6. Cache & Virtual Memory

Computer Architecture and Organizations

การออกแบบหน่วยความจำ

หนึ่งในข้อที่ยากต่อการออกแบบระบบคอมพิวเตอร์คือการจัดการกับหน่วยความจำ

หน่วยความจำของคอมพิวเตอร์มีความหลากหลายและแตกต่างกันมาก มีหลายชนิด หลายเทคโนโลยี มีความแตกต่างทางด้านประสิทธิภาพและราคา และไม่มีเทคโนโลยีใดที่เป็นคำตอบที่ดีที่สุดของการออกแบบหน่วยความจำ

การออกแบบหน่วยความจำที่ดีแบบหนึ่งคือการแบ่งหน่วยความจำของคอมพิวเตอร์ออกเป็นชั้นๆ (Hierarchy)

- หน่วยความจำภายใน (Internal) ที่ซีพียู สามารถเข้าถึงข้อมูลได้โดยตรง
- หน่วยความจำภายนอก (External) ที่ซีพียู สามารถเข้าถึงได้โดยผ่าน I/O module

จุดมุ่งหมายของการออกแบบหน่วยความจำคือ ออกแบบอย่างไรให้มีราคาถูกที่สุดและจะต้องมีความเร็วในระดับที่พอรับได้

ระดับชั้นของหน่วยความจำ

1. **Volatile cache memory** หน่วยความจำขนาดเล็ก มีความเร็วสูง ราคาแพงมาก และเป็นหน่วยความจำแบบลบเลือนได้
2. **RAM** หน่วยความจำขนาดกลาง มีความเร็วปานกลาง ราคาปานกลาง และเป็นหน่วยความจำแบบลบเลือนได้
3. **Nonvolatile memory** หน่วยความจำขนาดใหญ่ มีความเร็วต่ำ ราคาถูก และเป็นดิสก์เก็บข้อมูลที่เป็นแบบไม่ถูกลบเลือน

6.1 Cache

Cache

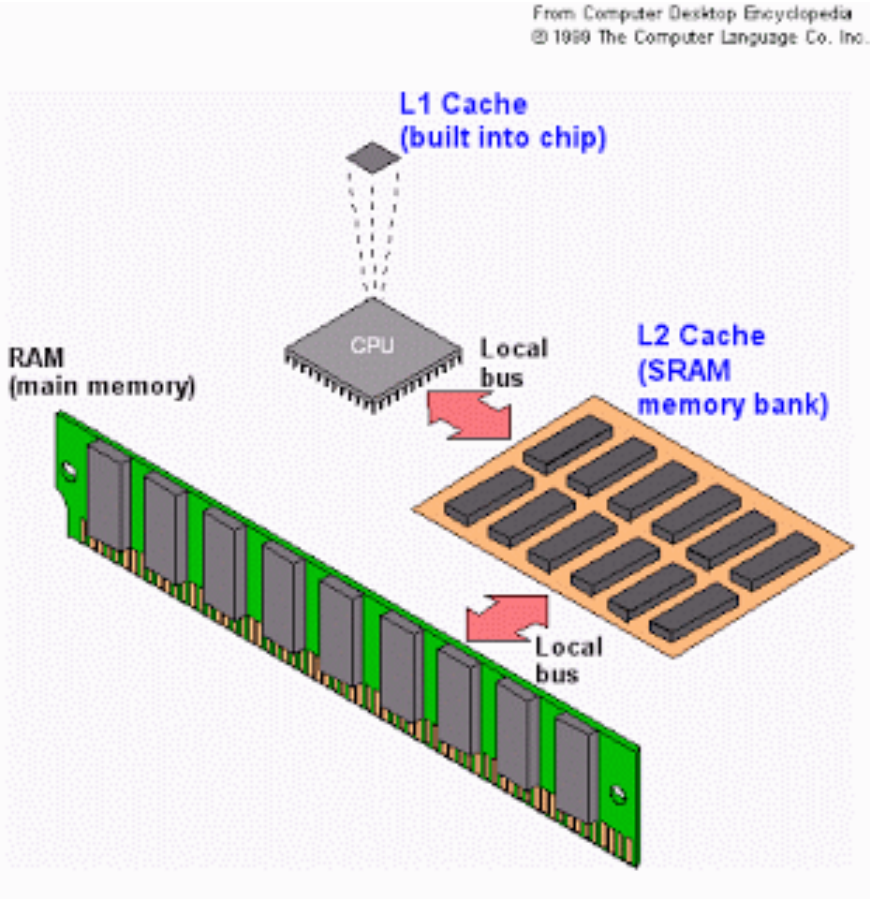
ในระบบคอมพิวเตอร์ถ้าส่วนของโปรแกรมหรือข้อมูล ที่กำลังทำงานหรือกำลังถูกเรียกใช้งานอยู่ในหน่วยความจำที่มีขนาดเล็ก และความเร็วสูง ค่าเฉลี่ยในการเข้าถึงข้อมูลก็จะลดลง

