IEE240 – Organización y Arquitectura de Computadoras

MSc. Stefano Romero

Capítulo 2

Estructura de una computadora y de un CPU

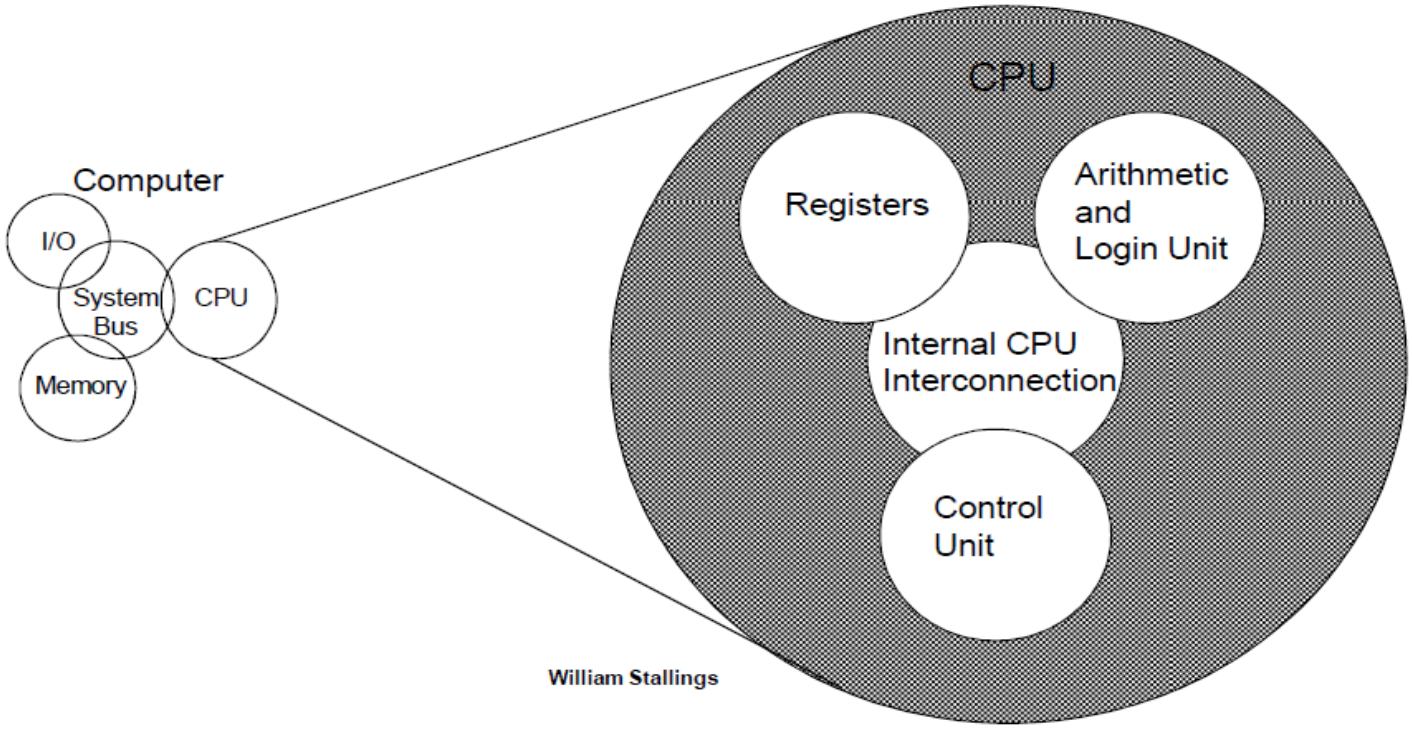


Figura 2.1 Estructura de un CPU. [1]

Capítulo 2

Registros, unidad aritmética lógica (ALU) y unidad de control

- El CPU se puede dividir de la siguiente manera:
  - **Datapath:**
    - ✓ **Registro:** Espacio asignado que proporciona almacenamiento interno a la CPU.
    - ✓ **ALU:** Encargada de las funciones de procesamiento de datos del computador.
  - **Unidad de control:** Módulo encargado del funcionamiento del CPU. Se asegura que la data esté donde se necesite en el tiempo correcto.
- Por lo que, un CPU se encarga de obtener instrucciones, decodificarlas y realizar la secuencia de operaciones indicada.

Capítulo 2

Registro

- Son espacios que almacenan diferente tipo de data como direcciones, contadores de programa y data para la ejecución de un programa.
- La data almacenada es binaria y está localizada en el mismo procesador por lo que acceder a ella es bastante rápido.
- Los registros suelen tener tamaños de 16, 32 y 64 bits. Así mismo, el número de registros típicamente son potencias de 2, siendo los más comunes 16, 32 y 64.
- Los registros contienen data, direcciones y/o control de información.
- Actualmente, las computadoras tienen distintos tipos de registro (almacenamiento, desplazamiento, de comparación, contadores, bandera, etc.)
- Ejemplo: La arquitectura de Pentium tiene registros de data y registro de direcciones.

Capítulo 2

Unidad aritmética-lógica

- Es la encargada de las operaciones lógicas y aritméticas (suma o multiplicación) que se requieren en la ejecución de un programa.
- Generalmente, la ALU tiene un dato de entrada y otro de salida.
- La ALU sabe qué operaciones puede realizar ya que es controlada por la señales de la unidad de control.

Capítulo 2

Unidad aritmética-lógica

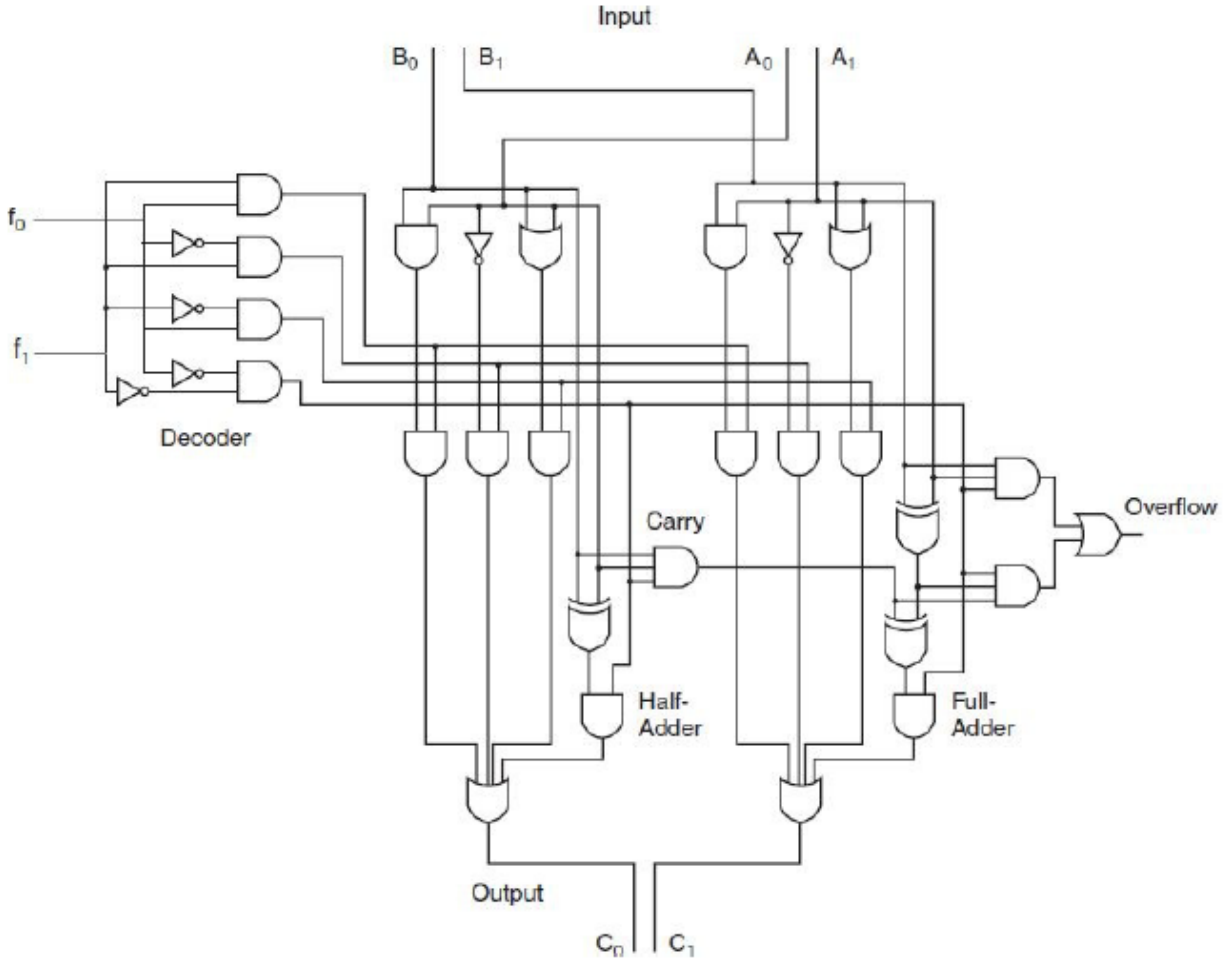


Figura 2.2 Simple ALU de 2 bits. [2]

Capítulo 2

Unidad de control

- Es la unidad encargada de “manejar el tráfico” del CPU monitoreando todas las instrucciones y la transferencia de información.
- Este módulo extrae la información de la memoria para asegurarse que los datos estén en el lugar correcto en el momento correcto. Controla a la ALU para el funcionamiento deseado.
- La unidad de control utiliza un registro de contador de programa para encontrar la instrucción que sigue para la ejecución y un registro de estado para realizar un seguimiento de los desbordamientos, acarreos, entre otros.

