Introducción a VLIW

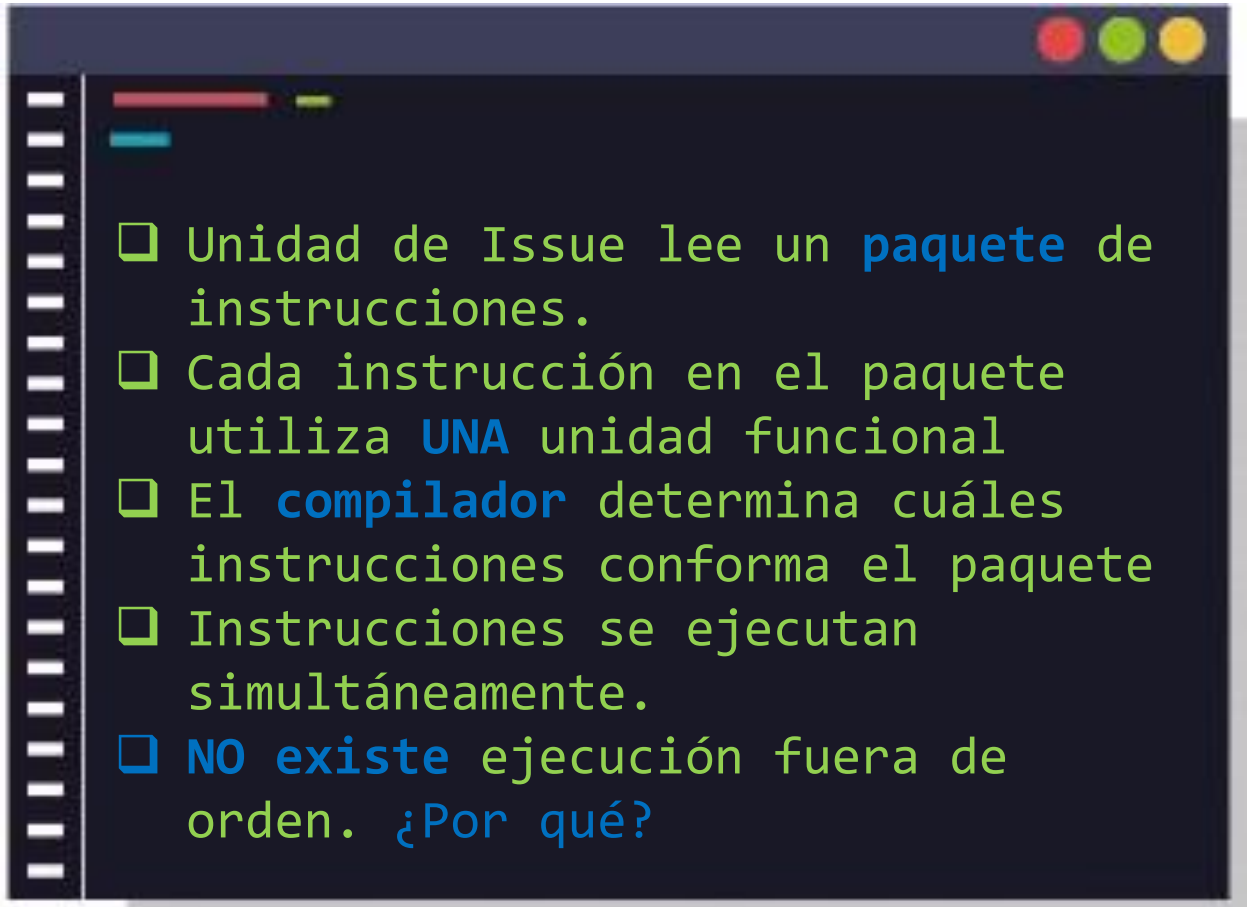
Agenda

- Descripción VLIW.
- Super escalar vs. VLIW.
- Arquitectura VLIW.
- Dependencias.
- Calendarización de código.

Descripción VLIW

Very Long Instruction Word

Técnica de paralelismo a nivel de instrucciones.



Procesadores VLIW

En procesadores Superescalares la calendarización de instrucción y detección de riesgos es realizada dinámicamente por hardware.

- **Idea básica:** Reducir el área y consumo de potencia haciendo que sea el compilador quién decide **estáticamente** las operaciones que se ejecutan en paralelo.
- El HW no necesita revisar explícitamente las dependencias. Es el compilador, quien debe asegurarse que no existan.

