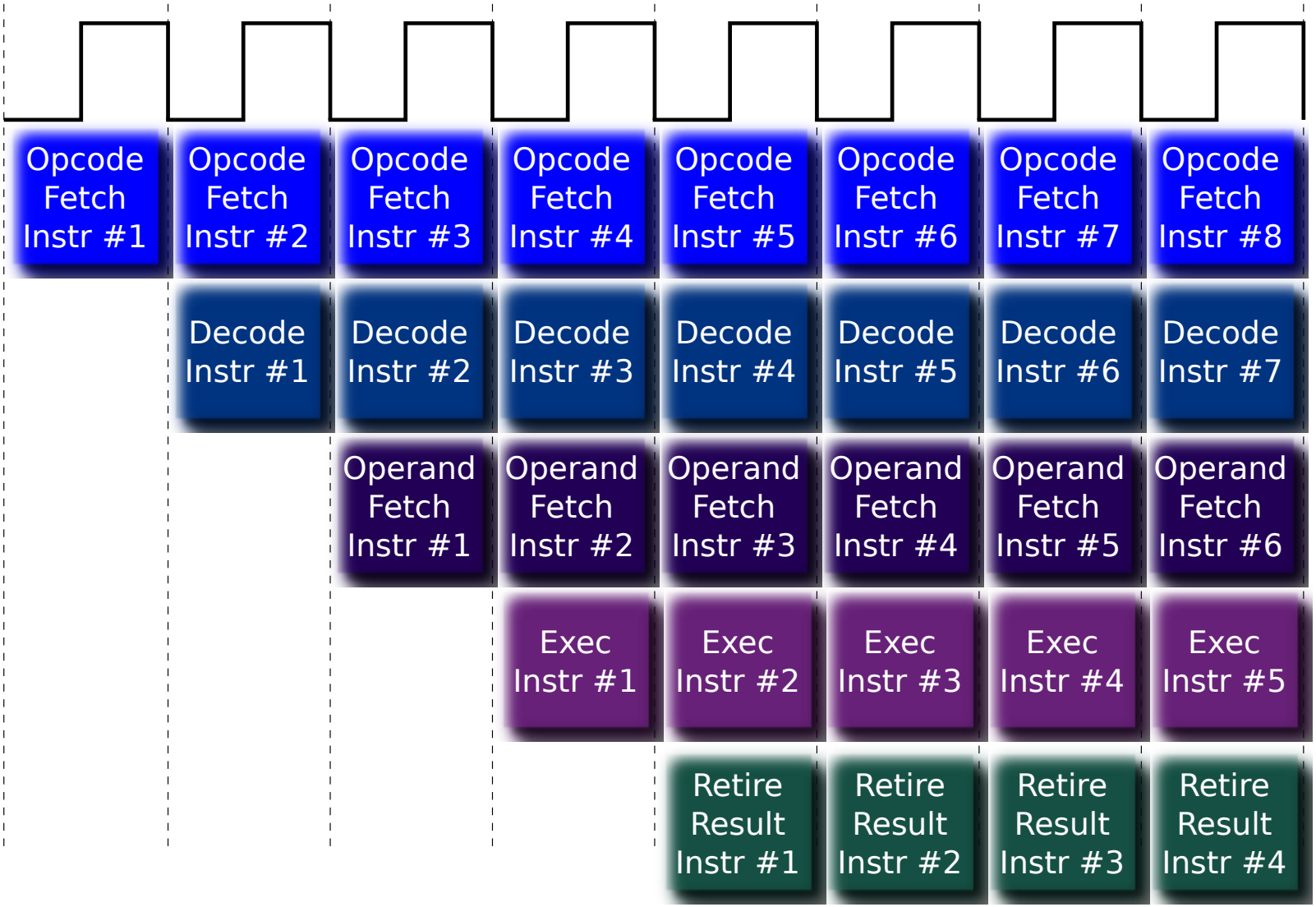


- En las primeras generaciones de microprocesadores cada etapa de este ciclo se ejecutaba en un ciclo de clock, y la CPU entera estaba dedicada a esa tarea.
- Ejecutar una instrucción insumía de varios ciclos de clock. . .

Pipeline

- Arquitectura que permite crear el efecto de superponer en el tiempo, la ejecución de varias instrucciones a la vez.
- Con él se formaliza el concepto de ***Instruction Level Paralelism (ILP)***.
- Requiere muy poco o ningún hardware adicional.
- Solo necesita que los bloques del procesador que resuelven la máqiuina de estados para la ejecución de una instrucción, operen en forma simultánea.
- Se logra si todos los bloques funcionales trabajan en paralelo pero cada uno en una instrucción diferente.
- Es algo parecido al concepto de una línea de montaje, en donde cada operación se descompone en partes, y se ejecutan en un mismo momento diferentes partes de diferentes operaciones.
- Cada parte se denomina etapa (stage)

Pipeline



Este modelo es teórico y representa la situación ideal.