Capítulo 2

Unidad de control cableada

- Es aquella que conecta las líneas de control a las instrucciones de la máquina.
- Las instrucciones están divididas por campos y estos están divididos por bits, los cuales están conectados a las líneas de entrada de los diferentes circuitos lógicos.
- Existen 3 componentes esenciales:
  - El decodificador de la instrucción.
  - El contador del ciclo.
  - La matriz de control.

Capítulo 2

Unidad de control cableada

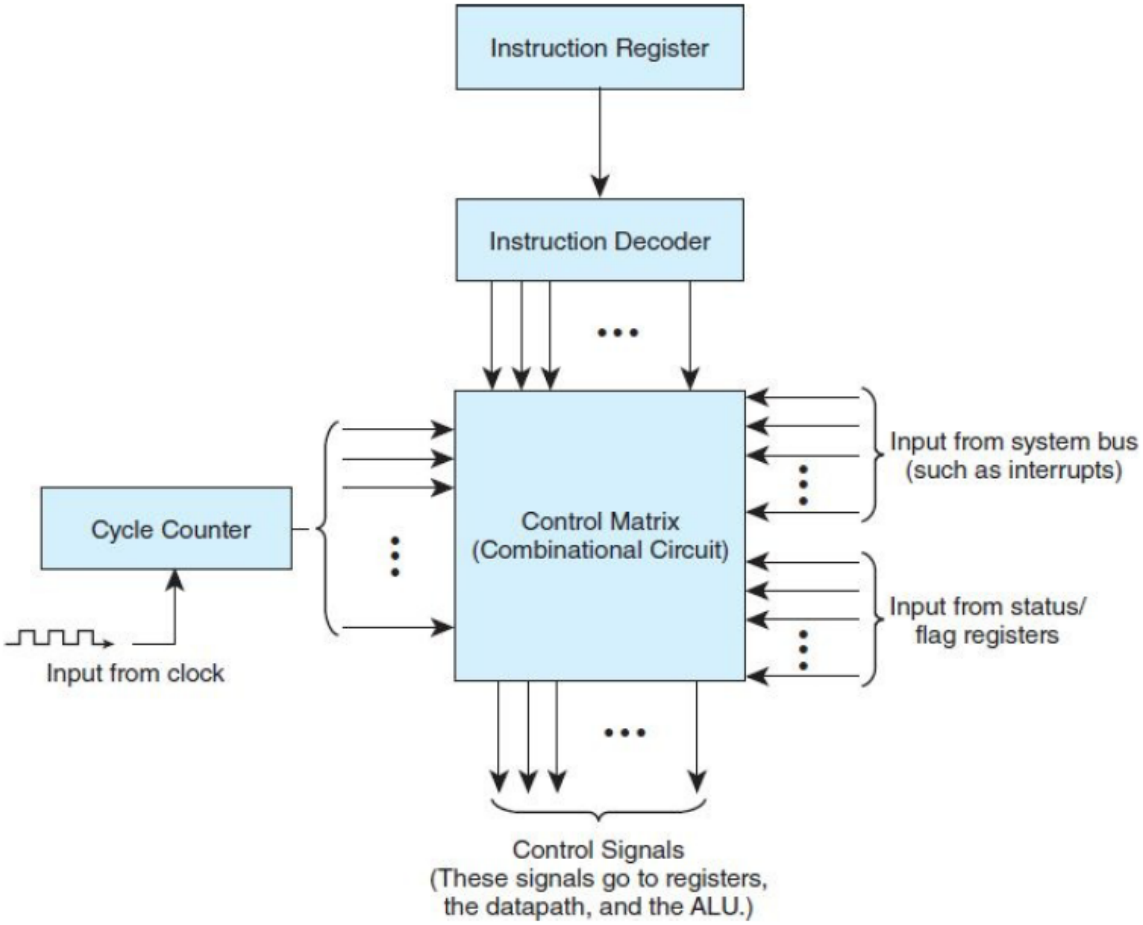


Figura 2.3 Estructura de la unidad de control cableada. [2]

Capítulo 2

Unidad de control micro-programada

- Este enfoque emplea el software basado en microinstrucciones que operan las instrucciones de las microoperaciones.
- Las señales controlan el movimiento de bytes a lo largo de la ruta del sistema informático.
- A diferencia del control cableado (dependencia de las compuertas del circuito lógico), el control micro-programado tiene un micro-código de instrucción que produce las señales de control necesarias.

Capítulo 2

Unidad de control micro-programada

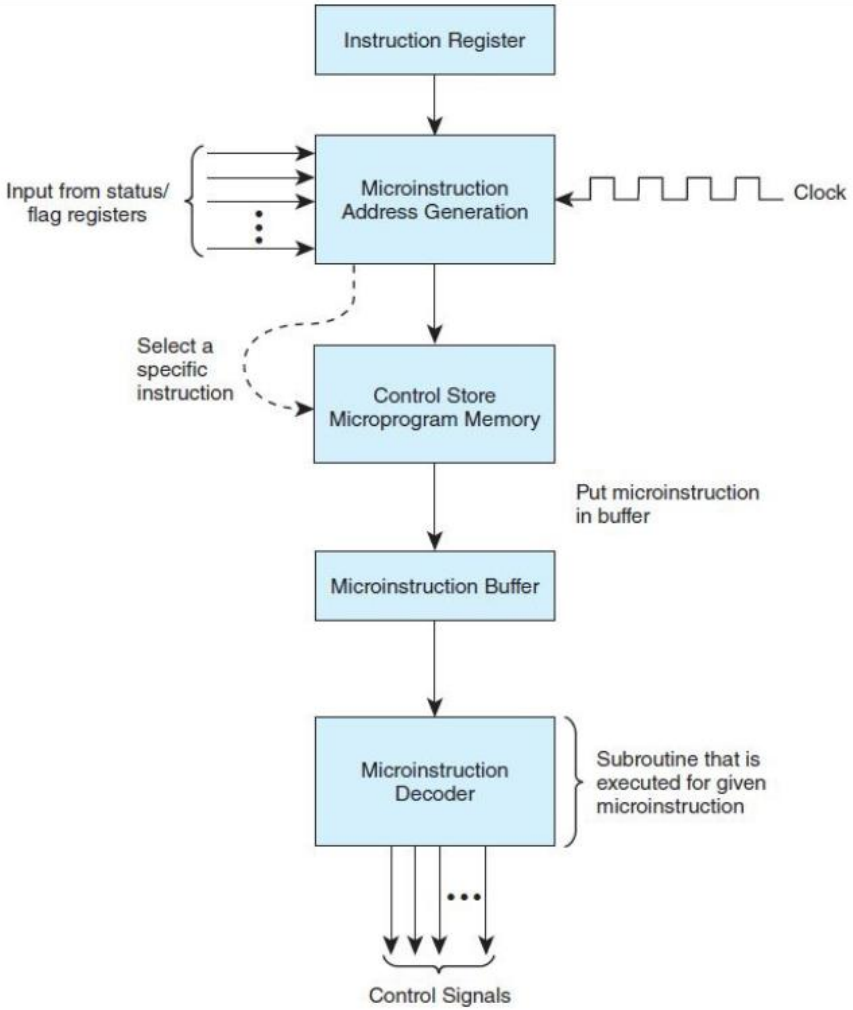


Figura 2.4 Estructura de la unidad micro-programada. [2]

Capítulo 2

Arquitectura CISC (Complex Instruction Set Computer)

- Es un diseño filosófico de arquitectura usado por cada miembro de la familia x86 de Intel.
- Principales características: gran número de instrucciones de longitud variable y diseños complejos.
- Mantenía las instrucciones cortas; sin embargo, también almacenada grandes y complejas instrucciones.
- Contenía un gran número de instrucciones que accedían directamente de la memoria, siendo las complejas las que requieren miles de ciclos, añadiendo tiempo de ejecución innecesariamente.

Capítulo 2

Arquitectura CISC (Complex Instruction Set Computer) Stalling 504

- La labor del escritor de compiladores es generar una secuencia de instrucciones máquina para cada sentencia de alto nivel; sin embargo, las instrucciones complejas son difíciles de aprovechar ya que no se sabe cuales se ajustan perfectamente a la construcción del programa.
- Su objetivo era que se produzcan programas pequeños y rápidos; sin embargo, con el tiempo, las ventajas de estos objetivos no terminaban siendo tan importantes.