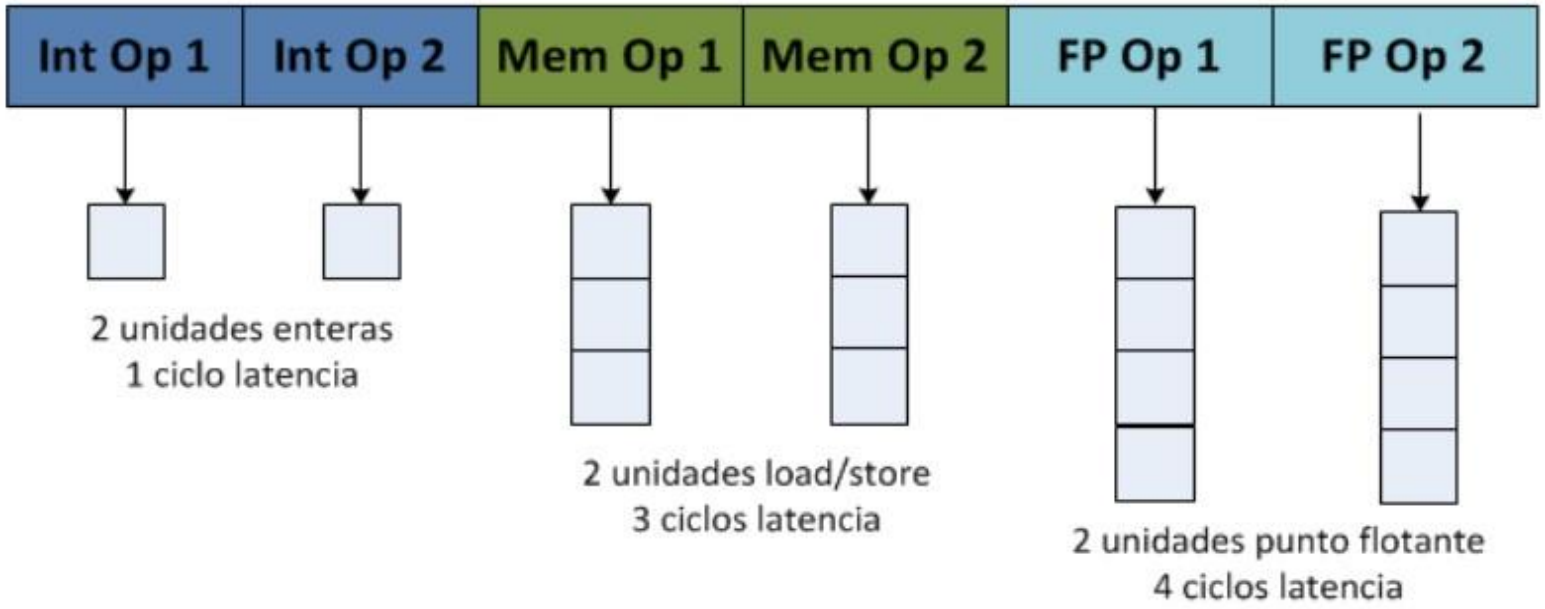
Supuestos Pipeline

- Hardware simplificado y menor consumo de potencia. El rendimiento principalmente viene determinado por optimizaciones en el compilador.

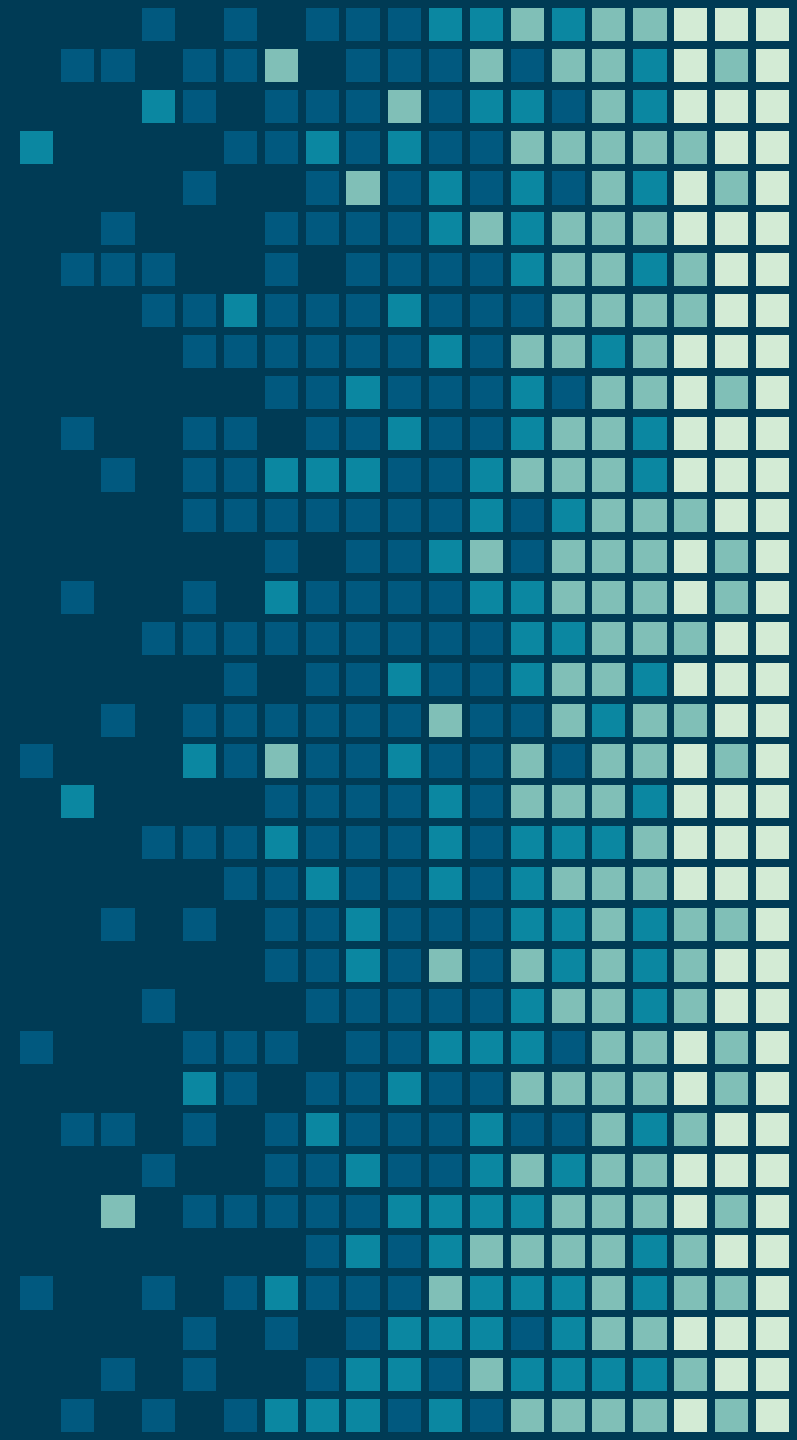
- Un paquete de instrucciones puede tener 64, 128, o más bits, con múltiples operaciones por instrucción: **Very Long Instruction Word**

- **Pipeline diversificado:** Las instrucciones en el paquete ("palabra" de instrucción) suelen ser de tipos distintos, lo que requiere segmentación paralela y **diversificada.**

- Múltiples operaciones en una sola instrucción. Unidades funcionales replicadas.
- La arquitectura que implemente VLIW requiere:
  - Paralelismo dentro de una misma instrucción (segmentación paralela + diversificada)
  - Los datos no son utilizados antes de que estén listos



Superescalar vs. VLIW



Superescalar vs. VLIW

Procesadores superescalares

- Ejecución dinámica y paralela cada ciclo de reloj.
- Flujo lineal de instrucciones (issue en orden).