Pipeline

Conclusiones

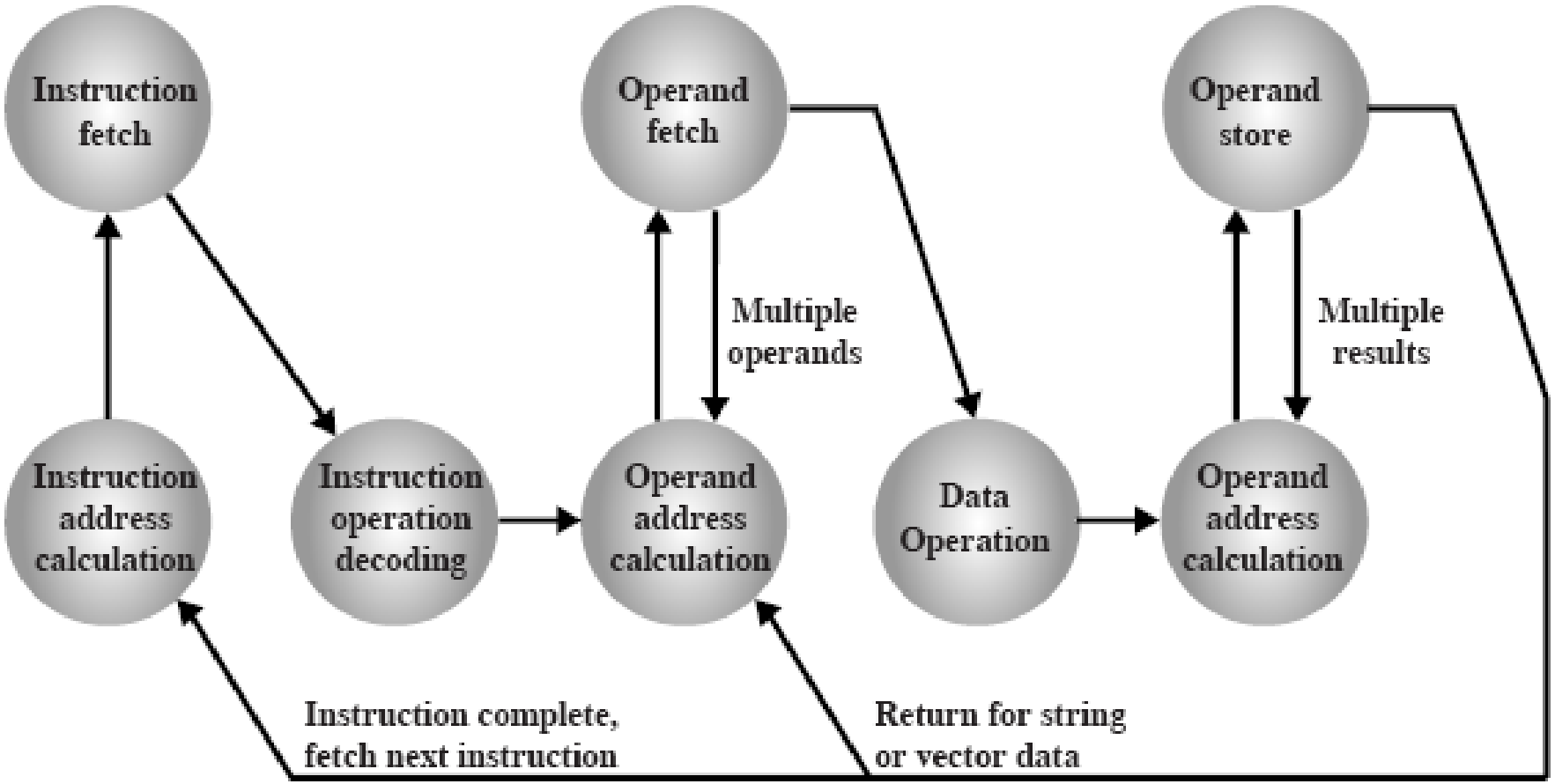
1. El pipeline tarda en llegar a esa condición de régimen tantos ciclos de clock como etapas tenga.

2. En un pipeline de 5 etapas, y en el caso ideal en que cada etapa consuma solamente un ciclo de clock, una arquitectura pipeline provee un resultado de instrucción por cada ciclo de clock, a partir del ciclo de clock en el cual se llega a resolver la primer instrucción.

3. Este escenario permanente es puramente teórico. En la práctica no se cumple todo el tiempo.

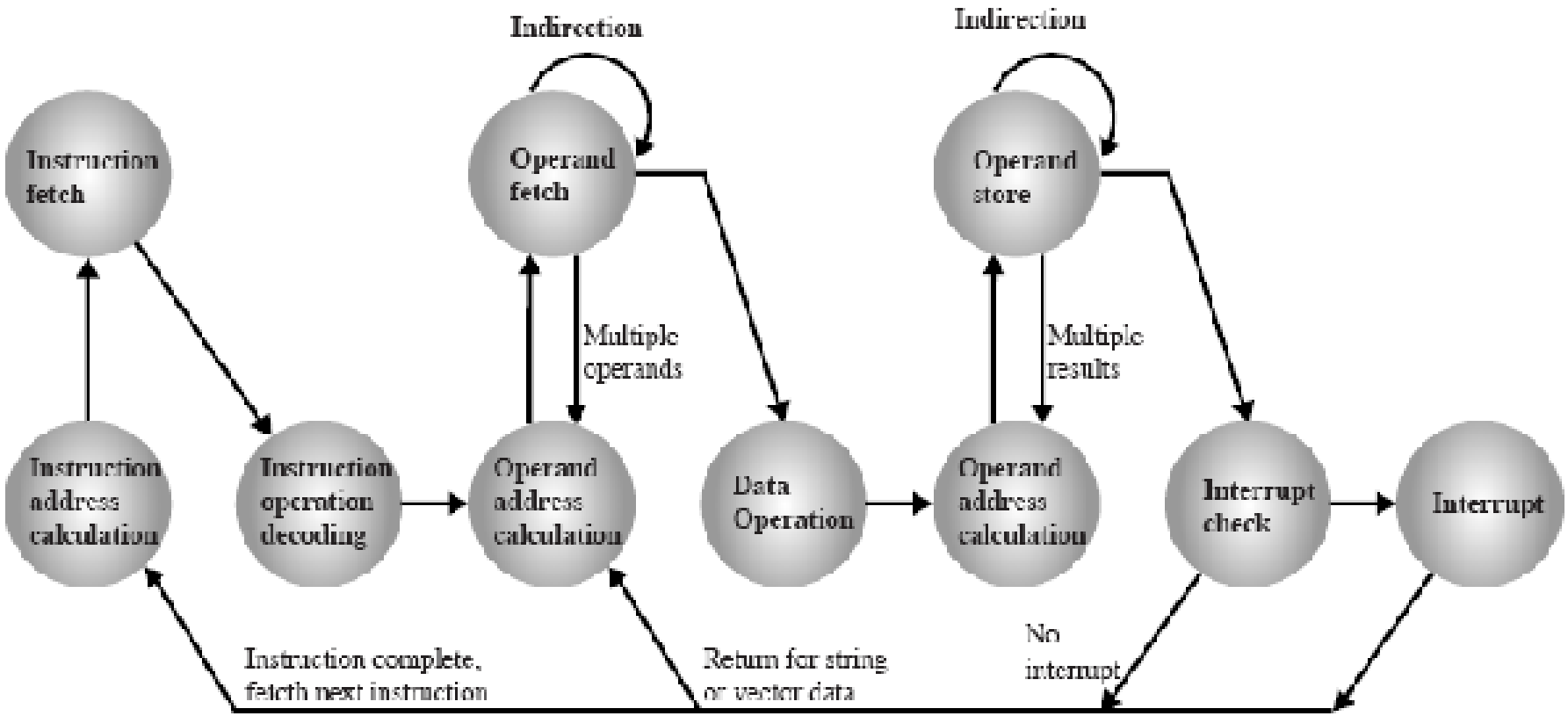
Deep Pipeline

Si las conclusiones son ciertas agregar etapas mejora la performance



Deeper

Etapas mas simples aseguran que trabajan en un solo clock. Por eso podríamos pensar en aumentarlas mas y mas



Eficiencia de un pipeline

- ¿Porque querríamos aumentar y aumentar el número de etapas?
- Para un tiempo ya establecido de procesamiento interno de una instrucción para una arquitectura “no pipelinezada”, intuitivamente puede comprenderse que en principio cuantas mas etapas podamos definir para ejecutar esta operación, al ponerlas a trabajar a todas en paralelo en un pipeline, el tiempo de ejecución de la instrucción se reducirá proporcionalmente con la cantidad de etapas.

$$TPI = \frac{Tiempo\ por\ instrucción\ en\ la\ CPU\ ''\ No - Pipeline''}{Cantidad\ de\ etapas} \quad (1)$$

Donde ***TPI*** significa Time Per Instruction

Conclusiones

- Considerar la situación teórica e ideal dada por la ecuación (1) no es 100 % cierto.
- En la práctica existen overheads introducidos por el pipeline, que suman pequeñas demoras, pero de todos modos el tiempo se aproxima mucho al ideal.
- El resultado es que la reducción puede apreciarse como si se requiriesen finalmente menos CPI para completar una instrucción. ¿Porque?