Capítulo 2

Arquitectura RISC (Reduce Instruction Set Computer)

- Busca reducir el número de instrucciones; sin embargo, su real objetivo es simplificar las instrucciones para que el programa se pueda ejecutar más rápido, por lo que solo *load* y *store* pueden acceder a memoria.
- Cada instrucción realiza una sola operación, todas tienen el mismo tamaño, solo tienen algunos diseños diferentes y todas las operaciones aritméticas deben realizarse entre registros.
- Ofrece un set de instrucciones menor que CISC.

Capítulo 2

CISC vs RISC

RISC	CISC
Multiples set de registros, >256	Simple set de registros, entre 6 y 16 registros en total
3 registros operados por instrucción	1 o 2 registros por instrucción
1 ciclo por instrucción (except load y store)	Múltiples ciclos por instrucción
Unidad de control cableado	Unidad de control micro-programada
Altamente paralelizable	Bajamente paralelizable
Hay pocas instrucciones simples	Muchas instrucciones complejas
Complejidad en el compilador	Complejidad en el microcodigo
Solo load y store pueden accede a memoria	Muchas instrucciones pueden acceder a memoria
Pocos modos de direccionamiento	Muchos modos de direccionamiento

[2] Null, L., & Lobur, J. (2014). The essentials of computer organization and architecture. Jones & Bartlett Publishers.

Capítulo 2

Representación interna: Little Endian – Big Endian

- El término “Endian” se refiere al orden de los bytes.
- Es posible definir Little Endian y Big Endian a partir de un número entero que sea de dos bytes:
  1. En representación Little endian: El byte menos significativo primero seguido del byte más significativo. Por lo tanto, el byte en una dirección inferior tiene menor significado.
  2. En representación Big endian: El byte más significativo primero seguido del byte menos significativo. Por lo tanto, el byte en una dirección inferior tiene mayor significado.

Capítulo 2

Arquitectura interna de la familia de procesadores Intel x86 (16 bits)

- Primer chip: 8086 fue usado en una PC IBM.
- Maneja datos de 16 bits y 20 bits de direcciones.
- Tuvo 4 registros de propósito general de 16 bits cada uno:
  - AX (acumulador primario).
  - BX (Registro base para extender addressing).
  - CX (Registro contador).
  - DX (Registro de datos).
- Cada uno de estos dividido en 2 partes: High y Low.
- Tuvo 3 registros de puntero:
  - SP (Offset en la pila).
  - BP (Referencia de parámetros en la pila).
  - IP (contenía la dirección de la siguiente instrucción).

Capítulo 2

Arquitectura interna de la familia de procesadores Intel x86 (16 bits)

- 2 registros de índice:
  - SI (Puntero fuente para operaciones string).
  - DI (Puntero de destino para operaciones string).
- Registro de estado de bandera: Indica condiciones como overflow, paridad, acarreo interrumpido, entre otros.

- En 1980, Intel introdujo a 8087, el cual añade instrucciones de punto flotante a la 8086 así como una pila de 80 bits de ancho.
- A mediados de los 80, Intel introdujo el 80386 el cual fue la primera familia de 32 bits (IA-32). Los diseñadores querían que sea *backward compatible* para que los programas de la 80286 corran en 80386.
- Las versiones 80386 y 80486 eran de 32 bits con buses de 32 bits. Adicionalmente, la versión 80486 tuvo una memoria caché de mayor velocidad.

Capítulo 2

Arquitectura interna de la familia de procesadores Intel x86

- La serie Pentium tuvo un procesador de registros de 32 bits y un bus de data de 64 bits empleando el diseño superescalar.
- La Pentium IV introdujo el concepto de hyperthreading (multihilos). Esto mejoró la eficiencia del procesador lo cual evita que esté menos tiempo sin realizar acciones.
- En el 2001, se introdujo el procesador Itanium el cual trajo el primer chip de 64 bits de Intel (IA-64). Este procesador emplea un emulador de hardware para mantener la compatibilidad hacia atrás con los conjuntos de instrucciones IA-32/ x86.

Capítulo 2

Arquitectura interna de la familia de procesadores Intel x86 (32 bits)

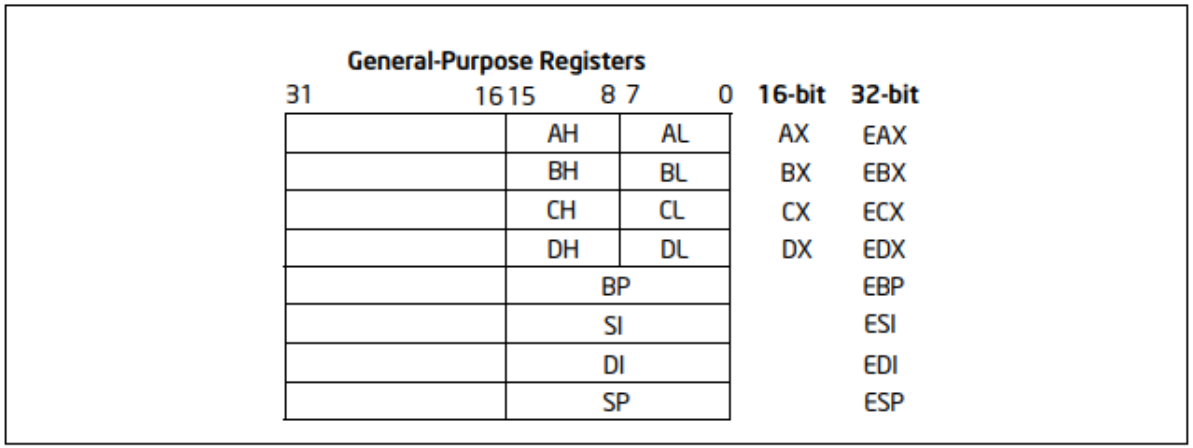


Figura 2.5 Registros de propósito general [3].

- Los registros de 32 bits fueron extendidos agregando el prefijo “E” (extended).
- Los registros de propósito general de 32 bits EAX, EBX, ECX, EDX, ESI, EDI, EBP y ESP se proporcionan para contener los siguientes elementos:
  - Operandos para operaciones lógicas y aritméticas.
  - Operandos para cálculos de direcciones.
  - Punteros de memoria.
- Se debe tener especial cuidado con el registro ESP ya que contiene el puntero de pila y como regla general, no debe utilizarse para otro propósito.

Capítulo 2

Arquitectura interna de la familia de procesadores Intel x86 (32 bits)

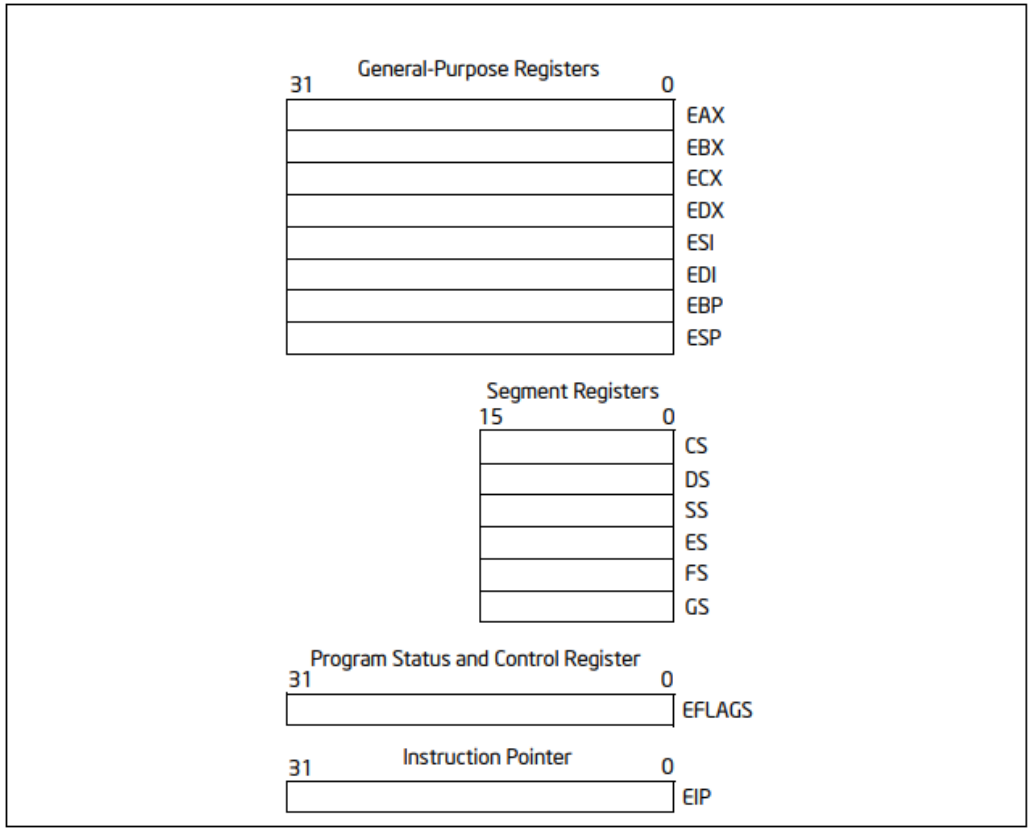


Figura 2.6 Registros generales de programación de sistemas y aplicaciones [3].