หน่วยความจำแคชจะถูกจัดวางไว้ระหว่างซีพียูและหน่วยความจำหลัก

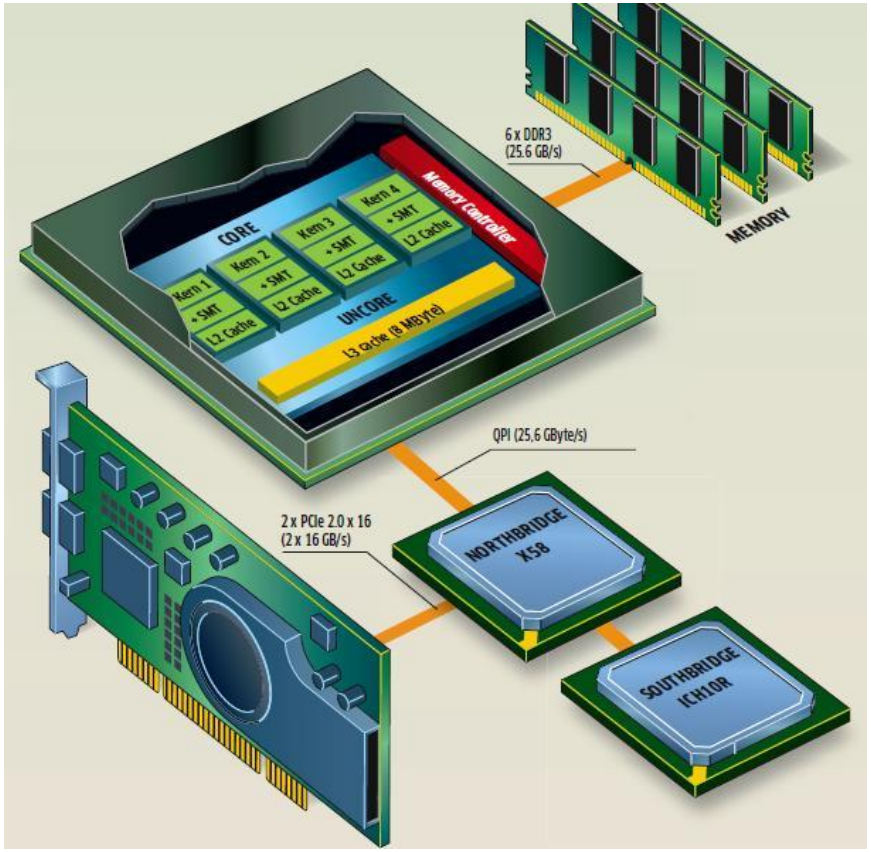
เวลาที่ใช้ในการเข้าถึงข้อมูลที่อยู่ในหน่วยความจำแคชจะมีความเร็วมากกว่าเวลาที่ใช้ในการเข้าถึงข้อมูลที่อยู่ในหน่วยความจำหลักประมาณ 5-10 เท่า

หน่วยความจำแคชเป็นส่วนหนึ่งในลำดับชั้นของหน่วยความจำที่มีความเร็วที่สุดและมีความเร็วที่ใกล้เคียงกับซีพียู

Classic cache



Modern cache



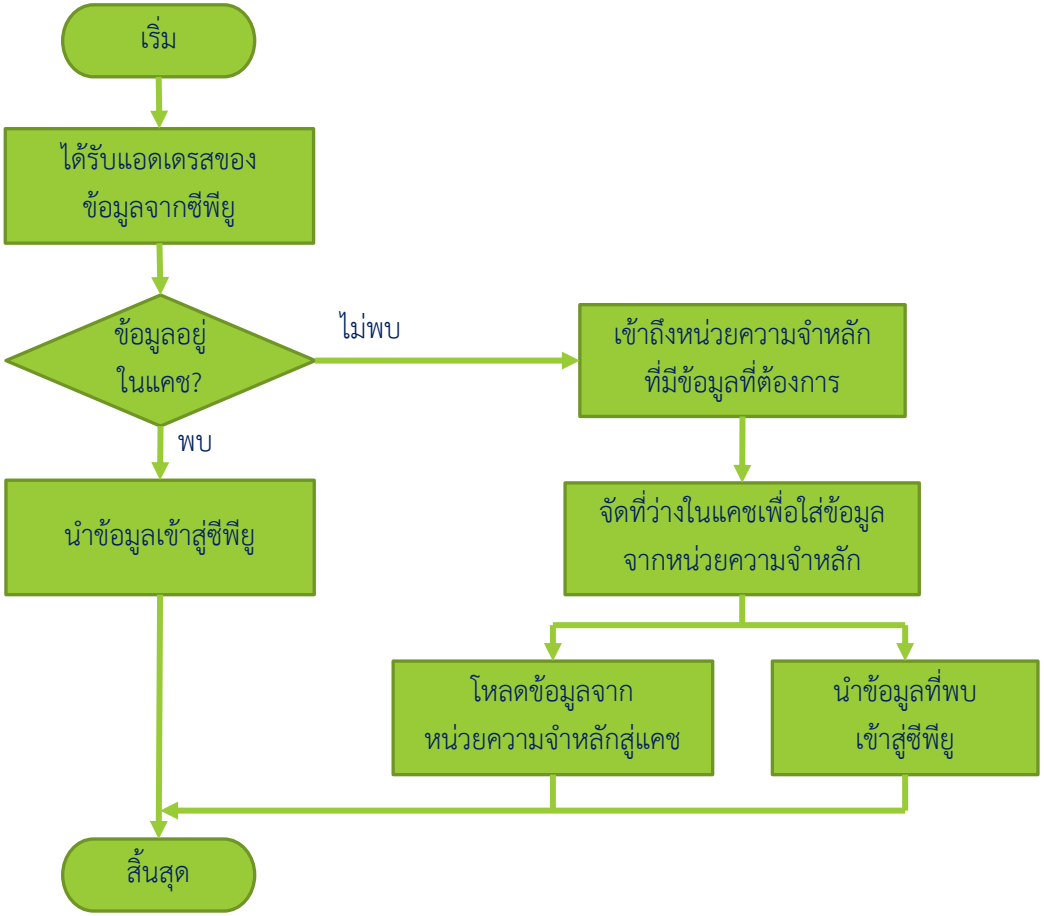
ความคิดขั้นพื้นฐานของการจัดการภายในหน่วยความจำแคช

คือการเก็บคำสั่งหรือข้อมูลที่ถูกเรียกใช้งานบ่อยๆ ลงในแคช

ทำให้ค่าเฉลี่ยของการเข้าถึงข้อมูลมีค่าใกล้เคียงกับเวลาในที่ใช้ในการเข้าถึงข้อมูลซีพียู

แม้ว่าแคชจะเป็นส่วนประกอบย่อยที่มีขนาดเล็กของหน่วยความจำในระบบคอมพิวเตอร์ แต่โปรแกรมหรือข้อมูลที่เรียกใช้งานส่วนใหญ่ก็ถูกเก็บไว้ในส่วนนี้

การทำงานขั้นพื้นฐานของหน่วยความจำแคช



6.2 การออกแบบหน่วยความจำแคช

ประสิทธิภาพการทำงานของแคช

วัดได้จาก อัตราการพบข้อมูล (Hit ratio)