Procesadores VLIW

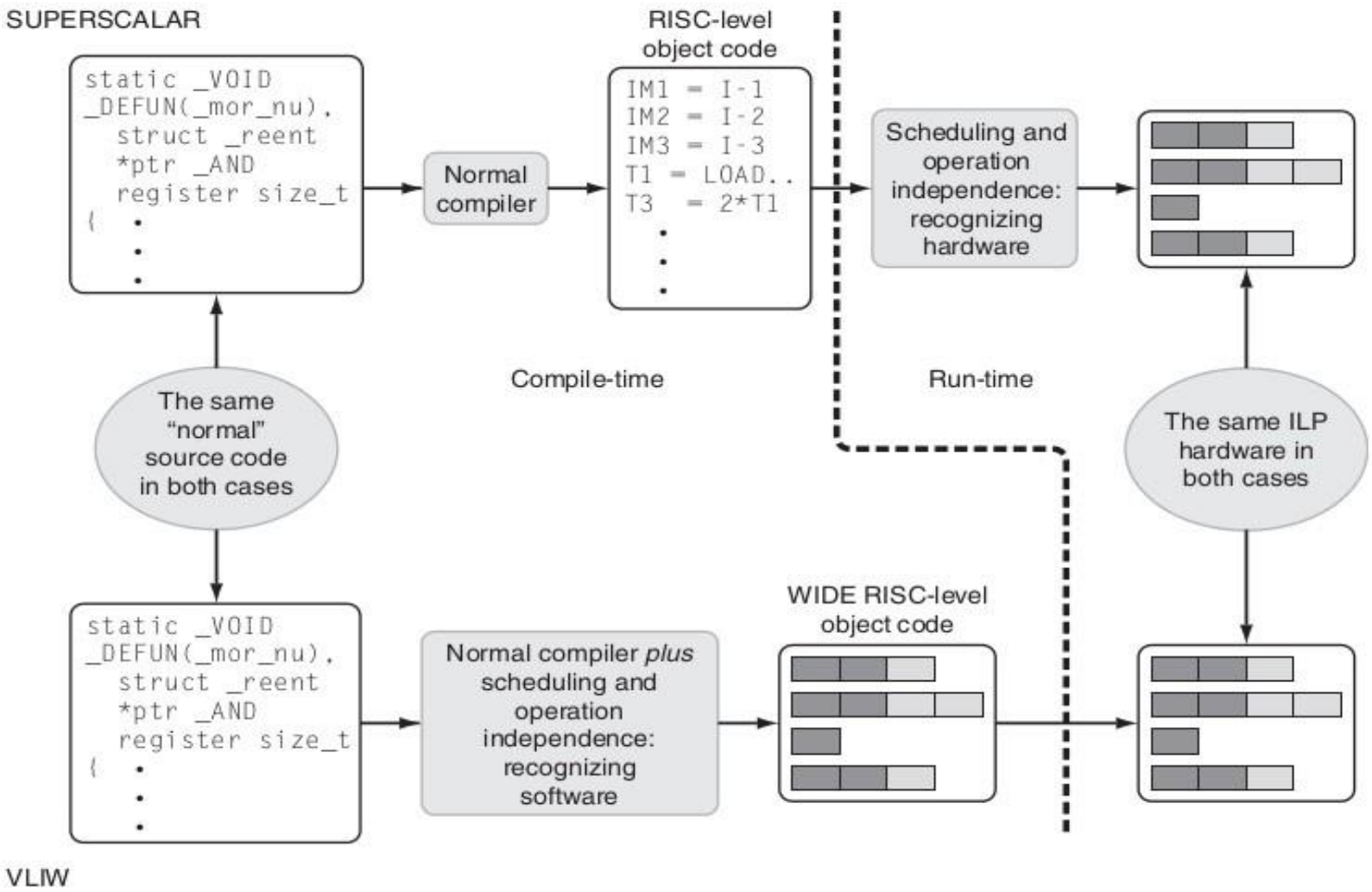
- Instrucción larga y estática.
- Segmentación diversificada y paralela. Issue en orden.

Superescalar vs. VLIW

Superescalar	VLIW
Las instrucciones son lanzadas de un flujo secuencial de operaciones escalares	Las instrucciones son lanzadas de un flujo secuencial de operaciones paralelas.
Instrucciones calendarizadas dinámicamente por el hardware	Instrucciones calendarizadas estáticamente por el compilador
El número de instrucciones lanzadas es determinado dinámicamente por el hardware	El número de instrucciones lanzadas es determinado estáticamente por el compilador.
Lanzamiento de instrucciones dinámico permite ejecución en orden y fuera de orden	Calendarización estática solamente permite lanzamiento de instrucciones en orden.

Desde el punto de vista de microarquitectura, ambas pueden tener las mismas unidades funcionales y tipo de segmentación, pero Superescalar requiere más Hardware.