Capítulo 2

Arquitectura interna de la familia de procesadores Intel x86 (32 bits)

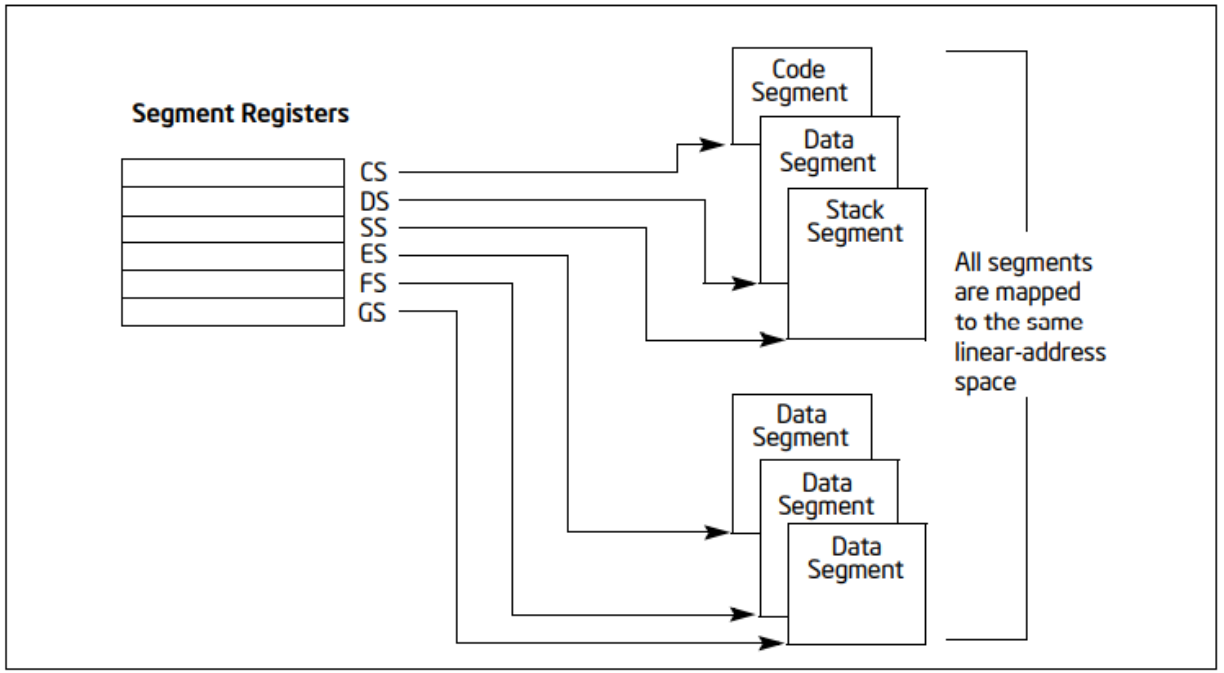


Figura 2.7 Registros de segmentos [3].

- Cada uno de los registros de segmento está asociado con uno de los tres tipos de almacenamiento: código, datos o pila.

Capítulo 2

Modos de operación

Mode	Submode	Operating System	Default Address (bits)	Default int (bits)
IA-32e or Long	64-bit	64-bit	64	32
	Compatibility		32	
			16	16
Legacy	Protected	32-bit	32	32
	Virtual-8086		16	16
	Real	16-bit		

Figura 2.8 Modos de operación de x86-64.[4].

- Los términos IA-32e (Intel) y Long (AMD) se usan en sistemas operativos de 64 bits.
- En el submodo “Compatibility” la mayoría de programas compilados para 32 bits o 16 bits se pueden ejecutar sin volver a compilar.
- En el submodo “64-bit” el programa debe ser compilado para la ejecución en 64 bits.

Capítulo 2

Arquitectura interna de la familia de procesadores Intel x86 (64 bits)

	64-bits	32-bits	16-bits	8-bits
Acumulador	RAX	EAX	AX	AL
Base	RBX	EBX	BX	BL
Contador	RCX	ECX	CX	CL
Datos	RDX	EDX	DX	DL
Source Index	RSI	ESI	SI	SIL
Destination Index	RDI	EDI	DI	DIL
Base Pointer	RBP	EBP	BP	BPL
Stack Pointer	RSP	ESP	SP	SPL
Propósito general	R8-R15	R8D-R15D	D8W	D8B

Capítulo 2

Arquitectura interna de la familia de procesadores Intel x86 (64 bits)

Bandera	Descripción de la bandera
CF	Acarreo (Carry)
PF	Paridad (Parity)
ZF	Zero (Cero)
SF	Signo (Sign)
OF	Desborde (Overflow)
AF	Ajuste (Adjust)
IF	Interrupciónes (Interrupt Enable)

- RFLAGS es el registro que almacena las banderas usadas para resultados de operaciones y como controlador del procesador. Estas banderas están formadas de las EFLAGS de los registros del x86 de 32-bit. Adicionalmente, se añaden los 32 bits superiores que normalmente se encontraban reservadas y en aquellos tiempos inutilizadas.

Capítulo 2

Arquitectura interna de la familia de procesadores Intel x86 (64 bits)

- Existen una serie de registros específicos que se usar para operaciones de coma flotante de 32 y 64 bits e instrucciones de Datos Múltiples de Instrucción Única (SIMD).
- Las instrucciones SIMD permiten que una sola instrucción se aplique simultáneamente a múltiples elementos de datos (mayor rendimiento).
- Las aplicaciones típicas incluyen procesamiento de gráficos y procesamiento de señales digitales.
- Algunos procesadores X86-64 más recientes admiten registros XMM de 256 bits.
- Los registros XMM se utilizan para admitir las Extensiones SIMD de transmisión por secuencias (SSE).

Registros de 128 bits	XMM0	XMM1	XMM2	XMM3	...	XMM12	XMM13	XMM14	XMM15

Capítulo 2

¿Porqué aprender lenguaje en ensamblador?

Ventajas	Desventajas
Propósitos educacionales	Tiempo de desarrollo
Depuración y verificación	Depuración y verificación
Realizar compiladores	Fiabilidad y seguridad
Sistemas embebidos	Dificultad en modificar y mantener
Controladores de hardware y códigos de sistemas	Portabilidad
Instrucciones no accesibles por alto nivel	Compiladores han mejorado los últimos años
Optimización por tamaño y velocidad	Códigos de sistemas usan funciones intrínsecas (C++)

Capítulo 2

Programación en lenguaje ensamblador – Formato

- El formato del código de ensamblador varía dependiendo la arquitectura que se use así como el ensamblador a utilizar.

- Generalmente, un programa en ensamblador puede subdividirse en las siguientes 3 secciones:
  - Sección data : Declaración y definición de data inicial.
    - db, dw, dd, dq.
  - Sección bss : Declaración y definición de data sin valor inicial.
    - resb, resw, resd, resq.
  - Sección text : Código de operación.

- Llamadas al sistema: Los *syscall* cambian el CPU a modo kernel lo que permite ejecutar una función de lectura/escritura. Los parámetros de las llamadas al sistema hacen uso de registros específicos (i.e. RAX) y se le asigna un código que le permite interactuar con el sistema.

Capítulo 2

Programación en lenguaje ensamblador – Formato

- Dependiendo si se trabaja con el formato de Intel o AT&T se podrán brindar características específicas de la codificación.

- Por ejemplo, utilizando un registro de 32 bits, la sintaxis de AT&T utiliza el orden opuesto para los operandos de origen y de destino:

  – Intel: mov eax, 4

  – AT&T: movl $4,%eax

- Por ese motivo, la forma de compilar el programa se regirá por la sintaxis en que es codificada.