เมื่อซีพียูต้องการนำคำสั่งหรือข้อมูลเข้า และพบว่าข้อมูลดังกล่าวถูกเก็บไว้ในแคชแล้ว เรียกว่า “พบ (Hit)”

ถ้าไม่พบข้อมูลที่ต้องการในแคช เนื่องจากข้อมูลดังกล่าวอยู่ในหน่วยความจำ เรียกว่า “พลาด (Miss)”

Hit ratio = Hit / (Hit + Miss)

Hit ratio >= 0.9 จะเป็นอัตราที่ยอมรับได้ ซึ่งเป็นการประเมินค่าของการออกแบบหน่วยความจำแคช

ค่าเฉลี่ยของเวลาที่ใช้ในการเข้าถึงข้อมูลของระบบคอมพิวเตอร์

สามารถทำให้เร็วขึ้นได้โดยการใช้หน่วยความจำแคช

ถ้าอัตราการพบข้อมูลในแคชมีค่าสูงพอจนกระทั่งเวลาในการเข้าถึงข้อมูลส่วนใหญ่ของซีพียูกระทำกับแคชเท่านั้น ค่าเฉลี่ยในการเข้าถึงข้อมูลก็จะมีค่าใกล้เคียงกับเวลาที่ใช้ในการเข้าถึงข้อมูลที่อยู่ในแคช

ถ้า

เวลาในการเข้าถึงข้อมูลในแคช = 100 ns

เวลาในการเข้าถึงข้อมูลในหน่วยความจำหลัก = 1000 ns

อัตราการพบข้อมูล = 0.9

ค่าเฉลี่ยเวลาในการเข้าถึงข้อมูลในแคช = 190 ns ซึ่งเร็วกว่าหน่วยความจำที่ต้องใช้เวลาถึง 1000 ns

6.3 การใช้ฟังก์ชันในการเชื่อมโยงข้อมูล

Mapping

Mapping function

- เป็นฟังก์ชั่นที่ใช้ในการเชื่อมโยงข้อมูล
- ทำหน้าที่จัดการการทำงานต่างๆ ระหว่างซีพียู แคช และหน่วยความจำหลัก
- ฟังก์ชั่นจะถูกจัดการหรือใช้ในระดับฮาร์ดแวร์ เพราะต้องการให้ฟังก์ชั่นนี้ทำงานที่ความเร็วสูง

คุณสมบัติของ Mapping function

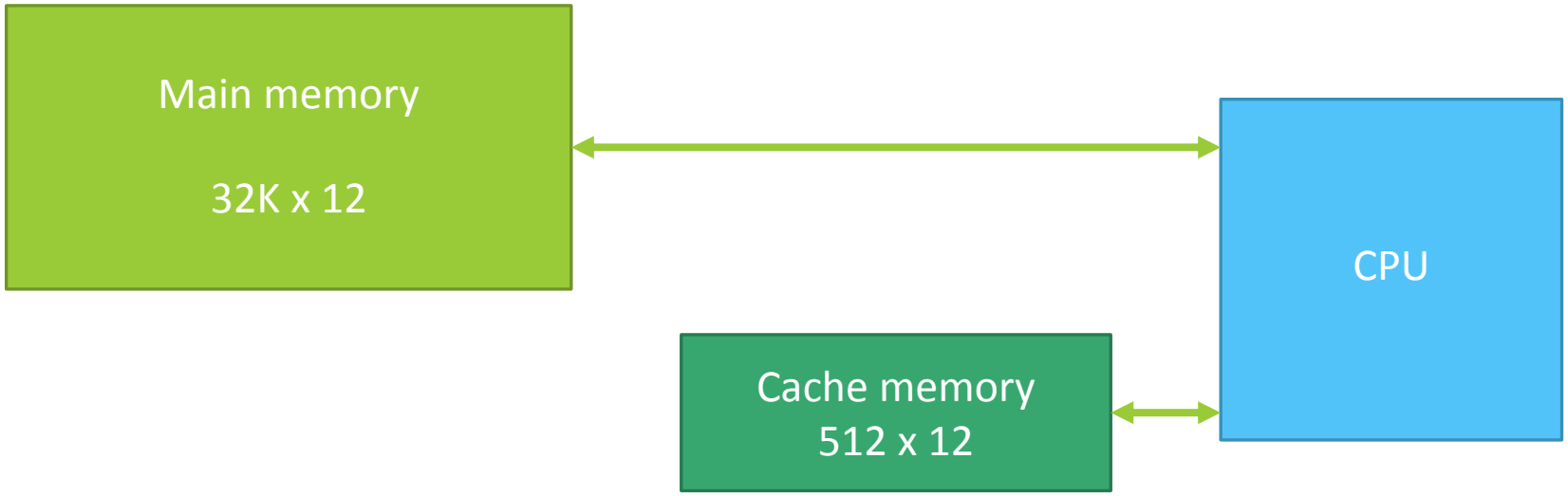
- **Placement** กลยุทธ์ในการวางข้อมูล – การวางบล็อกของข้อมูลลงในแคช
- **Replacement** กลยุทธ์ในการแทนที่ข้อมูล – บล็อกที่จะนำข้อมูลมาแทนที่เมื่อไม่มีการพบข้อมูลที่ต้องการในแคช

ต้องมีนโยบายในการอ่านและเขียนข้อมูล – วิธีการจัดการ การอ่านและการเขียนข้อมูลลงในแคช ที่ขึ้นอยู่กับการ พบและพลาด (Hit & Miss) ของข้อมูล

ชนิดของกระบวนการเชื่อมโยง (Mapping)

1. การเชื่อมโยงแบบสัมพันธ์ (Associative mapping)
2. การเชื่อมโยงแบบโดยตรง (Direct mapping)
3. การเชื่อมโยงแบบกลุ่มสัมพันธ์ (Block-set associative mapping)