Superescalar vs. VLIW * [Fisher, J. A]

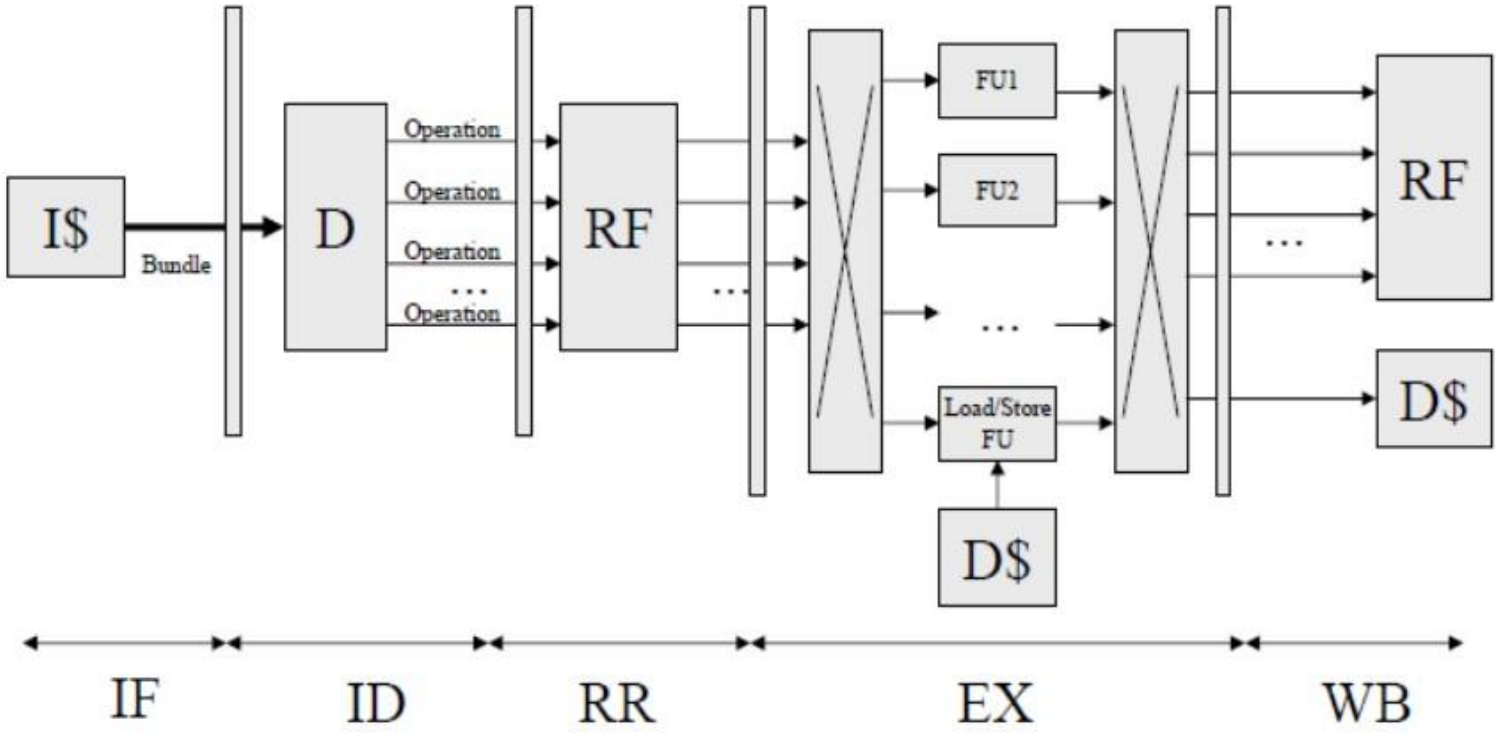


Arquitectura VLIW

VLIW

- La decodificación se reduce a lógica simple para cada operación.
- Red de despacho (control) direcciona operaciones a FU según opcodes (no determina orden, solo direcciona).
- Debe haber suficiente paralelismo en el código para mantener unidades funcionales y no tener desperdicio de HW. Implica en algunos casos ingresar ”NOPS” en FU que no se utilicen.