Capítulo 2

Programación en lenguaje ensamblador – (Ejemplo 32 bits)

Línea		Intel		AT&T	
1	001	1	; Inicio del segmento de texto	1	# Inicio del segmento de texto
2	002	2	section .text	2	.section .text
3	003	3		3	
4	004	4	global _start	4	.globl _start
5	005	5		5	
6	006	6	; Punto de entrada al programa	6	# Punto de entrada al programa
7	007	7	_start:	7	_start:
8	008	8		8	
9	009	9	; Número para la llamada al sistema	9	# Número para la llamada al sistema
10	010	10	mov eax, 1	10	movl $1, %eax
11	011	11		11	
12	012	12	; Regresa el valor	12	/* Regresa el valor*/
13	013	13	mov ebx, 2	13	movl $2, %ebx
14	014	14		14	
15	015	15	; Llamar al Sistema Operativo	15	# Llamar al Sistema Operativo
16	016	16	int 80h	16	int $0x80

Capítulo 2

Programación en lenguaje ensamblador – (Ejemplo 32 bits)

Línea		Intel		AT&T	
1	001	1	; Inicio del segment de data	1	# Inicio del segment de data
2	002	2	section .data	2	.section .data
3	003	3		3	
4	004	4	var1 dd 40	4	var1:
5	005	5		5	.int 40
6	006	6	var2 dd 20	6	var2:
7	007	7		7	.int 20
8	008	8	var3 dd 30	8	var3:
9	009	9		9	.int 30
10	010	10		10	
11	011	11	section .text	11	.section .text
12	012	12		12	
13	013	13	global _start	13	.globl _start
14	014	14		14	
15	015	15	_start:	15	_start:
16	016	16		16	
17	017	17	; Mueve el contenido de las variables	17	# Mueve el contenido de las variables
18	018	18	mov ecx, [var1]	18	movl (var1), %ecx
19	019	19	cmp ecx, [var2]	19	cmpl (var2), %ecx
20	020	20	jg check_third_var	20	jg check_third_var
21	021	21	mov ecx, [var2]	21	movl (var2), %ecx
22	022	22		22	
23	023	23	check_third_var:	23	check_third_var:
24	024	24	cmp ecx, [var3]	24	cmpl (var3), %ecx
25	025	25	jg _exit	25	jg _exit
26	026	26	mov ecx, [var3]	26	movl (var3), %ecx
27	027	27		27	
28	028	28	_exit:	28	_exit:
29	029	29	mov eax, 1	29	movl $1, %eax
30	030	30	mov ebx, ecx	30	movl %ecx, %ebx
31	031	31	int 80h	31	int $0x80

Capítulo 2

Programación en lenguaje ensamblador – Toolchains

- El toolchain es la cadena completa de programas que se utilizan para convertir el código fuente en código de máquina binario, vincular entre sí los módulos de código ensamblados / compilados, desensamblar los binarios y convertir sus formatos.

- Inicializa su composición con lo siguiente:

1. El ensamblador convierte los programas en lenguaje ensamblador legibles por humanos en código binario de lenguaje de máquina. Genera los archivos de objetos .o.

2. El linkeador se utiliza para combinar varios archivos de objetos resolviendo sus referencias de símbolos externos y reubicando sus secciones de datos, y generando un único archivo ejecutable. Por lo general, toma como entrada los archivos de objeto .o y los scripts del vinculador .ld

- Ejemplos: “Hola Intel” “Hola AT&T”.

Programación en lenguaje ensamblador

Modos de direccionamiento:

Las operaciones se realizan entre registros o registros y memoria pero no entre memoria y memoria (salvo algunas operaciones con cadenas de caracteres). Los modos de direccionamiento determinan el lugar en que reside un operando, un resultado o la siguiente instrucción a ejecutar (según sea el caso).

Capítulo 2

Programación en lenguaje ensamblador

Ejemplo: Instrucción MOV destino, fuente

1.- Direccionamiento inmediato: El operando aparece directamente en la instrucción:

MOV AX, 1234H

2. Modo registro: El operando es un registro:

MOV AX, BX

3. Directo absoluto a memoria: El operando es una dirección de memoria a la que se desea acceder

MOV AX, [078AH]

***Se copia en el registro AX, el contenido de la dirección DS:078AH

Capítulo 2

Programación en lenguaje ensamblador

4.- Directo relativo a un registro base: El operando es una dirección de memoria a la que se desea acceder y se calcula mediante un registro base (BX o BP si es segmento DS o SS, respectivamente)

MOV AX, [BX+2]

***** Se copia en AX el contenido de la dirección DS: BX+2

5. Directo relativo a un registro índice: El operando es una dirección de memoria a la que se desea acceder, y se calcula en base a un registro índice. Este registro índice es SI o DI.

MOV AX, [SI+10]

***** Se copia en AX el contenido de la dirección DS: SI+10

6. Indexado a partir de una base: El operando es una dirección de memoria a la que se desea acceder, y se calcula en base a un registro base (BX o BP) y un registro índice(SI o DI). Este registro índice es SI o DI.

MOV [BP+DI+2], AX

*****Copia en la dirección SS:BP+DI+2 el contenido de AX

Capítulo 2

Manejo de memoria Intel x86 (32 bits)

Segmentación

- Cada dirección lógica (terminología en el manual de Intel para DV) consiste en una referencia de segmento de 16 bits y un desplazamiento de 32 bits.
- Dos bits de la referencia del segmento tratan con el mecanismo de protección, dejando 14 bits para especificar un segmento en particular. Por lo tanto, con la memoria no segmentada, la memoria virtual del usuario es de 4 GB.
- Con la memoria segmentada, el espacio total de memoria virtual visto por un usuario es de 64 TB. El espacio de direcciones físicas emplea una dirección de 32 bits para un máximo de 4 GB.

Capítulo 2

Memoria virtual – Ejemplo de manejo de memoria Intel x86 (32 bits)

- Asociadas a cada segmento hay dos formas de protección: nivel de privilegio y atributo de acceso. Hay cuatro niveles de privilegio, desde la mayoría protegida (nivel 0) hasta la menos protegida (nivel 3).
- El nivel de privilegio asociado con un segmento de datos es su "clasificación“.
- El nivel de privilegio asociado con un segmento del programa es su "autorización".
- Un programa en ejecución solo puede acceder a los segmentos de datos para los cuales su nivel de autorización es inferior a (más privilegiado) o igual (el mismo privilegio) al nivel de privilegio del segmento de datos.

Capítulo 2

Memoria virtual – Ejemplo de manejo de memoria Intel x86 (32 bits)

- El atributo de acceso de un segmento de datos especifica si se permiten los accesos de lectura / escritura o de solo lectura. Para los segmentos del programa, el atributo de acceso especifica acceso de lectura / ejecución o de solo lectura.
- El mecanismo de traducción de direcciones para la segmentación implica la asignación de una dirección virtual a lo que se conoce como una dirección lineal.
- Una dirección virtual consiste en el desplazamiento de 32 bits y un selector de segmento de 16 bits. Una instrucción que busca o almacena un operando especifica el desplazamiento y un registro que contiene el selector de segmento.

Capítulo 2

Ejemplo de manejo de memoria Intel x86 (32 bits)

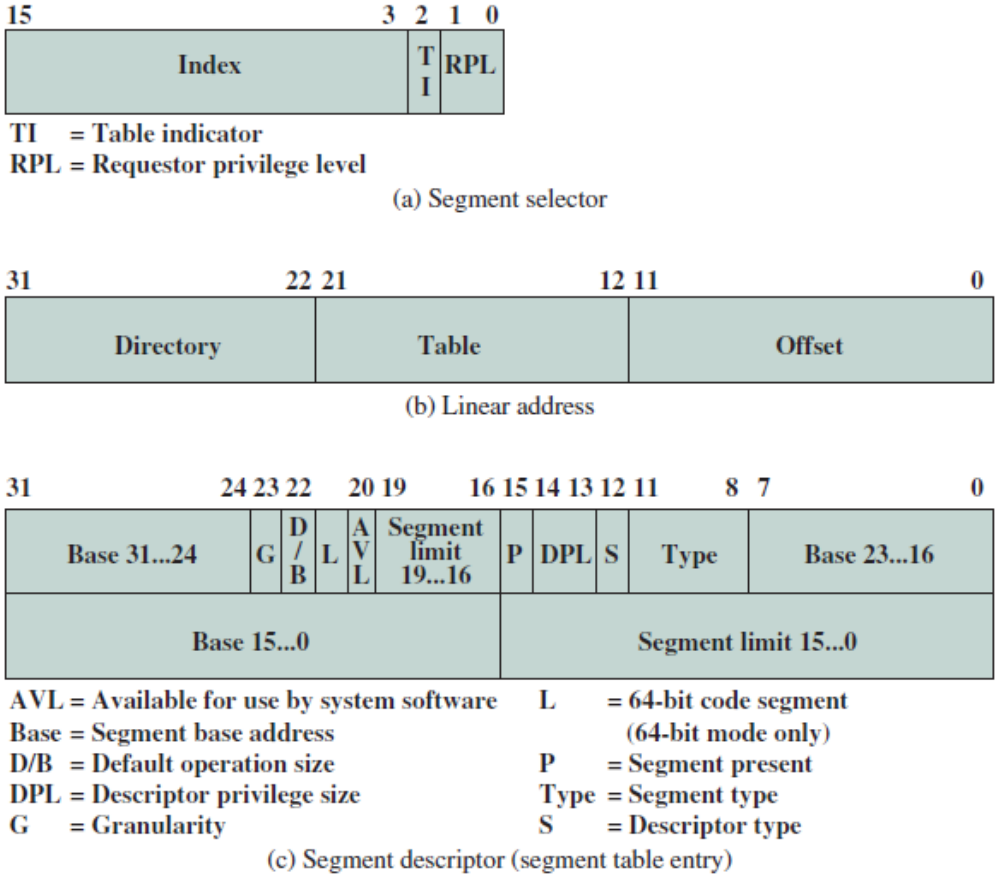


Figura 2.9. Ilustración de la combinación de segmentación y paginación (sin TLB ni memoria) [1].