Conclusiones

- El pipeline no reduce el tiempo de ejecución de cada instrucción individual, sino que al aplicarse en paralelo al flujo de instrucciones, incrementa el número de instrucciones completadas por unidad de tiempo.
- De hecho el overhead del pipeline perjudica el tiempo de ejecución individual, de manera poco significativa, pero agrega tiempo a cada instrucción.
- El rendimiento (throughput) del procesador mejora notablemente ya que los programas ejecutan notablemente mas rápido.

Hazards (obstáculos)

- Hay obstáculos que conspiran contra la eficiencia de un pipeline.
- Podemos agruparlos en tres categorías:

1. ***Obstáculos estructurales***.
2. ***Obstáculos de datos***.
3. ***Obstáculos de control***.

- En cualquier caso al efecto ocasionado por un obstácuo se lo denomina ***pipeline stall***.
- Su efecto degrada la performance del procesador

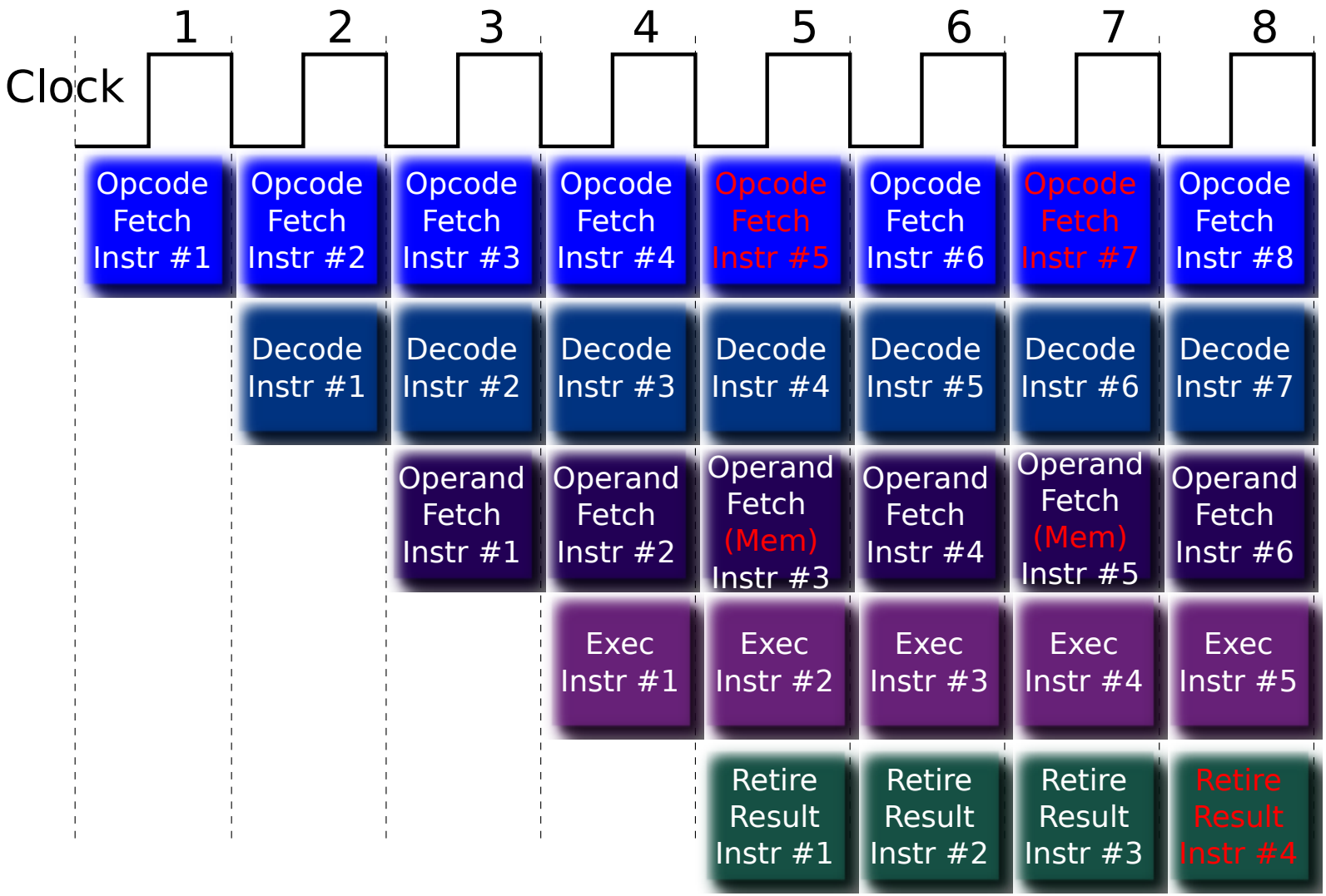
Obstáculos Estructurales

- Se pueden dar por varias razones:
  - Una etapa no está suficientemente atomizada, y concentra aún demasiadas funciones lo que hace que para completarse requiere mas de un ciclo de clock.
  - Si dos instrucciones que utilizarán esta etapa están mas próximas del tiempo que necesita esta etapa para procesar su parte, caerán en conflicto de recursos para su ejecución.
  - Este grupo de instrucciones entonces, no puede pasar por esa etapa del pipeline en un ciclo de clock.
  - En estas circunstancias una de las instrucciones debe detenerse. Como consecuencia CPI se incrementa en 1 o mas respecto del valor ideal.