การจัดวางหน่วยความจำแคช



การเชื่อมโยงแบบสัมพันธ์

Associative mapping

เป็นวิธีการที่เร็วและยืดหยุ่นที่สุดของการจัดการกับหน่วยความจำแคช

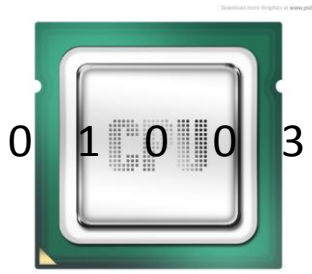
แคชจะเก็บทั้งแอดเดรสและข้อมูลในหน่วยความจำ ทำให้แคชสามารถเก็บข้อมูลจากเวิร์ดใดก็ได้จากหน่วยความจำหลัก

เมื่อโปรเซสเซอร์ต้องการข้อมูลก็จะทำการส่งแอดเดรสของข้อมูลที่ต้องการไปให้รีจิสเตอร์

ข้อมูลในแคชก็จะถูกสืบค้นเพื่อค้นหาแอดเดรสที่เท่ากับที่เก็บไว้ในรีจิสเตอร์

ถ้าพบข้อมูลขนาด 12 บิต ที่เก็บไว้ในแอดเดรสดังกล่าวก็จะถูกอ่านออกมา และส่งไปให้โปรเซสเซอร์

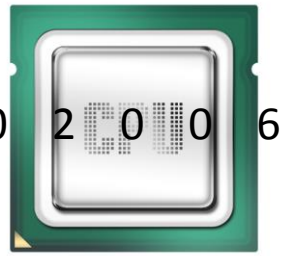
ถ้าไม่พบก็จะไปสืบค้นที่หน่วยความจำหลัก แอดเดรสและข้อมูลที่พบในหน่วยความจำหลักก็จะถูกคัดลอกเข้ามาไว้ในแคช



	Argument register

Cache

Address (Oct.)	Data (Oct.)
0 1 0 0 0	2 1 5 0
0 1 0 0 1	2 3 0 2
0 1 0 0 2	3 4 5 6
0 1 0 0 3	**3 4 5 0**
0 1 0 0 4	7 0 7 0
0 1 0 0 5	8 4 2 3



Argument register

Cache

Address (Oct.)	Data (Oct.)
0 1 0 0 0	2 1 5 0
0 1 0 0 1	2 3 0 2
0 1 0 0 2	3 4 5 6
0 1 0 0 3	3 4 5 0
0 1 0 0 4	7 0 7 0

Main memory

Address (Oct.)	Data (Oct.)
0 2 0 0 6	**1 4 7 0**
0 2 0 0 7	2 3 3 5
0 2 0 1 0	6 2 5 4
0 2 0 1 1	
0 2 0 1 2	
0 2 0 1 3	

กรณีที่แคชเต็ม

ข้อมูลและแอดเดรสจะเข้าไปแทนที่ข้อมูลที่ไม่ต้องการแล้ว

การตัดสินใจว่าข้อมูลคู่ใดที่จะถูกสับเปลี่ยนออกไปนั้น ขึ้นอยู่กับ อัลกอริทึ่มของผู้ที่ออกแบบระบบ

วิธีการสับเปลี่ยนที่ง่ายวิธีหนึ่งก็คือการสับเปลี่ยนแบบ มาก่อน-ออกก่อน (First-in First-out: FIFO)

ข้อดีของการเชื่อมโยงแบบสัมพันธ์

- มีความยืดหยุ่นสูง
- ใช้ความจุของแคชอย่างเต็มที่
- มีการเก็บบล็อกของข้อมูลที่ใดก็ได้ที่ต้องการ

ข้อเสียของการเชื่อมโยงแบบสัมพันธ์

- เมื่อมีการอ่านข้อมูลของแคชในแต่ละครั้ง ฟังก์ชั่นนี้จะต้องทำการค้นหาแอดเดรสทั้งหมดภายในแคชให้ตรงกับแอดเดรสที่อ้างอิง
- การอ่านหรือค้นหาแอดเดรสแบบลำดับ (Sequential) เป็นสิ่งที่ยอมรับไม่ได้เพราะเวลาที่ใช้ในการเข้าถึงข้อมูลจะช้ามาก
- ข้อมูลแอดเดรสที่เก็บในแคชควรถูกแยกเก็บไว้ในหน่วยความจำที่เก็บแอดเดรสโดยเฉพาะ (Content-addressable memory) จะทำให้การค้นหาข้อมูลแบบสัมพันธ์เป็นไปในลักษณะคู่ขนานได้
- หน่วยความจำที่มีฟังก์ชั่นการทำงานแบบนี้จะมีราคาแพงกว่าแบบอื่นๆ ในเรื่องของจำนวนเกตที่ใช้ เพราะต้องมีการเปรียบเทียบข้อมูลแบบบิตต่อบิต พร้อมๆ กันทุกบิตในหน่วยความจำโดยใช้เกต XOR การใช้งานจึงจำกัดอยู่แค่เพียงระบบที่มีขนาดเล็ก

การเชื่อมโยงแบบโดยตรง

Direct mapping

การเชื่อมโยงแบบสัมพันธ์กับหน่วยความจำจะมีราคาที่สูงเมื่อเทียบกับการเชื่อมโยงแบบสุ่ม (Random access)

เนื่องจากระบบต้องเพิ่มโปรแกรมในการสร้างความสัมพันธ์ให้กับการสร้างความสัมพันธ์ทุกบล็อกข้อมูลในแคช

การใช้หน่วยความจำแคชแบบโดยตรง จะแบ่งแอดเดรสของโปรเซสเซอร์จากทั้งหมด 15 บิต ออกเป็น 2 ส่วน

- 9 บิต หลังจะเป็นฟิลด์อินเด็กซ์ (Index)
- 6 บิต ที่เหลือจะใช้เป็นแท็ก (Tag) ที่ใช้ในการเปรียบเทียบ

ถ้ามีหน่วยความจำทั้งหมด 2^k เวิร์ด ในแคช และมีหน่วยความจำทั้งหมด 2^n เวิร์ด ในหน่วยความจำหลัก

จำนวน k บิตแรกเป็นอินเด็กซ์ และ n-k ที่เหลือจะใช้เป็นฟิลด์แท็ก

การเชื่อมโยงแคชแบบโดยตรง

เมื่อมีข้อมูลเข้ามาใหม่ครั้งแรก บิตของแท็กจะถูกบันทึกคู่กับข้อมูลจริง

เมื่อซีพียูต้องการเรียกใช้ข้อมูล ฟิลด์ที่เป็นอินเด็กซ์ก็จะถูกนำมาใช้เป็นแอดเดรสสำหรับการเข้าถึงแคช