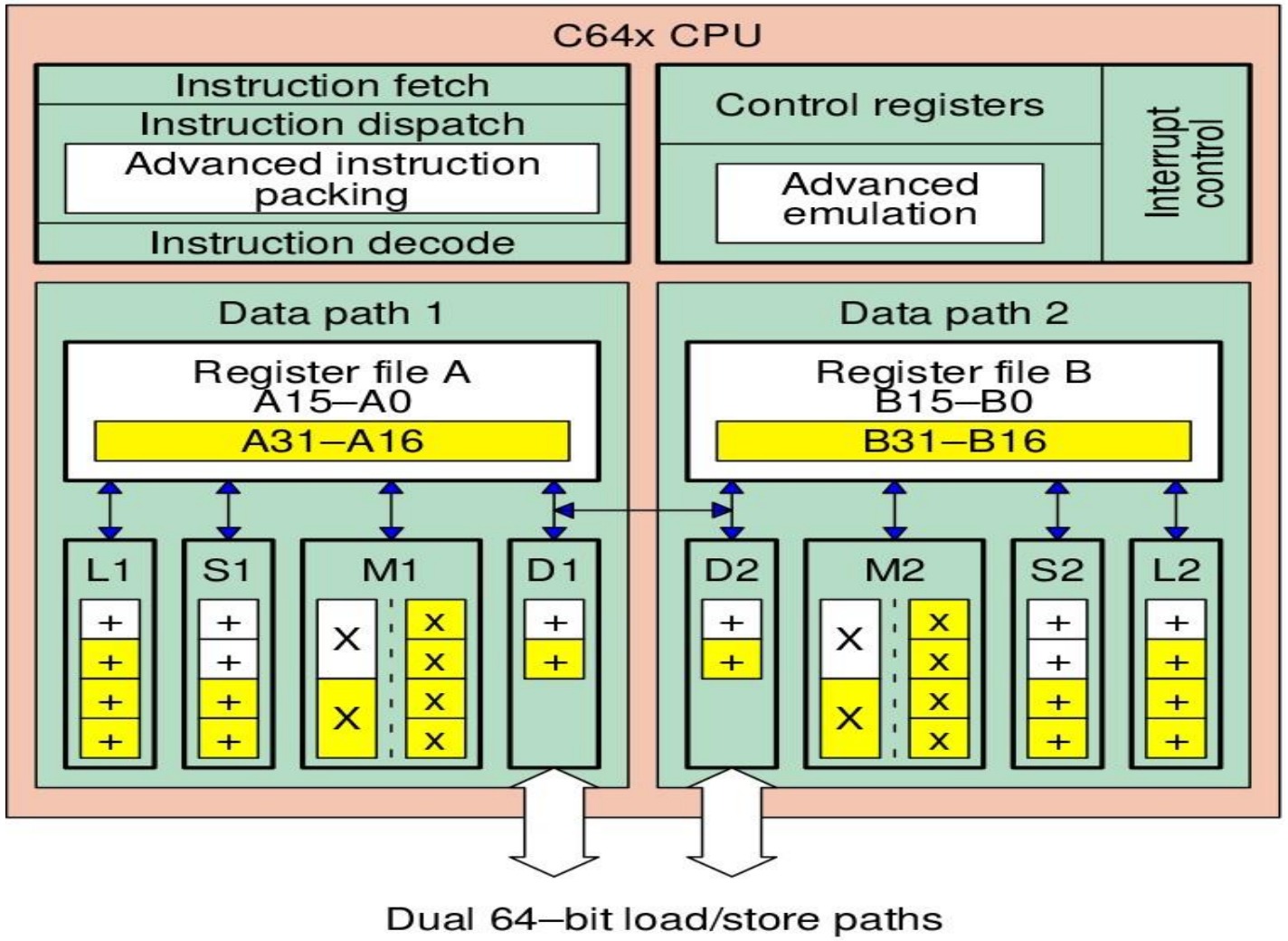
Arquitectura interna



Ejemplo DSP C64X TI Características

- **Predecesores** (1997) : C62x y C67x
- **Frecuencia:** Hasta 1.1 GHz
- **Throughput:** 8800+ MIPS
- **Arquitectura:** VLIW RISC like + pipeline = Ejecución en Paralelo
- **CPU:** VelociTI.2
  - 8 unidades funcionales
  - 2 archivos de registros
  - 2 datapaths
- **Principales Aplicaciones**
  - Comunicaciones Digitales
  - Procesamiento de imágenes

VelociTI.2


C64x CPU
Instruction fetch
Instruction dispatch
Advanced instruction packing
Instruction decode
Control registers
Advanced emulation
Interrupt control
Data path 1
Register file A
A15–A0
A31–A16
Data path 2
Register file B
B15–B0
B31–B16
L1
S1
M1
D1
D2
M2
S2
L2
Dual 64–bit load/store paths

C64X: Manejo de instrucciones

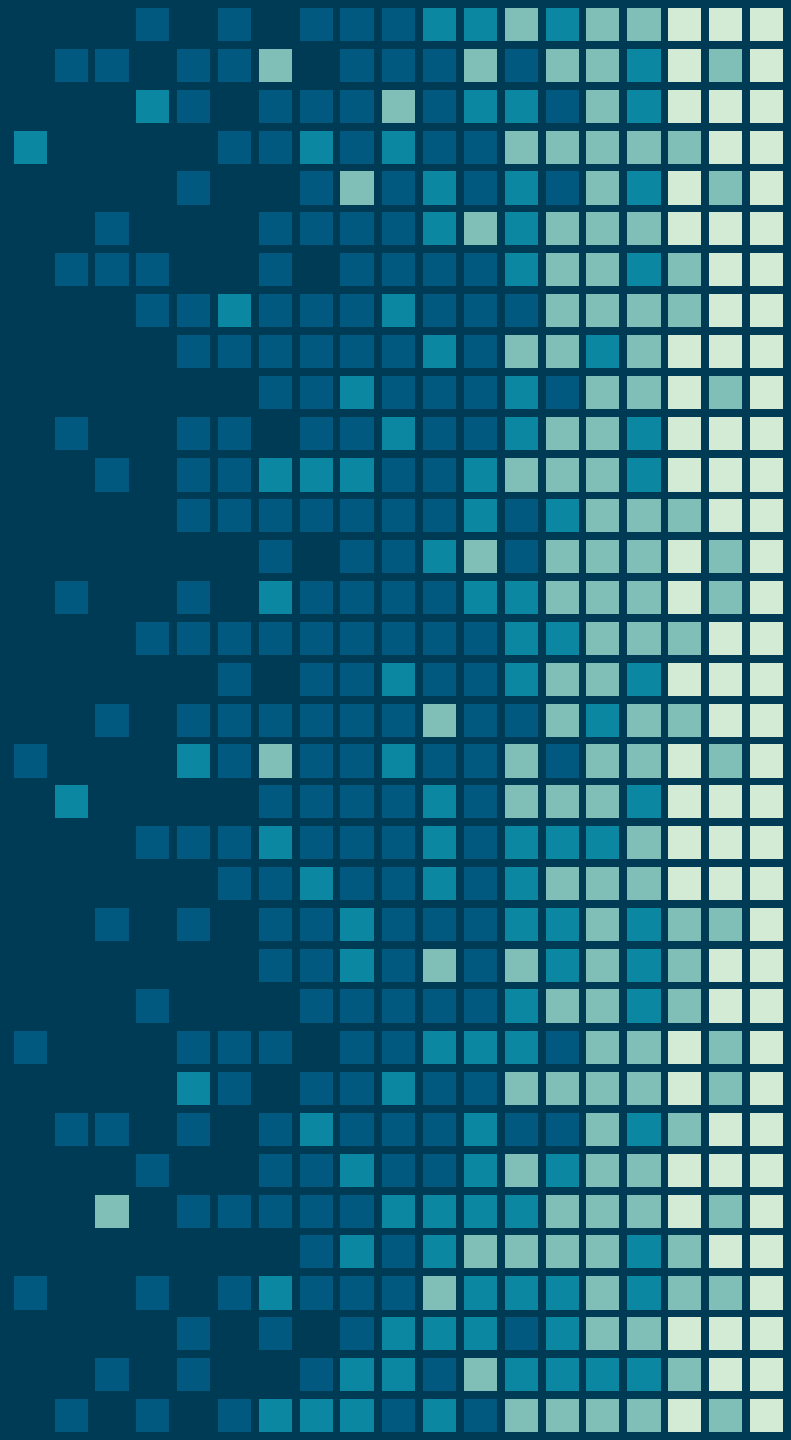
- Las instrucciones que son ejecutadas en paralelo son llamadas en paquetes de ejecución.
- El bus de instrucciones de 256 bits permite buscar 8 instrucciones de 32 bits simultáneamente, esto constituye el fetch package.
- VelociTI permite introducir más de un paquete de ejecución en el fetch package. Un fetch package contiene de 1 -8 paquetes de ejecución.
- Instrucciones se procesan en 3 etapas:
  - fetch
  - decode
  - execute