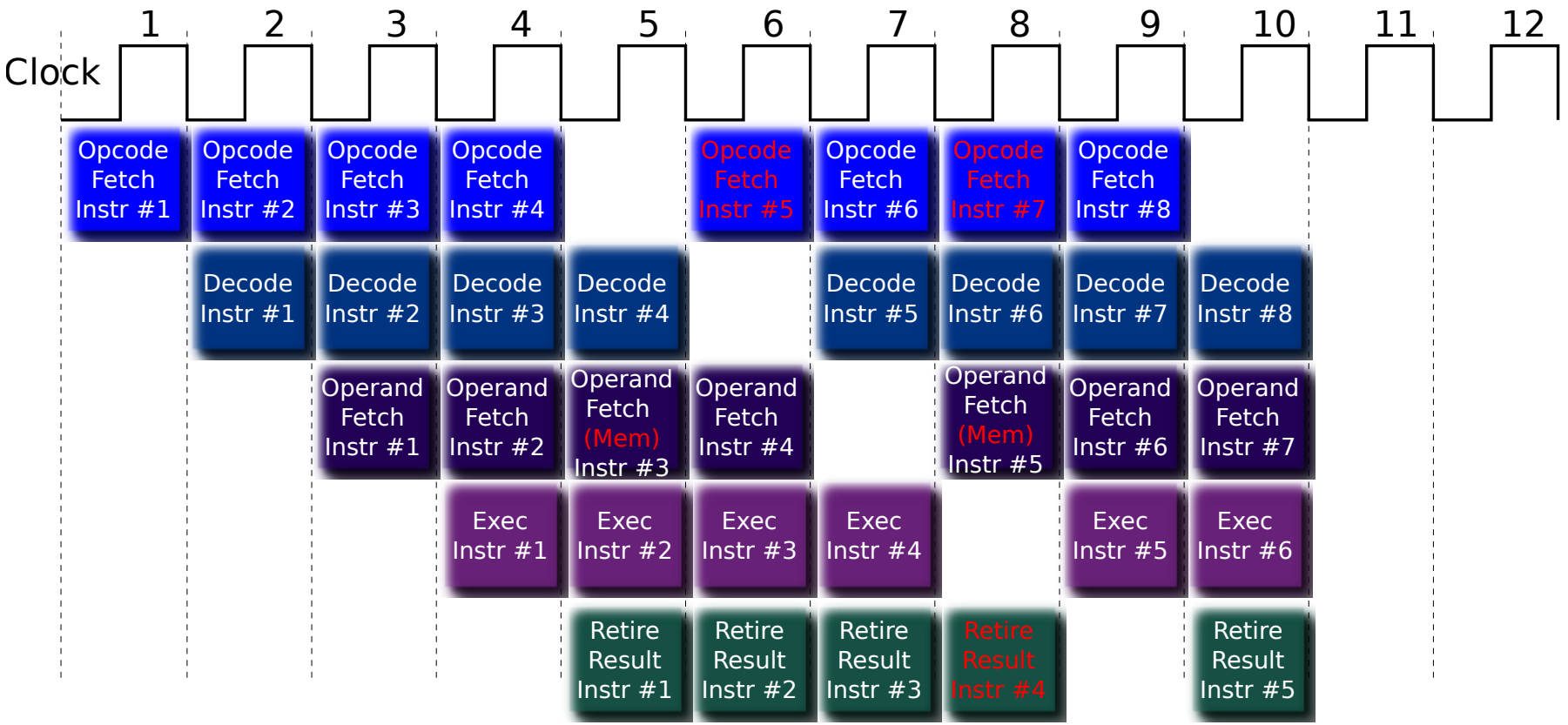
Obstáculos Estructurales: Ejemplo

- Tenemos un procesador que solo tiene una etapa para acceder a memoria y la comparte para acceso a datos e instrucciones.
- En el caso de que se necesite un operando de memoria, el acceso para traer este operando interferirá con la búsqueda del operando de una instrucción mas adelante del programa.
- También interferirá con el Fectch de la siguiente instrucción