ฟิลด์แท็กของแอดเดรสที่ซีพียูต้องการก็จะถูกเปรียบเทียบกับข้อมูลแท็กที่อยู่ในแคช

- ถ้าข้อมูลของแท็กตรงกันข้อมูลที่ต้องการก็จะถูกพบในแคช
- ถ้าข้อมูลของแท็กไม่ตรงกันก็จะไม่พบข้อมูลในแคช และจะต้องไปหาข้อมูลที่ต้องการในหน่วยความจำหลักต่อไป

หลังจากนั้นข้อมูลดังกล่าวก็จะถูกบันทึกทับไว้ในแคชที่ตำแหน่งเดิมพร้อมกับแท็กใหม่

ข้อเสียคือถ้าเกิดบล็อกข้อมูลที่ต้องการมีแอดเดรสของอินเด็กซ์เหมือนกัน แต่แท็กไม่เหมือนกัน จะถูกเรียกใช้งานซ้ำๆ กันเป็นจำนวนมาก

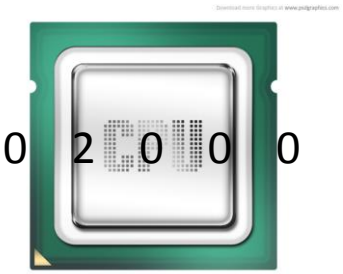
Main memory

Memory address	Data
0 0 0 0 0	1 2 2 0
0 0 7 7 7	2 3 4 0
0 1 0 0 0	3 4 5 0
0 1 7 7 7	4 5 6 0
0 2 0 0 0	5 6 7 0
0 2 7 7 7	6 7 1 0

Cache memory

Index address	Tag	Data
0 0 0	0 2	1 2 2 0
7 7 7	0 2	6 7 1 0

Tag	Index



การเชื่อมโยงแคชแบบโดยตรงสำหรับบล็อกข้อมูล 8 เวิร์ด

ฟิลด์ที่ใช้เป็นอินเด็กซ์จะถูกแบ่งออกเป็น 2 ส่วนคือ ฟิลด์ที่แสดงตำแหน่งบล็อกข้อมูลของ 8 เวิร์ด ทั้งหมด 64 บล็อก (64 x 8 = 512)

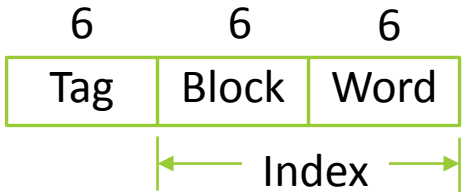
เลขบล็อกข้อมูลจะถูกกำหนดจากฟิลด์บล็อก 6 บิต และลำดับของเวิร์ดจะถูกกำหนดโดยใช้ข้อมูล 3 บิต ส่วนแท็กฟิลด์ใน 8 เวิร์ด แอดเดรสในบล็อกเดียวกันจะมีค่าเท่ากัน

ทุกครั้งที่เกิดการพลาดของข้อมูล หรือไม่พบข้อมูลในแคช ข้อมูลทั้งบล็อกทั้ง 8 เวิร์ด จะต้องถูกเก็บเข้ามาจากหน่วยความจำหลักและบันทึกไว้ในแคช

ถึงแม้ว่าการนำข้อมูลทั้งหมดเข้ามาจะเป็นการเพิ่มเวลาบ้าง แต่อัตราการพบข้อมูลจะเพิ่มมากขึ้นเมื่อขนาดของบล็อกข้อมูลใหญ่ขึ้น เนื่องจากการเขียนข้อมูลส่วนใหญ่นั้นจะเป็นการเขียนให้มีการทำงานแบบลำดับ ซึ่งทำให้เกิดการเก็บข้อมูลเป็นบล็อก

การเชื่อมโยงแคชแบบโดยตรงสำหรับบล็อกข้อมูล 8 เวิร์ด

	Index	Tag	Data
Block 0	0 0 0 0 0 7	0 1 0 1	3 4 5 0 3 6 5 7 8
Block 1	0 0 0 0 0 7	0 1 0 1	3 4 5 0 3 6 5 7 8
Block 63	0 0 0 0 0 7	0 1 0 1	3 4 5 0 3 6 5 7 8



การเชื่อมโยงแคชแบบโดยตรง

มีข้อดีตรงที่ความง่ายของระบบ แต่ก็มีข้อเสียตรงที่การนำเสนอข้อมูลลงแคชในแต่ละครั้งนั้นสามารถทำได้เพียงบล็อกข้อมูลเดียวจากในกลุ่มใดกลุ่มหนึ่ง