VLIW: código ejemplo

Memory reference 1	Memory reference 2	FP operation 1	FP op. 2	Int. op/ branch
LD F0,0(R1)	LD F6,-8(R1)	NOP	NOP	NOP
LD F10,-16(R1)	LD F14,-24(R1)	NOP	NOP	NOP
LD F18,-32(R1)	LD F22,-40(R1)	ADDD F4,F0,F2	ADDD F8,F6,F23	NOP
LD F26,-48(R1)	NOP	ADDD F12,F10,F2	ADDD F16,F14,F2	NOP
NOP	NOP	ADDD F20,F18,F2	ADDD F24,F22,F2	NOP
SD 0(R1),F4	SD-8(R1),F8	ADDD F28,F26,F2	NOP	NOP
SD-16(R1),F12	SD-24(R1),F16	NOP	NOP	NOP
SD-32(R1),F20	SD-40(R1),F24	NOP	NOP	SUBI R1,R1,#56
SD 8(R1),F28	NOP	NOP	NOP	BNEZ R1,LOOP

- Promedio 2.5 operaciones por ciclo
- Hay garantía de que cada operación en el paquete es independiente.

Dependencias



Latencia de operaciones

La latencia de cada operación es considerada en el código, por medio del compilador:

```
I     [C=A*B, . . .];
I+1   [NOP, . . . ]; insertado por el
      compilador
I+2   [X=C*F, . . .];
I+3   [NOP, . . .]; insertado por el
      compilador
```

Para una multiplicación con latencia 2, se deben generar NOPS en instrucción siguiente para evitar riesgos estructurales.

Dependencias

Dependencias son resueltas por el compilador y no por hardware.

Tomando en cuenta latencia de unidades funcionales.

En caso de riesgos **RAW:**

Escalar y superescalar generar nops/stalls, dinámicamente

En VLIW son detectados por el compilador. Se agregan NOPs estáticamente en el segmento correspondiente a esa unidad funcional.

Riesgos

WAR y WAW: El compilador determina el orden correcto de ejecución tal que no se presenten riesgos. Algunas arquitecturas pueden implementar renombramiento de registros por software.

Control:
El compilador provee información sobre cómo predecir saltos estáticamente (especulación, predicado).
Puede incluir soporte de HW para especulación dinámica.