Capítulo 2

Unidades de rendimiento - Ciclos por instrucción

- Es la medida que nos permite identificar cuántos ciclos le toma al procesador ejecutar una instrucción individual.
- Se define la siguiente expresión:

$$CPI = \frac{\sum_{i=1}^{n} CPI_i \; I_i}{I_c}$$

Siendo I_c el total de instrucciones del programa, I_i el número de instrucciones de un tipo de programa y CPI_i el ciclo por instrucción de un tipo de programa.

Capítulo 2

Unidades de rendimiento - Tiempo de ejecución

- Sabiendo cuál es el valor del CPI, se puede hallar el tiempo de ejecución de un procesador de la siguiente manera:

$$T = I_c \,.\, CPI . \tau$$

Siendo I_c el total de instrucciones del programa, CPI el ciclo por instrucción del programa y τ el periodo del ciclo del CPU.

Unidades de rendimiento

Ejemplo:

- Supongamos dos computadoras con la misma arquitectura de repertorio de instrucciones y el mismo número de instrucciones:

Máquina	Tiempo de ciclo de reloj	CPI para un programa específico
A	250 ps	2.0
B	500 ps	1.2

- ¿Qué máquina es más rápida para este programa? ¿Cuánto más rápida es?

Unidades de rendimiento – MIPS y MFLOPS

MIPS (Millions of instructions per second)*

- Es otro indicador para medir el rendimiento del procesador basado en la cantidad de instrucciones ejecutadas en un segundo.

$$MIPS = \frac{I_c}{T.10^6} = \frac{f}{CPI.10^6}$$

FLOPS (Floating-point operation per second)

- Indicador de rendimiento para operaciones en coma flotante.

$$FLOPS = \frac{\#\ Operaciones\ en\ coma\ flotante}{Tiempo\ de\ ejecución}$$

*No confundir el término con el lenguaje ensamblador MIPS

Capítulo 2

Operaciones en un ciclo y en multiciclo

- Para completar cualquiera de las instrucciones es necesario que se siga el circuito del datapath; sin embargo, no todas los componentes pueden tomar más de un ciclo en ejecutar cualquier instrucción (memorias).

- ¿Es posible lograr ejecutar instrucciones en paralelo?

- ¿Por qué no se diseña el circuito para que opere en un solo ciclo?

Capítulo 2

Segmentación en el camino de datos (Pipeline)

- Hasta ahora, es razonable relacionar que cada instrucción ejecutada puede ser asociada a un ciclo de reloj; sin embargo, a veces es posible que se usen pequeños pulsos de control entre instrucción.
- Esto es conocido como el método de nivel de instrucción en paralelo.
- Para mayor entendimiento, se puede decir que la instrucción se puede dividir en pequeños pasos y al terminar la secuencia de estos pasos se puede dar por finalizada la instrucción.

Capítulo 2

Segmentación en el camino de datos (Pipeline)

- A partir de aquí, dividiremos una instrucción en 5 pasos:
  1. Instruction Fetch (IF)
  2. Instruction Decode (ID)
  3. Execute (EX)
  4. Memory Access (MEM)
  5. Write back (WB)

Capítulo 2

Segmentación en el camino de datos (Pipeline)

- Para un mejor entendimiento, tomaremos el ejemplo del “lavado de ropa”.
- Si dividimos este proceso en 4 etapas, tendríamos lo siguiente:

1. Colocar la ropa sucia en la lavadora.
2. Colocar la ropa mojada limpia en la secadora.
3. Colocar la ropa seca en la tabla para doblarlar.
4. Guardar la ropa en el cajón.

Capítulo 2

Segmentación en el camino de datos (Pipeline)

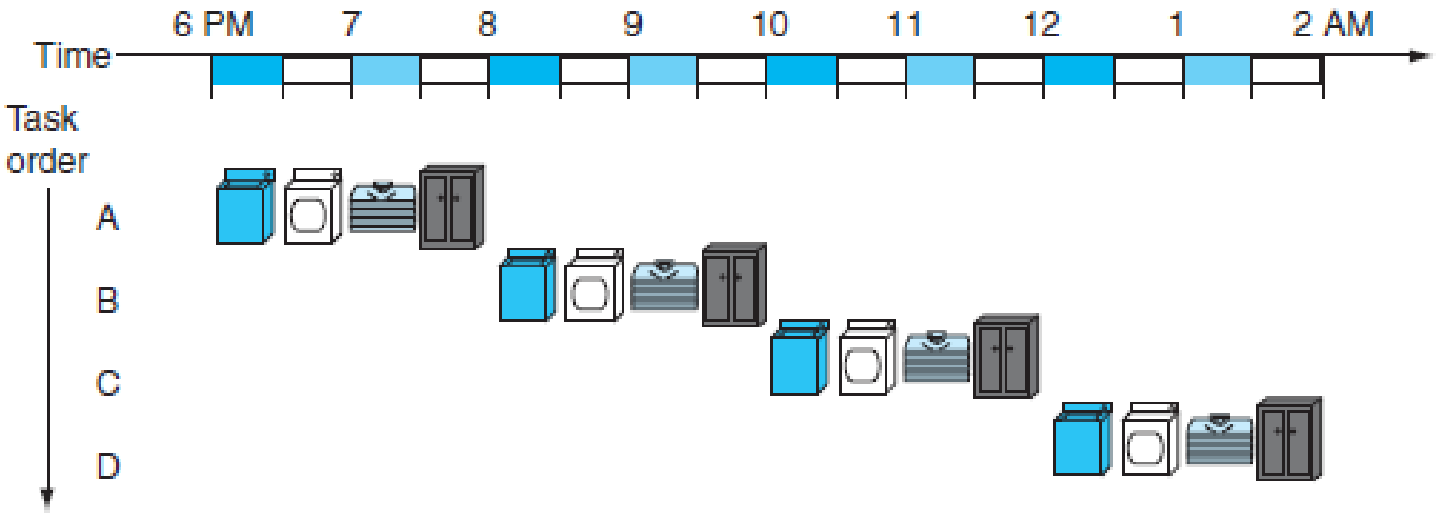


Figura 2.10 Ejemplo del lavado de ropa secuencial sin paralelismo. [5]

¿Qué sucede si hacemos este proceso en paralelo?

Capítulo 2

Segmentación en el camino de datos (Pipeline)

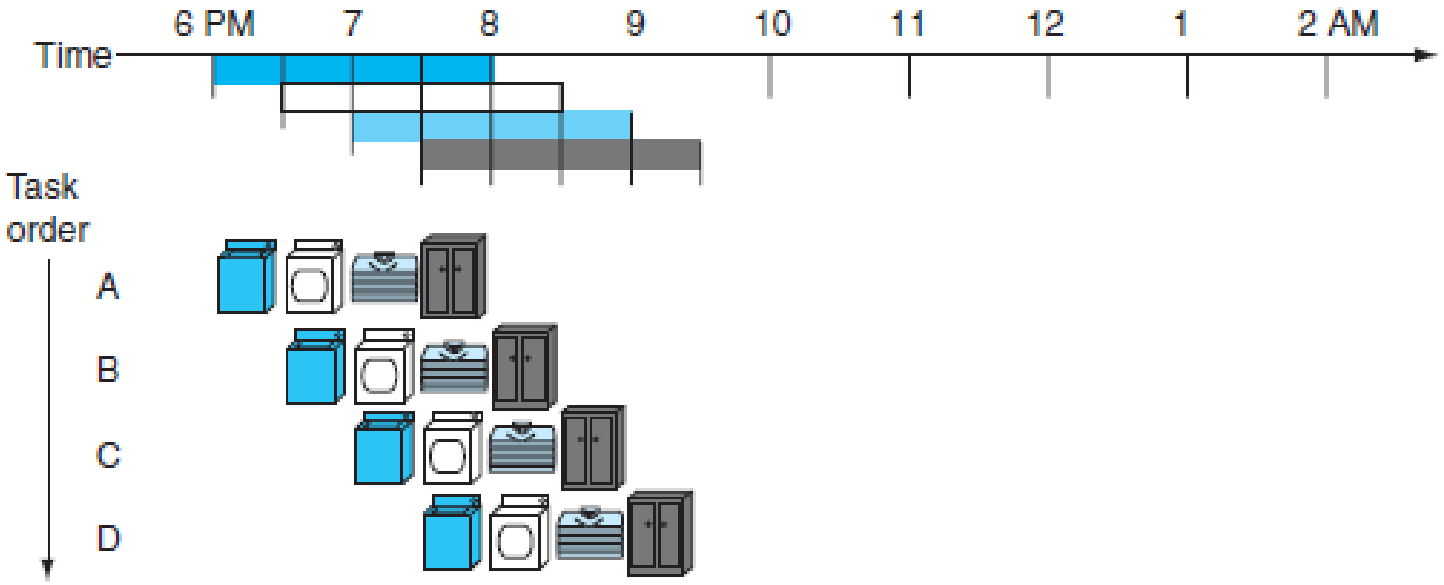


Figura 2.11 Ejemplo del lavado de ropa con paralelismo. [5]

Capítulo 2

Segmentación en el camino de datos (Pipeline)