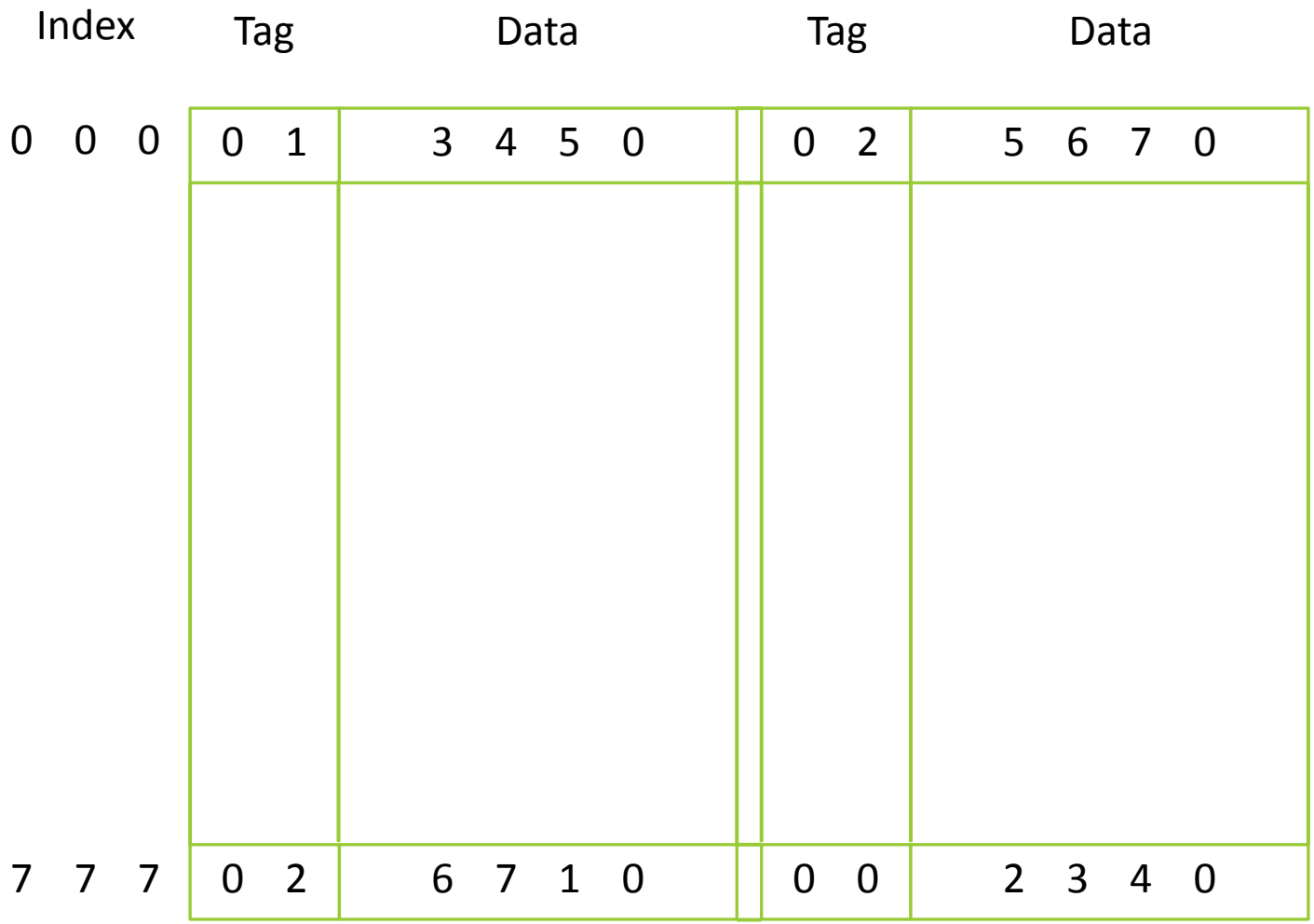
เช่น ถ้ามีบล็อกข้อมูล 2 บล็อกที่อยู่ในกลุ่มเดียวกันและถูกเรียกใช้งานใหม่ (บางทีอาจจะอยู่ภายในลูปของโปรแกรมเดียวกัน) ก็จะทำให้มีการย้ายบล็อกของข้อมูลทั้งสองบล็อกนี้ เข้าและออกจากแคชซ้ำๆ กัน

การปรับปรุงแคชที่มีการเชื่อมโยงแบบโดยตรงที่สำคัญคือ การยอมหรืออนุญาตให้มีการเก็บข้อมูลในแคชมากกว่า 1 บล็อกข้อมูลในกลุ่มใดๆ และการค้นหาบล็อกของข้อมูลภายในกลุ่มสามารถทำได้โดยใช้วิธีแบบสัมพันธ์อีกที (Block set associative)

การเชื่อมโยงแบบกลุ่มสัมพันธ์

Block-set associative mapping

- แต่ละเวิร์ดที่อยู่ในแคชสามารถบันทึกข้อมูลที่มีฟิลด์ของอินเด็กซ์ที่เหมือนกันได้มากกว่า 1 เวิร์ดขึ้นไป
- ข้อมูลจริงของแต่ละเวิร์ดจะถูกบันทึกพร้อมกับแท็กของมัน และจำนวนของแท็กพร้อมข้อมูลในหนึ่งเวิร์ดจะถูกกำหนดเป็นกลุ่ม
- ฟิลด์อินเด็กซ์สามารถกำหนดข้อมูลได้ 2 เวิร์ดพร้อมแท็ก ทำให้ต้องการแท็ก 6 บิต และเวิร์ดของข้อมูล 12 บิต ทั้งหมด 2 ชุด ทำให้ความยาวของเวิร์ดมีค่าเท่ากับ 2(6+12) = 36 บิต


Index
Tag
Data
Tag
Data
0 0 0
0 1
3 4 5 0
0 2
5 6 7 0
7 7 7
0 2
6 7 1 0
0 0
2 3 4 0

การเชื่อมโยงแบบกลุ่มสัมพันธ์

อัตราการพบข้อมูลจะดีขึ้น เมื่อขนาดของกลุ่มใหญ่ขึ้น เพราะจะทำให้มีจำนวนเวิร์ดที่มีฟิลด์ของอินเด็กซ์เหมือนกันแต่แท็กที่แตกต่างกันมากขึ้น

อย่างไรก็ตามการเพิ่มขนาดของกลุ่มนั้นก็จะทำให้ขนาดของแคชเพิ่มมากขึ้นด้วย ซึ่งเป็นผลไม่ดีมากนักสำหรับค่าใช้จ่ายที่เพิ่มขึ้น และยังทำให้ระบบต้องการโปรแกรมที่ใช้ในการเปรียบเทียบแท็กที่ซับซ้อนมากขึ้นด้วย

6.4 การจัดการด้านอื่นๆ ของแคช

การสับเปลี่ยนข้อมูล

Replacement

การเชื่อมโยงแบบโดยตรง

- บล็อกของข้อมูลจะมีเพียงตัวเลือกเดียวเท่านั้น คือบล็อกที่อยู่ในตำแหน่งของฟิลด์อินเด็กซ์ที่ต้องการนั่นเอง

การเชื่อมโยงแบบสัมพันธ์และกลุ่มสัมพันธ์

- อัลกอริทึ่มการสับเปลี่ยนข้อมูลยังเป็นสิ่งที่ต้องการและน่าสนใจมาก ในการที่เราจะได้แคชที่มีความเร็วสูง
- วิธีการสับเปลี่ยนจะต้องกระทำภายในฮาร์ดแวร์ด้วย