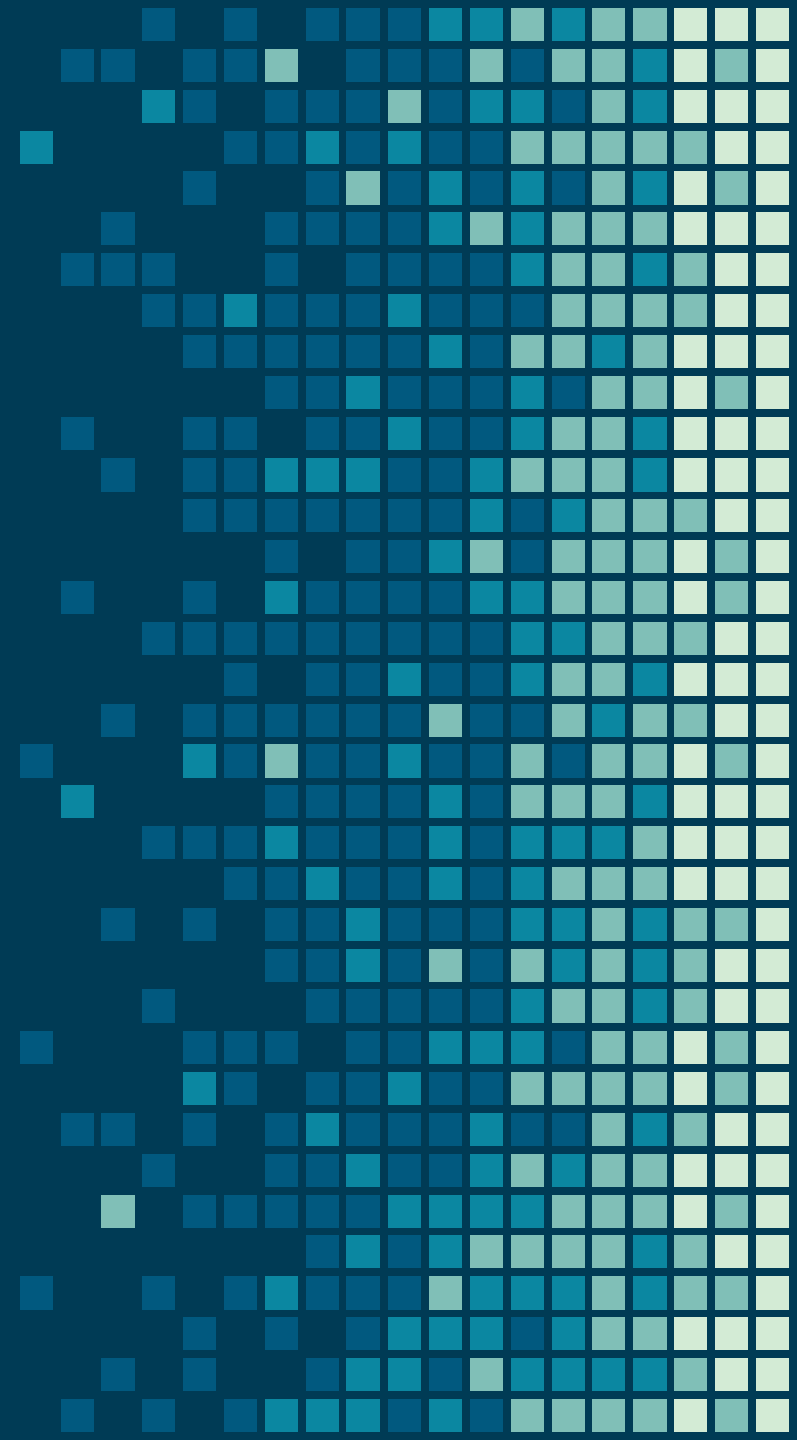
Ventajas

- El compilador puede utilizar algoritmos sofisticados para explotar paralelismo dentro del programa e incrementar el rendimiento.
- Instrucciones tienen campos definidos: fácil decodificación.
- La complejidad del Hardware se reduce lo que implica menor área, menor consumo de potencia, menor costo.

Desventajas

- Baja compatibilidad entre diferentes instancias de la misma arquitectura (diferente número y tipo de FU, latencias, ancho de instrucción)
- Añadir NOPS incrementa el tamaño del código significativamente.
- Compiladores especializados

Calendarización de código



Ejemplo MIPS vs VLIW

Calendarización MIPS

INSTRUCTIONS	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13
L1: lw$2,BASEA($4)	IF	ID	EX	M	WB								
addi $2,$2,INC1		X	IF	ID	EX	M	WB						
lw $3,BASEB($4)				IF	ID	EX	M	WB					
addi $3,$3,INC2					X	IF	ID	EX	M	WB			
addi $4, $4, 4							IF	ID	EX	M	WB		
bne $4,$7, L1								X	IF	ID	EX	M	WB

Calendarización VLIM

SLOT1: LD/ST Ops	SLOT2: LD/ST Ops	SLOT3: Integer Ops	SLOT4: Integer+Branch Ops
lw$2,BASEA($4)	lw $3,BASEB($4)	NOP	NOP
NOP	NOP	NOP	NOP
NOP	NOP	addi $2,$2,INC1	addi $3,$3,INC2
NOP	NOP	addi $4, $4, 4	NOP
NOP	NOP	NOP	NOP
NOP	NOP	NOP	bne $4,$7, L1

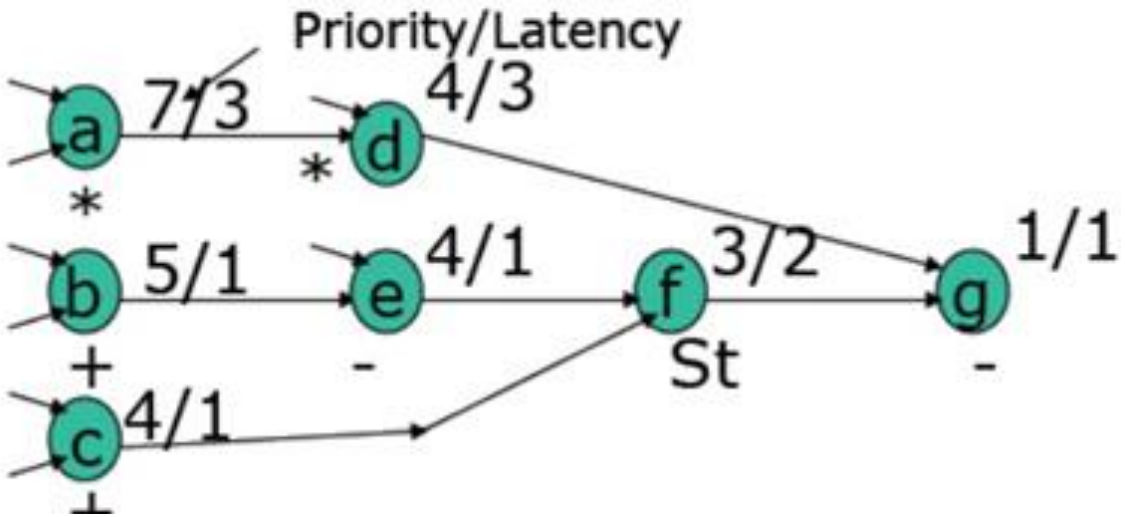
Calendarización de código

La calendarización selecciona el ciclo de ejecución de cada operación de forma que se ejecute en la menor cantidad de tiempo.

- Descomponer código en **bloques básicos**
- Armar grafo de dependencias
- Calcular ruta crítica
- Calendarizar tomando en cuenta recursos (FU)

Grafo de dependencias

Un grafo de dependencias captura dependencias verdaderas, anti dependencias y dependencias de salida entre las instrucciones de una manera ordenada dentro del programa.