Tabla 2.1 Instrucciones en MIPS (Microprocessor without Interlocked Pipeline Stages) assembler. [5]

Instruction class	Instruction fetch	Register read	ALU operation	Data access	Register write	Total time
Load word (`lw`)	200 ps	100 ps	200 ps	200 ps	100 ps	800 ps
Store word (`sw`)	200 ps	100 ps	200 ps	200 ps		700 ps
R-format (`add`, `sub`, `and`, `or`, `slt`)	200 ps	100 ps	200 ps		100 ps	600 ps
Branch (`beq`)	200 ps	100 ps	200 ps			500 ps

Capítulo 2

Segmentación en el camino de datos (Pipeline)

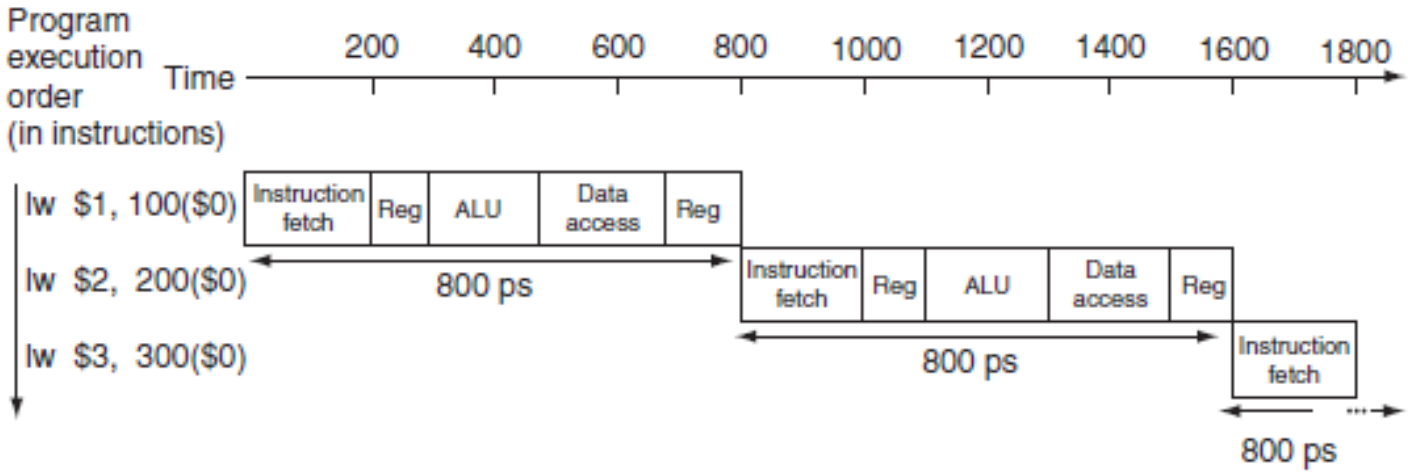


Figura 2.12 Ejemplo del programa sin pipeline. [5]

Capítulo 2

Segmentación en el camino de datos (Pipeline)

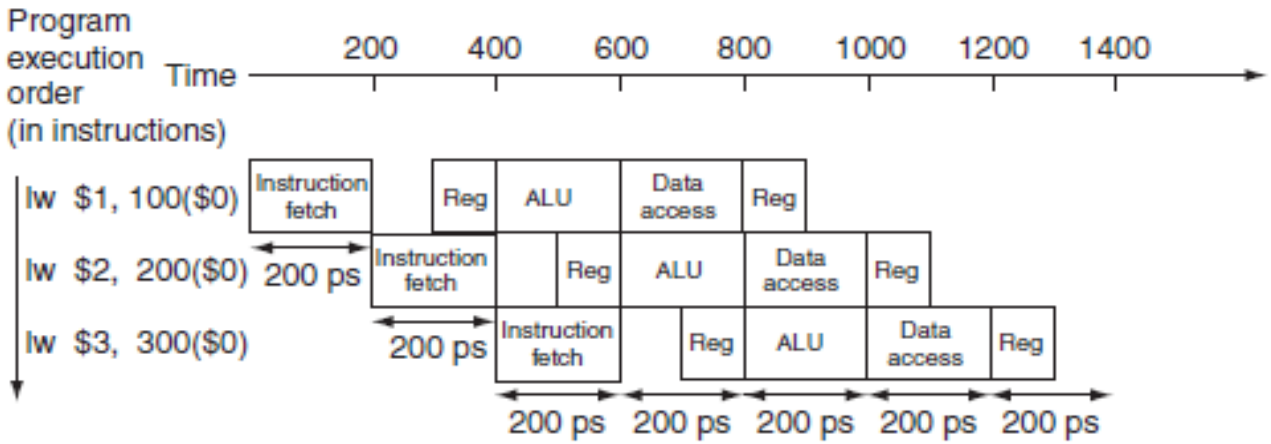
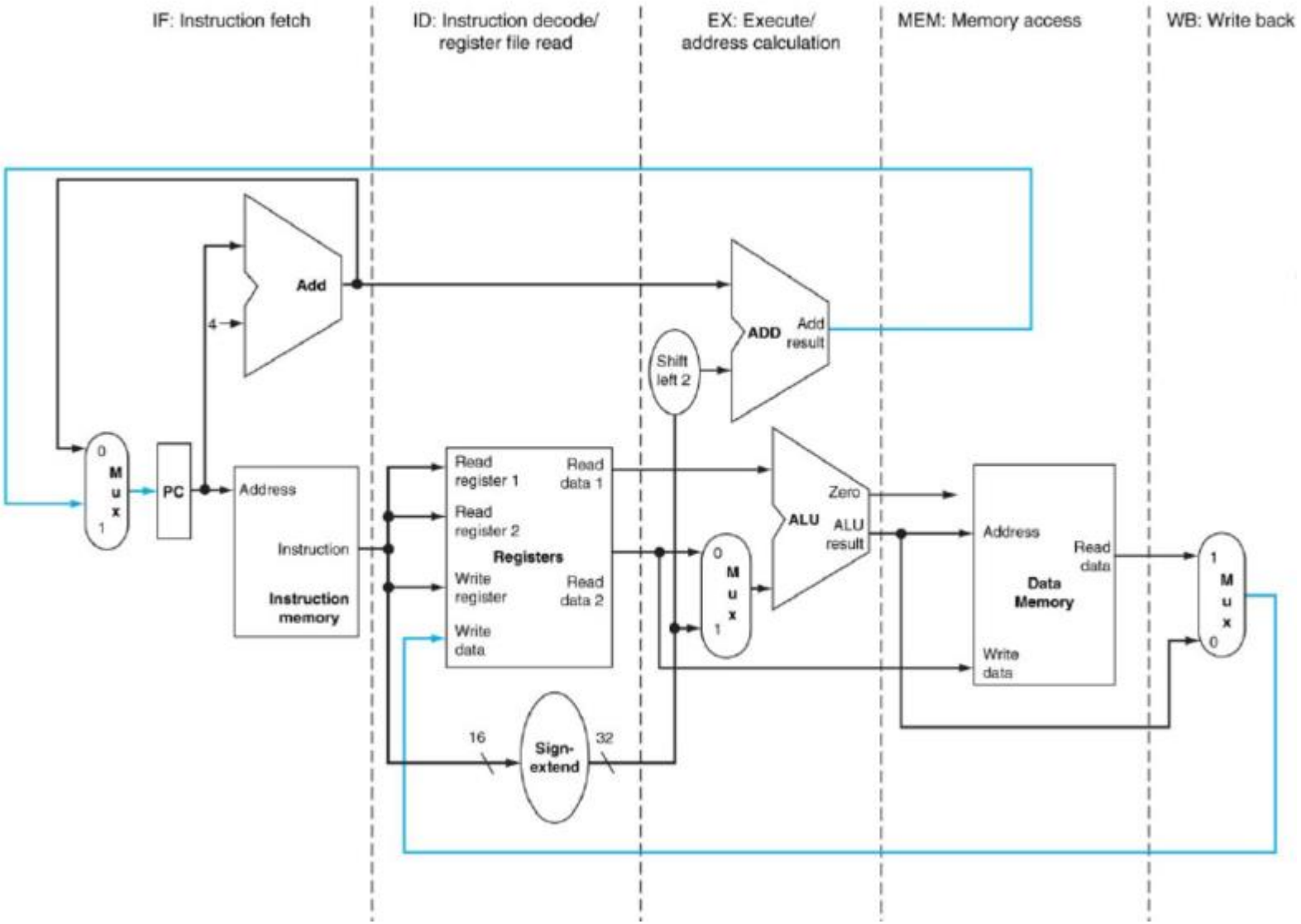


Figura 2.13 Ejemplo del programa con pipeline. [5]

Capítulo 2


IF: Instruction fetch
ID: Instruction decode/ register file read
EX: Execute/ address calculation
MEM: Memory access
WB: Write back
Add
4
M u x
PC
Address
Instruction
Instruction memory
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
Shift left 2
ADD
Add result
ALU
Zero
ALU result
Address
Data Memory
Read data
Write data
16
Sign-extend
32

Segmentación en el camino de datos (Pipeline)

- Suponiendo condiciones ideales podemos establecer la siguiente relación:

$$tiempo\ entre\ instrucciones_{segmentado} = \frac{tiempo\ entre\ instrucciones_{no\ segmentos}}{\#\ de\ etapas\ de\ segmentación}$$

- Esto sugiere que para 5 segmentos se debería tener 5 veces la velocidad; sin embargo, si armamos un set de instrucciones de puros *load word* veremos que las etapas del pipeline se encuentran desequilibradas (i.e. 200 ps y 100 ps) y el pipeline genera algún gasto adicional de tiempo. Esto implica que, en la realidad, el número de etapas no determina la ganancia de tiempo; sin embargo, para una gran cantidad de instrucciones, la ganancia termina siendo menor que el número de etapas.