อัลกอริทึ่มการสับเปลี่ยนข้อมูล

วิธีสับเปลี่ยนแบบใช้น้อยที่สุดออกไปก่อน (LRU)

- คือการสับเปลี่ยนเอาบล็อกข้อมูลที่อยู่ในแคชนานที่สุดและไม่ได้ถูกเรียกใช้ในช่วงเวลาที่ผ่านมาเลยออกไปก่อน
- โดยให้แต่ละเวิร์ดมีบิตพิเศษที่เรียกว่า USE bit เมื่อมีการเรียกใช้เวิร์ดในแคชแถวใดก็เปลี่ยนบิตให้เป็น 1 และกำหนดบิตที่เหลือของเวิร์ดอื่นๆ ในแถวนั้นเป็น 0

วิธีสับเปลี่ยนแบบเข้ามาก่อนออกก่อน (FIFO)

- สับเปลี่ยนบล็อกของข้อมูลที่อยู่ในแคชนานที่สุดออกไป
- ใช้เทคนิคของบัฟเฟอร์ที่เป็นวงรอบ (Round-robin)

อัลกอริทึ่มการสับเปลี่ยนข้อมูล

วิธีสับเปลี่ยนข้อมูลที่ใช้จำนวนครั้งน้อยที่สุดออกไปก่อน (Least Frequency Used : LFU)

- สามารถทำได้โดยใช้ตัวนับสำหรับเวิร์ดแต่ละตัว

วิธีการสับเปลี่ยนแบบสุ่ม

- ไม่ขึ้นอยู่กับบันทึกของการใช้งานบล็อกข้อมูลว่ามากน้อยเพียงใด
- ไม่ได้ช่วยให้แคชทำงานมีประสิทธิภาพเพิ่มขึ้นมากนัก เมื่อเปรียบเทียบกับการนำบันทึกของการใช้งานเป็นปัจจัยในการคัดเลือกบล็อกของข้อมูล

การเขียนข้อมูล

Write policy

การเขียนต้องถูกกระทำในเวิร์ดที่อยู่ในแคช และทำการเปลี่ยนค่าแอดเดรสในหน่วยความจำหลักด้วย

ยังมีปัจจัยอื่นๆ ที่ทำให้หน่วยความจำหลักมีการเปลี่ยนแปลงข้อมูล และแคชก็จะต้องทำการอัปเดทค่าของข้อมูลด้วย

เนื่องจากมีอุปกรณ์ที่มากกว่า 1 ที่สามารถเข้าถึงหน่วยความจำได้ เช่น I/O Module ที่สามารถอ่าน/เขียนข้อมูลในหน่วยความจำหลักโดยตรง ถ้าอุปกรณ์ทำการเปลี่ยนข้อมูลในหน่วยความจำหลัก โดยไม่มีการอัพเดทข้อมูลในแคช ข้อมูลก็จะไม่ถูกต้อง

ระบบที่มีซีพียูมากกว่า 1 และในแต่ละตัวมีแคชแยกเป็นของตัวเอง เมื่อมีการเปลี่ยนแปลงข้อมูลหนึ่งในแคช อาจจะทำให้ข้อมูลในแคชอื่นๆ ผิดพลาดไปด้วย

การเขียนทั้งหมด

Write through

เป็นวิธีที่ง่ายที่สุดและใช้งานมากที่สุด คือให้มีการปรับปรุงข้อมูลในหน่วยความจำหลักทุกครั้งที่มีการเขียนข้อมูลเกิดขึ้น พร้อมกับอัพเดทข้อมูลที่อยู่ในแคชที่มีแอดเดรสตรงกัน

ข้อดีคือข้อมูลที่อยู่ในหน่วยความจำหลักนั้นจะมีข้อมูลที่ตรงกับข้อมูลที่อยู่ในแคช

การทำงานแบบนี้เป็นสิ่งที่สำคัญอย่างหนึ่งในระบบที่มีการเข้าถึงหน่วยความจำแบบโดยตรง ซึ่งจะทำให้ข้อมูลที่อยู่ในหน่วยความจำหลักถูกต้องและพร้อมใช้งานได้ตลอดเวลา

การเขียนทีหลัง

Write back

การปรับเปลี่ยนข้อมูลจะกระทำในแคชเท่านั้น แต่ตำแหน่งของข้อมูลดังกล่าวก็จะถูกบันทึกโดยอาจจะใช้บิตพิเศษที่เรียกว่า flag ทำการกำหนดว่าเมื่อนำเวิร์ดนั้นออกจากข้อมูลเมื่อใด ก็จะต้องทำการอัพเดทข้อมูลที่อยู่ในหน่วยความจำด้วย

เหตุผลคือตลอดเวลาที่บล็อกของข้อมูลอยู่ในแคช ข้อมูลอาจจะถูกแก้ไขเปลี่ยนแปลงได้หลายๆ ครั้ง โดยที่ไม่ต้องกังวลว่าข้อมูลในหน่วยความจำจะไม่ถูกต้องอย่างไร เพราะการเรียกใช้ข้อมูลในตำแหน่งนั้นก็ถูกเรียกจากแคชอย่างเดียวอยู่แล้ว

เป็นการประหยัดเวลาในการปรับปรุงข้อมูล แต่มีข้อเสียคือการเข้าถึงข้อมูลจากอุปกรณ์ I/O จะต้องกระทำผ่านแคชเท่านั้น ซึ่งจะทำให้แผงวงจรเกิดการซับซ้อนมากขึ้นและเกิดความคับคั่งของข้อมูลได้

จำนวนของแคช

หน่วยความจำแบบแคชที่นำมาใช้ครั้งแรกนั้นในระบบจะมีแคชเพียง 1 ตัวเท่านั้น

ต่อมาการใช้แคชหลายๆ ตัวได้กลายมาเป็นบรรทัดฐานมากขึ้น

การออกแบบระบบที่มีแคชมากกว่า 1 ตัวนั้น มีสิ่งที่จะต้องคำนึงถึง 2 อย่าง

- จำนวนลำดับชั้นของแคช
- การใช้แคชแบบเก็บข้อมูลรวมหรือแบบแยก

แคชที่มีหลายลำดับชั้น

เนื่องจากการออกแบบแผงวงจรได้ถูกพัฒนาขึ้นเข้าสู่ขั้น VLSI ทำให้สามารถสร้างแคชให้อยู่บนชิพของซีพียูได้ (on-chip cache)