Obstáculos Estructurales: Accesos concurrentes a memoria

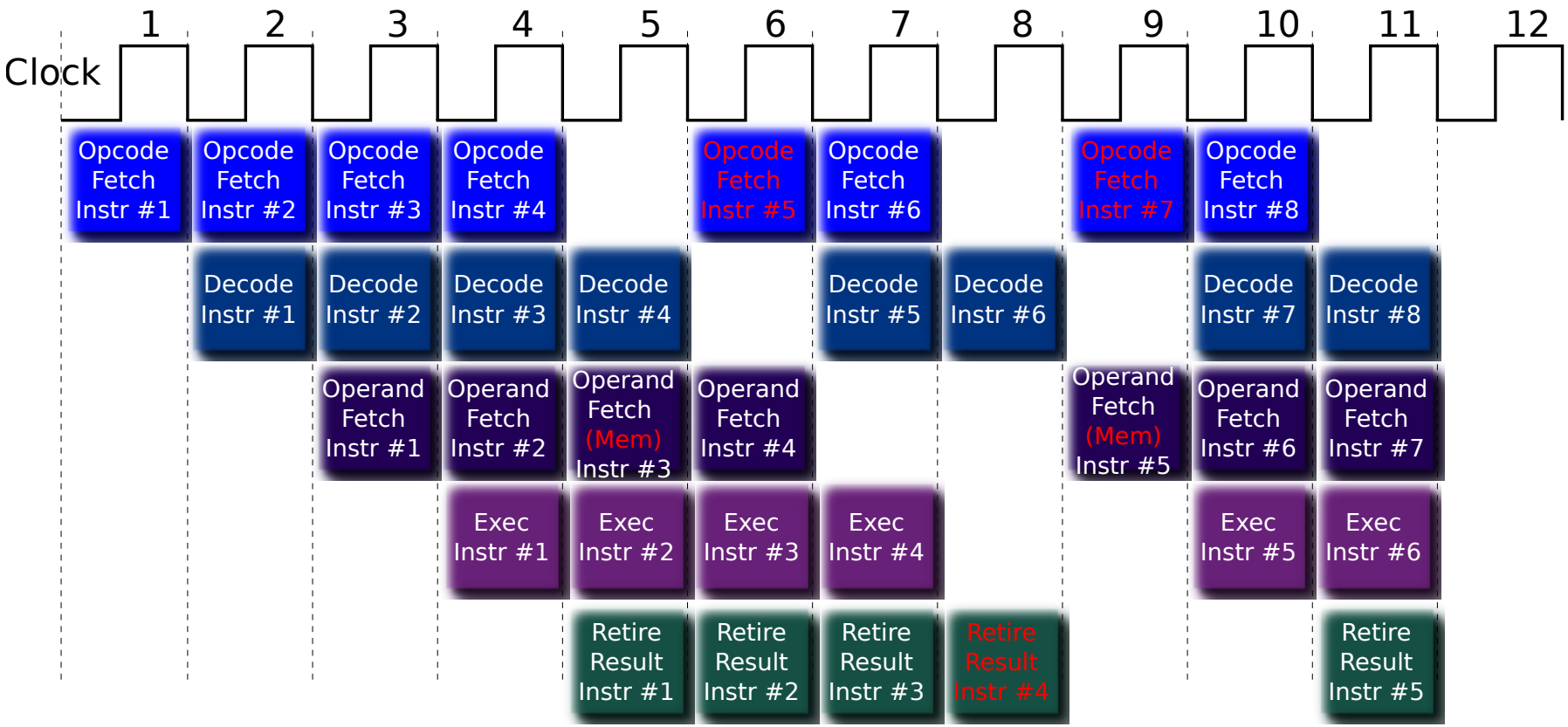


Obstáculos Estructurales - Efecto en CPI



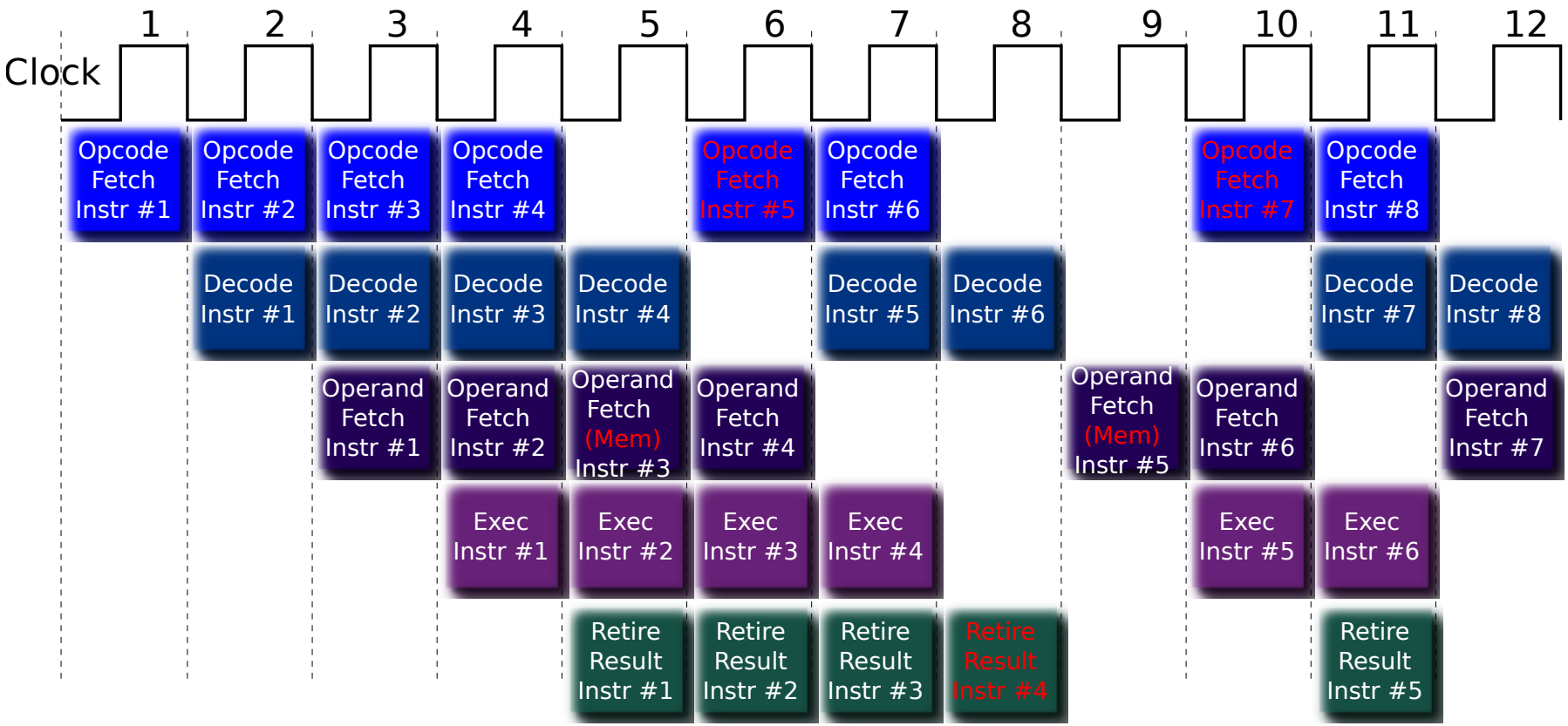
- Por cada Obstáculo, pospone una operación. $CPI = CPI + 1$
- En general la cantidad de CPI que se incrementan es igual a la cantidad de concurrencias menos 1, en el lapso considerado

Obstáculos Estructurales - Efecto en CPI



- Por cada Obstáculo, pospone una operación. $CPI = CPI + 1$
- En general la cantidad de CPI que se incrementan es igual a la cantidad de concurrencias menos 1, en el lapso considerado

Obstáculos Estructurales - Efecto en CPI



- Por cada Obstáculo, pospone una operación. $CPI = CPI + 1$
- En general la cantidad de CPI que se incrementan es igual a la cantidad de concurrencias menos 1, en el lapso considerado

Obstáculos Estructurales - Posibles soluciones

- Cualesquiera sean las soluciones que deseemos implementar, es necesario agregar hardware.
- En el caso de accesos a memoria, se puede resolver mediante las siguientes opciones por separado o combinadas:
  1. Desdoblamiento del cache L1 en Cache de datos y Cache de instrucciones.
  2. Empleo de buffers de instrucciones implementados como pequeñas colas FIFO.
  3. Ensanchamiento de los buses mas allá de los anchos de palabra del procesador.
- Aumentar la profundidad del pipeline, produce etapas mucho mas simples capaces de resolver en un clock su tarea.