Segmentación en el camino de datos (Pipeline)

- Entonces, es posible concluir que la segmentación mejora las prestaciones incrementando la productividad de las instrucciones en lugar de disminuir el tiempo de ejecución de cada instrucción individual, pero la productividad de las instrucciones es la métrica importante ya que los programas reales ejecutan millones de instrucciones.

Riesgos estructurales

Existen situaciones de segmentación en que no es posible ejecutar el ciclo siguiente de la instrucción que continua. A estos inconvenientes se les conoce como riesgos (*hazards*):

- Riesgos estructurales:
  - Un requerimiento del sistema está ocupado.
- Riesgos de datos:
  - Esperar a que la instrucción previa complete su lectura/escritura.
- Riesgos de control:
  - Decisiones de control que dependen de instrucciones previas

Capítulo 2

Supersegmentación

- Es posible dar una mejora en el rendimiento del procesador con pipeline al utilizar alternativas que permitan una interacción ciclos e instrucciones.
- En este caso, la supersegmentación se define como la realización del pipeline usando partes más pequeñas de los 5 pasos antes definidos.
- Esto contribuye a acercarnos más a instrucciones que se ejecuten en paralelo.

Capítulo 2

Superescalaridad

- Es capaz de realizar dos etapas del programa por ciclo del reloj. Esto significa que, explícitamente, tenemos dos instrucciones ejecutándose en paralelo; sin embargo, esto necesariamente conlleva a que se utilicen más recursos del hardware dado que se están utilizando más pipelines en paralelo.
- Es importante entender cómo se relacionan las instrucciones considerando sus posibles limitaciones.

Capítulo 2

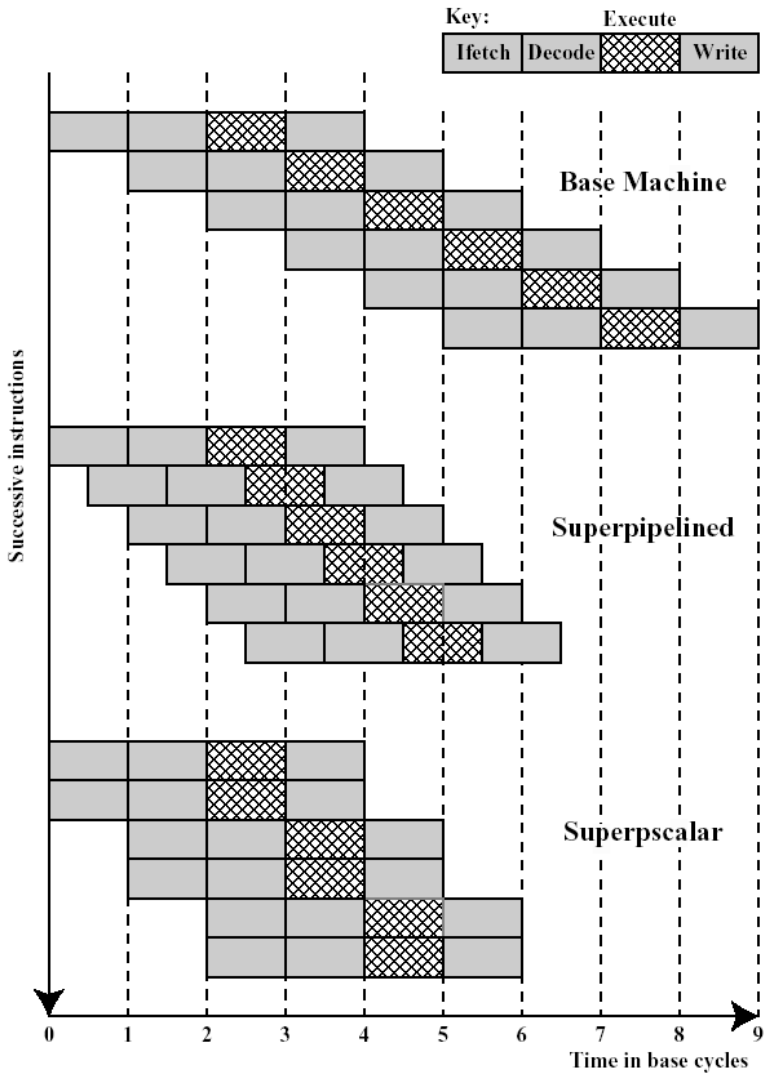


Figura 2.14 Pipeline vs supersegmentado vs superescalar. [1]

Capítulo 2

Benchmarks

- Considerando que los sets de instrucciones contienen una variedad de diferencias, el ratio de ejecución de instrucciones podría no ser la forma más adecuada para realizar la comparación del desempeño de diseños de arquitecturas.

- Por lo tanto, medidas de rendimiento como MIPS o MFLOPS podrían no ser las más adecuadas para este fin.

- Actualmente, una buena estimación con benchmark es capaz de identificar sistemas que brinden un “buen rendimiento” a “precios razonables”.

Capítulo 2

Benchmarks

- Considerando las discrepancias de las arquitecturas, el rendimiento de los sistemas se empezó a realizar con programas estandarizados (~1980). Estos servían como referencia para evaluar el rendimiento de un sistema.

- El objetivo concreto es que los programas puedan ser corridos en diferentes máquinas y generar un tiempo de ejecución que genere resultados comparables entre las máquinas participantes.

- Los benchmarks tienen como característica lo siguiente:
  - Lenguaje en alto nivel.
  - Estilo de programación específico (i.e. programación numérica).

Capítulo 2

Benchmarks

- Los primeros benchmarks demostraron lo fácil que era optimizar el rendimiento de los productos cuando un punto de referencia terminaba siendo sencillo y pequeño. Esto generó que los benchmark se vuelvan más complejos.
- SPEC (Standard Performance Evaluation Corporation) fue fundado en 1988 y tuvo como objetivo establecer una medida de rendimiento que sea real y equitativa.
- Actualmente, tiene 3 comités los cuales son los siguientes:
  - Open Systems Group (OSG): Workstations, el servidores y los entornos informáticos de escritorio.
  - High-Performance Group (HPG): Sistemas multiprocesadores y supercomputadoras de nivel empresarial.
  - Gráficos y Workstation Performance Group (GWPG): Desarrollo de gráficos y estaciones de trabajo.

Capítulo 2

Benchmarks

- Una versión que ha sido bien recibida es el SPEC CPU 2006. Cuenta con 17 programas de punto flotante en C, C++ y Fortran. Así mismo, tiene 12 programas escritos en C y C++.

- Actualmente, la alternativa más apropiada para obtener el tiempo de ejecución es utilizar la media geométrica de todos los ratios de tiempos de referencia.

$$r_G = \left(\prod_{i=1}^{n} r_i \right)^{1/n}$$

- Donde r_G es el ratio total, r_i es el ratio para el i-ésimo benchmark y n es el número de benchmark.

Capítulo 2

Benchmarks

Benchmark	Reference Time	Run Time	Ratio to Reference Time
400.perlbench	9,770	739	13.2
401.bzip2	9,650	920	10.5
403.gcc	8,050	814	9.89
429.mcf	9,120	1,161	7.86
445.gobmk	10,490	719	14.6
456.hmmer	9,330	697	13.4
458.sjeng	12,100	870	13.9
462.libquantum	20,720	1,342	15.4
464.h264ref	22,130	995	22.2
471.omnetpp	6,250	659	9.49
473.astar	7,020	732	9.59
483.xalancbmk	6,900	820	8.42

Figura 2.15 Lista de benchmarks en una máquina. [2]

Capítulo 2

Bibliografía

[1] Stallings, William. Computer organization and architecture: designing for performance. Pearson Education India, 2010.

[2] Null, L., et al. The essentials of computer organization and architecture. Jones & Bartlett Publishers, 2014.

[3] Guide, Part. "Intel® 64 and ia-32 architectures software developer's manual." Volume 3B: System programming Guide, Part 2 (2011).

[4] Plantz, R. "Introduction to Computer Organization." (2013).

(Free version http://bob.cs.sonoma.edu/my_book/IntroCompOrg_preview.pdf)

[5] Patterson, D. A., & Hennessy, J. L. (2009). Computer Organization and Design: The Hardware/Software Interface. Newnes.