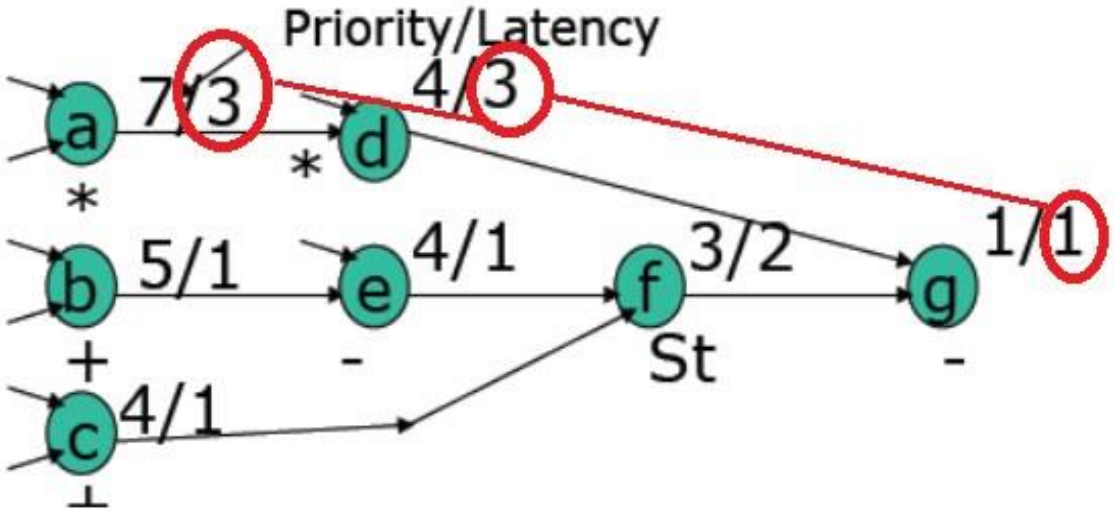


Ruta crítica

La ruta crítica en un grafo de dependencias es el camino más largo en términos de latencias por operación

- Determina el tiempo mínimo de ejecución de un bloque básico

En un procesador con recursos infinitos, se podría calendarizar todas las operaciones dentro de la ruta crítica primero, y después las siguientes explotando recursos disponibles



Ruta crítica

Con recursos finitos (caso real) el tiempo de ejecución no solo depende de la ruta crítica, sino de la forma que se calendaricen instrucciones restantes..

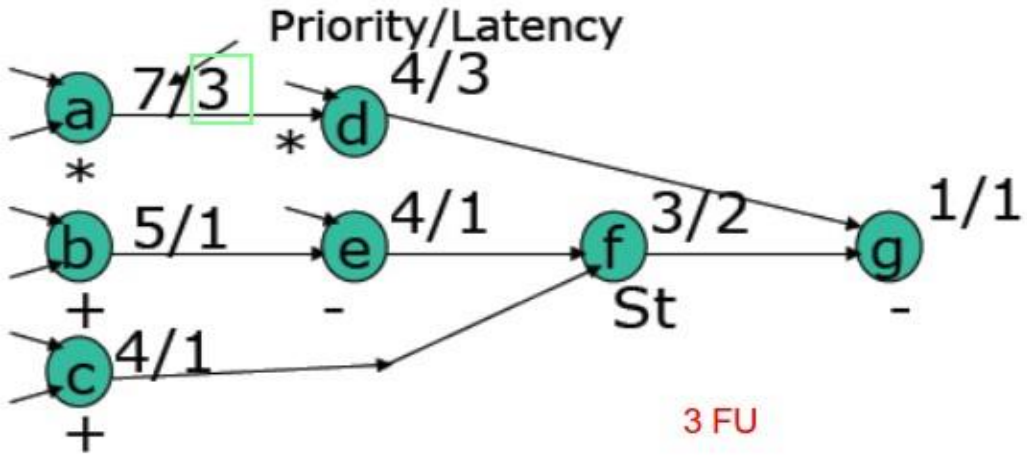
- ✓ Un compilador debe buscar exhaustivamente la calendarización óptima.
- ✓ Complejidad puede ser muy grande.

Calendarización de bloques básicos

Desde el primer ciclo, para cada ciclo:

- Intentar calendarizar operaciones en estado listo (todas instrucciones anteriores en el grafo han sido calendarizadas previamente)
- Cuando haya varias operaciones (nodos) en estado listo, se elige la que presenta más prioridad (ruta más larga hasta al final del grafo)
- Se utiliza una tabla de reserva de recursos. Si una FU está ocupada y es requerida por una instrucción, se intenta con otra en el conjunto de operaciones listas.

Calendarización estática: Ejemplo



VLIW CODE

NOP

3 FU

	Ready List
1	a,b,c
2	c,e
3	e
4	d,f
5	
6	
7	g

RRT		
ALU1	L/S	MUL
b		a
c		a
e		a
	f	d
	f	d
		d
g		

VLIW		
ALU	L/S	MUL
b	n	a
c	n	n
e	n	n
n	f	d
n	n	n
n	n	n
g	n	n

Referencias

Material basado en [Aguilar , 2014]

- C. Murillo, M. Aguilar (2014)
  Arquitectura VLIW.

- J Hennesy and David Patterson (2012)
  Computer Architecture: A Quantitative Approach. 5th Edition. Elsevier – Morgan Kaufmann.

- Fisher, J. A., Faraboschi, P., & Young, C. (2005).
  Embedded computing: a VLIW approach to architecture, compilers and tools. Elsevier.

- Documentación C64X
  http://www.ti.com/lsds/ti/dsp/c6000_dsp/c64x/products.page

¿Preguntas?

Realizado por: Jason Leitón Jiménez.

Tecnológico de Costa Rica

Ingeniería en Computadores

2024