Obstáculos de datos

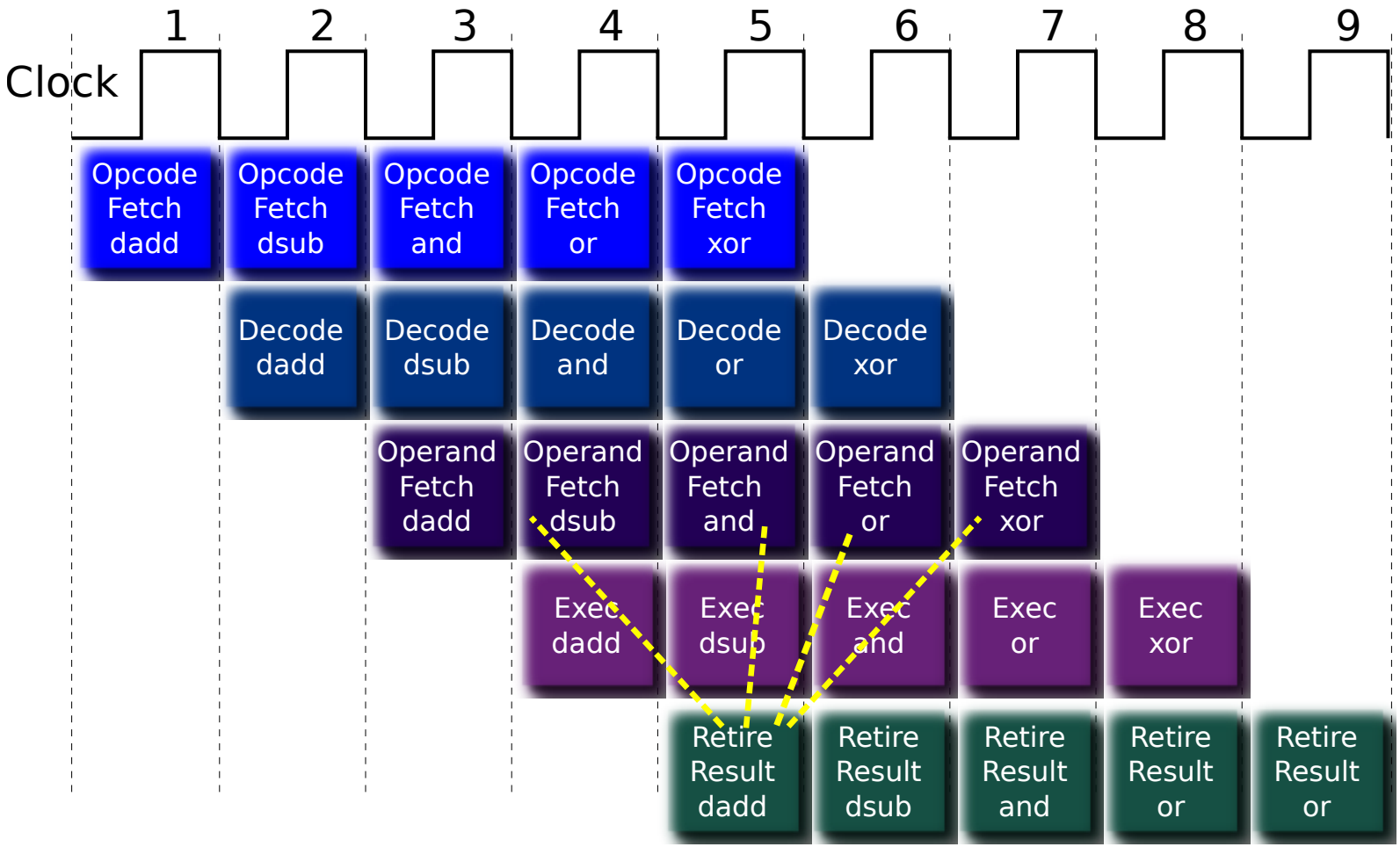
- Se producen cuando por efecto del pipeline, una instrucción requiere de un dato antes de que este esté disponible por efecto de la secuencia lógica prevista en el programa.
- Consideremos el siguiente código para un procesador MIPS.

Listado 2

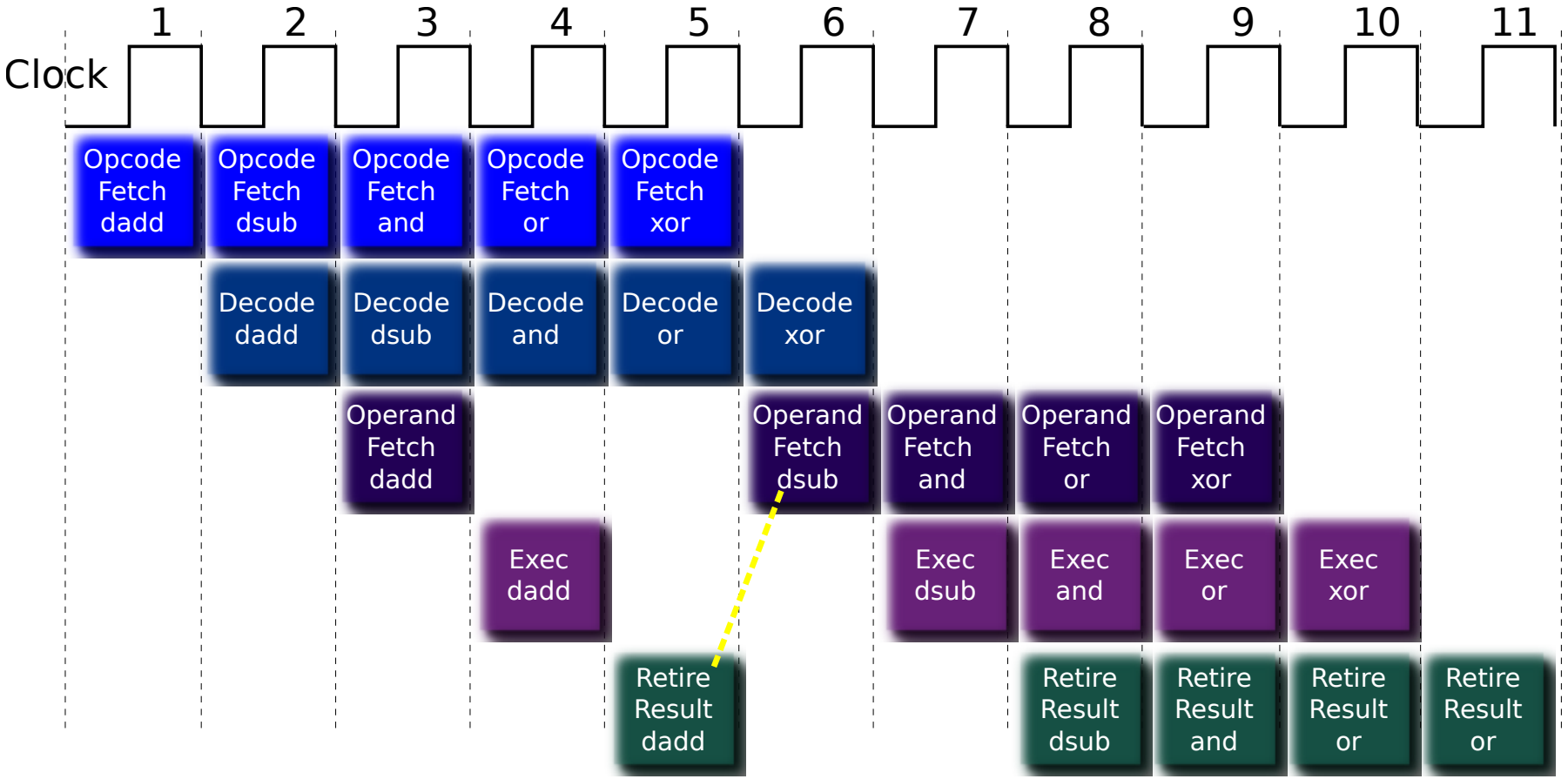
```
dadd   R1, R2, R3
dsub   R4, R1, R5
and    R6, R1, R7
or     R8, R1, R9
xor    R10, R1, R11
```

- Tenemos dependencias para el Registro R1. Hasta que la instrucción ***dadd*** no complete su operación, R1 no tiene un valor válido. Por lo tanto no puede continuar aplicándolo en las restantes que lo utilizan.
- La situación en el pipeline es la siguiente:

Obstáculos de Datos - Conflicto de recursos



Obstáculos de Datos - Efecto en CPI

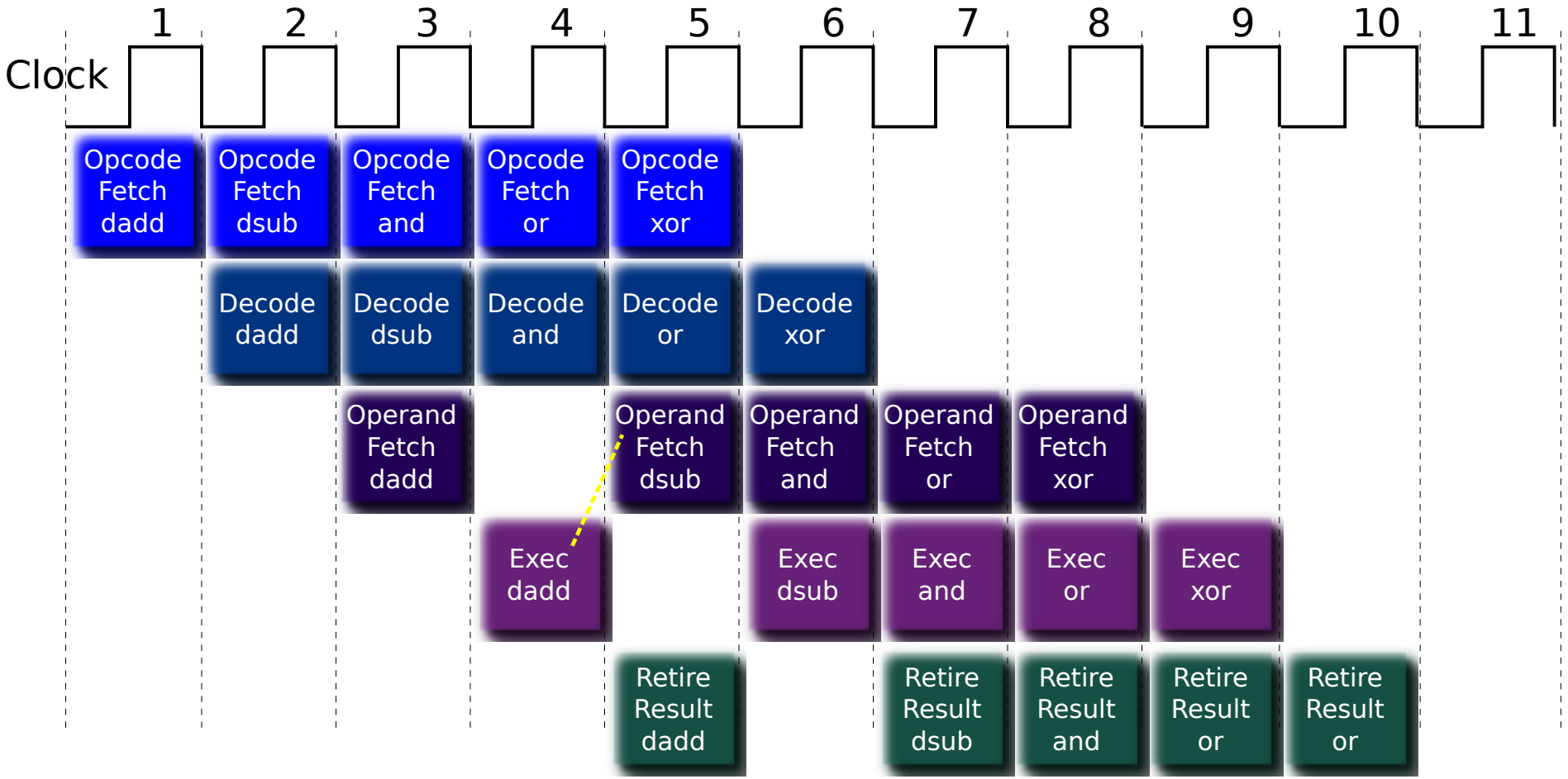


- Por cada Dependencia, pospone una operación. $CPI = CPI + n$, siendo n la distancia entre las etapas del pipeline que requieren el mismo dato.

Solución a Obstáculos de Datos - Forwarding

- Se extrae el resultado directamente de la salida de la unidad de Ejecución (ALU, Floating Point, o la que corresponda a la instrucción), cuando está disponible y se lo envía a la entrada de la etapa que lo requiere en el mismo ciclo de clock en que se escribe en el operando destino. Esto permite disponer del dato en la siguiente instrucción ahorrando en la espera el tiempo de escritura en el operando destino. sin que ésta deba esperar que se retire el resultado (es decir que se aplique en el operando destino).
- Se aplica solamente a las etapas posteriores que quedarían en estado stall.
- A aquellas etapas que no quedarían en estado stall como consecuencia de la dependencia de datos, se les envía la salida del resultado cuando este es aplicado en el operando destino.

Obstáculos de Datos - Forwarding



Algunas veces forwarding no es factible

- Normalmente forwarding aplica a operaciones de ALU denominadas back-to-back, ya que el bypass se efectúa desde la ALU a un registro entero del procesador.
- Consideremos ahora este otro código para un procesador MIPS.

Listado 3

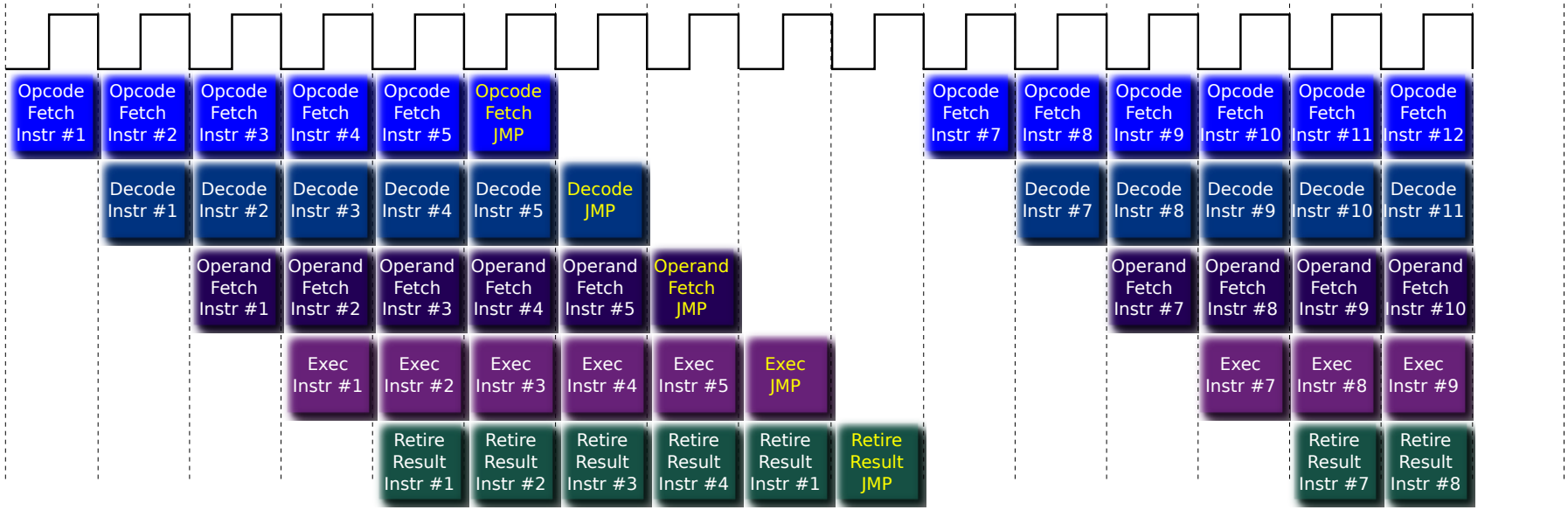
```
ld      R1, 0(R2)
dsub    R4, R1, R5
and     R6, R1, R7
or      R8, R1, R9
xor     R10, R1, R11
```

- En este caso el dato no puede adelantarse hasta que no se complete el ciclo de clock en el que se impacta el resultado dentro del procesador.

Obstáculos de Control

- Un branch es la peor situación en pérdida de performance.
- Un branch es una discontinuidad en el flujo de ejecución.
- El pipeline busca instrucciones en secuencia.
- El branch hace que todo lo que estaba pre procesado deba descartarse. Y el pipeline se vacía debiendo transcurrir ***n -1*** ciclos de clock hasta el próximo resultado. Siendo ***n*** la cantidad de etapas del pipeline. Esto se conoce como **branch penalty**.

”La conspiración de los branches”



saltos, branches, interrupciones, llamadas...

- En las interrupciones la situación es la del gráfico del slide anterior.
- El principal inconveniente se tiene cuando el salto es condicional, ya que es necesario determinar si la condición es true o false.
- Si la ondición es true se habla de ***branch taken*** (cambia el registro PC a la dirección de salto), y en el segundo de ***branch untaken*** (PC apunta a la instrucción siguiente al branch).

Saltos: ¿Cuando puede determinarse si es taken?

En el esquema de pipeline clásico que venimos analizando, esta condición se verifica:

1. en la fase de ejecución, si es un salto condicional.
2. en la fase de búsqueda de operando si es un salto incondicional o llamada a subrutina, con direccionamiento indirecto (es decir la dirección de salto está en la memoria en la dirección que contiene la instrucción).
3. o, en el mejor de los casos en la fase de decodificación si es un salto incondicional o llamada a subrutina con direccionamiento directo (es decir la dirección de salto viene a continuación del código de operación).

Efectos de los branches y como neutralizarlos

- Forwarding puede ayudar a disminuir el efecto de los diferentes casos expuestos anteriormente. Sin embargo, no es óptima, ya que solo lograremos disminuir algunos ciclos de clock del **branch penalty**, pero no se puede eliminar del todo su efecto.

- Para soluciones mas eficientes es necesario recurrir a análisis mas pormenorizados, que en general tienen en cuenta el comportamiento de los algoritmos y de los saltos.

- Ingresamos al universo de las unidades de predicción de saltos. Las hay desde muy simples a muy sofisticadas, y tiene que ver no solo con las diferentes generaciones de procesadores, sino también con el tipo de procesador bajo análisis.

Orígenes de RISC

- Antes de las Personal Computers, el mercado era dominado por los denominados mainframes.
- Los players eran IBM, Digital Electronic Corporation (DEC), Amdahl, entre otros. Sus computadores hoy son piezas de museo.
- Sobre los años 80 aparecen los primeros microprocesadores de 16 bits con ciertas capacidades para hacer frente a una nueva generación de equipos de cómputo:
  - Intel evoluciona el 8086/88 al 80286, y ensaya el iAPx432 citado como micromainframe processor, especialmente diseñado para ejecutar lenguajes de alto nivel como Ada.
  - Motorola evoluciona el 68000 (citado como m68k)
- Vimos que tienen todos un punto en común: Set de instrucciones de complejidad creciente

Historia de RISC

Líneas de Investigación

Durante los años ’60 y ’70, IBM, Control Data Corporation (CDC), y Data General (DG), incursionaron líneas de investigación tendientes a implementar procesadores con pocas instrucciones simples.
En los ’80 David Patterson[1] [2] (Universidad de Berkeley), y John Hennesy[3] (Universidad de Stanford) publicaron los primeros trabajos con resultados concretos, que presentaban una arquitectura contrapuesta con la de los Computadores que en ese momento dominaban la industria. la denominaron **RISC** (**R**educed **I**nstruction **S**et **C**omputer). Y esto motivó que a los procesadores diseñados hasta entonces se los etiquetar como **CISC** (Por **C**omplex **I**nstrucion **S**et **C**omputer)

[1] David A. Patterson, Carlo H. Sequin. RISC I: A Reduced Instruction Set VLSI Computer

[2] David A. Patterson, David R. Ditzel. The case for the reduced instruction set computer

[3] Hennessy, J.L., Jouppi, N., Baskett, F., Gill, J. MIPS: A VLSI Processor Architecture. In Proceedings CMU Conference on VLSI Systems and Computations. Computer Science Press, October 1981

Relación con el pipeline

- Se analizaron los obstáculos en un pipeline.
- En particular los obstáculos de datos impactan cuando involucran accesos a memoria para buscar operandos.
- Claramente si se evita que los operandos se accedan en memoria, en lugar de cargarlos previamente en un registro tal vez el pipeline pueda mejorarse

Hennesy & Patterson

Los Mandamientos RISC

1. Se dispone de un juego de registros numeroso, todos de propósito general.
2. Las instrucciones se ejecutan en un solo ciclo de clock.
3. Las instrucciones derivan en códigos de operación de igual formato y tamaño.
4. Las instrucciones deben ser sencillas de decodificar. Los números de registros se deben tener la misma ubicación en los códigos de la instrucción y deben requerir la misma cantidad de bits para su decodificación.
5. No se utiliza micro código para decodificar instrucciones (no hay instrucciones complejas, como DIV o MUL).
6. Los datos en memoria se acceden mediante instrucciones simples de transferencia: LOAD y STORE.

Corolario

1. La validación de una máquina RISC es mucho mas simple de realizar, y también es menor la probabilidad de liberar al mercado un procesador con defectos de lógica. (Recordemos el bug de división del Pentium@90Mhz)
2. Es esperable que el tamaño de los programas sea mayor, debido a que lo que los procesadores CISC resuelven con microcódigo, los RISC lo resuelven en el software. Una multiplicación se resuelve siempre por acumulación de sumas.

$$A * B = \sum_{i=1}^{A} B$$

- En microcódigo se implementa la instrucción MUL.
- Por software se implementa con una subrutina.

```
1            sub  R2,R2,1
2   sumar:   add  R1,R1,R1
3            sub  R2,R2,1
4            blez R2,sumar
```

Algunas comparaciones de código

Se requieren dos cosas:

1. Ingresar a Compiler Explorer
2. Para sacar conclusiones útiles, pensar y aplicar lo que vimos hasta aqui.