เมื่อเราทำการเปรียบเทียบแคชที่มีการเข้าถึงโดยการใช้บัสพบว่า แคชที่อยู่บนชิพนั้นสามารถลดเวลาในการเข้าถึงข้อมูลที่ผ่านบัสได้อย่างมาก ทำให้การเรียกใช้คำสั่งมีความรวดเร็ว และการทำงานโดยรวมของระบบมีประสิทธิภาพเพิ่มขึ้น

เมื่อพบคำสั่งหรือข้อมูลที่อยู่ในแคชออนชิพ ก็ไม่มีความจำเป็นที่จะต้องเข้าถึงข้อมูลผ่านบัส เพราะว่าเส้นทางของข้อมูลภายในซีพียูย่อมสั้นและเร็วกว่าเส้นทางที่ผ่านบัส

นอกจากนั้นเส้นทางข้อมูลของบัสก็จะว่างและสามารถนำไปใช้ส่งผ่านข้อมูลอื่นๆ ได้อีก

เหตุผลที่ต้องมีแคชระดับที่สอง

- ถ้าไม่มีแคชภายนอก (L2) และซีพียูเรียกใช้ข้อมูลที่ไม่มีอยู่ใน L1 ซีพียูก็ต้องเข้าถึงข้อมูลที่อยู่ในหน่วยความจำหลัก เช่น DRAM ผ่านทางบัส
- ความเร็วของบัสช้าและการเข้าถึงหน่วยความจำหลักช้าก็จะทำให้ประสิทธิภาพการทำงานของระบบต่ำลง
- ถ้านำแคชภายนอก เช่น L2 SRAM มาใช้ ข้อมูลที่ไม่มีใน L1 สามารถได้รับจาก L2 และถ้า SRAM มีความเร็วพอๆ กับความเร็วของบัส ก็จะทำให้ซีพียูทำงานได้ทันทีโดยไม่ต้องรอ

ข้อควรคำนึงถึงในการออกแบบแคชที่มีหลายลำดับชั้น

การออกแบบแคช L2 ส่วนใหญ่จะไม่ใช้เส้นทางบัสของระบบเชื่อมต่อระหว่าง L2 กับซีพียู แต่จะใช้เส้นทางของข้อมูลแยกต่างหาก เพื่อเป็นการลดการคับคั่งของข้อมูลที่มีอยู่ในบัสของระบบ

เนื่องจากเทคโนโลยีนาโนเทคโนโลยีก้าวหน้าขึ้น เราสามารถออกแบบให้แคช L2 มาวางอยู่บนชิพได้เช่นเดียวกัน ซึ่งจะทำให้ประสิทธิภาพการทำงานของระบบเร็วยิ่งขึ้นไปอีก

อย่างไรก็ตามการออกแบบแคชแบบลำดับค่อนข้างที่จะซับซ้อนมาก และการออกแบบจะต้องมีความสัมพันธ์กันทั้งในเรื่องของขนาดของแคชในแต่ละชั้น วิธีการสับเปลี่ยน และวิธีที่ใช้ในการเขียนข้อมูล

การใช้งานแคชแบบเก็บข้อมูลรวม

โดยปกติแล้วแคชที่เก็บข้อมูลรวมนั้น จะมีอัตราการพบข้อมูลมากกว่าแคชที่เก็บข้อมูลแบบแยกประเภท

การใช้งานจะสมดุลระหว่างคำสั่งและข้อมูลที่ถูกเรียกใช้โดยอัตโนมัติ ถ้าโปรแกรมทำการเรียกใช้คำสั่งมากกว่าการเรียกใช้ข้อมูล แคชก็จะเก็บคำสั่งไว้ในแคชทั้งสอง ทำให้อัตราการพบข้อมูลเพิ่มมากขึ้น

การออกแบบแคชแบบรวมนั้นง่ายทั้งการออกแบบและการนำไปใช้

การใช้งานแคชแบบเก็บข้อมูลแยก

เทคโนโลยีจะมุ่งไปหาการออกแบบแคชแบ่งแยกประเภทของข้อมูลมากกว่าแบบรวม ในระบบที่เน้นการเรียกใช้คำสั่งแบบขนานและการดึงคำสั่งที่คาดหมายไว้ล่วงหน้ามาใช้งาน

มีข้อดีตรงที่เราสามารถลดการแข่งขันของแคชในการเรียกนำข้อมูลเข้า/แปลข้อมูลกับการเรียกใช้คำสั่งได้ ซึ่งเป็นหัวใจของการออกแบบของการเรียกใช้คำสั่งแบบขนานที่เรียกว่า Pipeline นั่นเอง

การเริ่มต้นใช้งานของแคช

- เริ่มต้นเมื่อมีกระแสไฟฟ้าเข้าไปเลี้ยงระบบคอมพิวเตอร์ หรือเมื่อหน่วยความจำถูกโหลดโปรแกรมเริ่มต้นมาจากหน่วยความจำอื่นๆ เช่น ROM
- หลังจากระบบเริ่มทำงาน แคชจะถูกกำหนดให้ว่างเปล่า แต่ในความเป็นจริงแล้วแคชจะมีข้อมูลที่เป็นขยะอยู่ข้างใน ดังนั้นเราต้องเพิ่มบิตพิเศษ (valid bit) เข้าไปในแต่ละเวิร์ดที่อยู่ในแคช เป็นตัวตรวจสอบความถูกต้องของข้อมูล
- แคชอาจจะเริ่มต้นด้วยการกำหนดบิตตรวจสอบของข้อมูลว่าเวิร์ดดังกล่าวเป็น 0 และบิตตรวจสอบข้อมูลจะถูกกำหนดเป็น 1 ก็ต่อเมื่อมีบล็อกของข้อมูลถูกโหลดเข้ามาจากหน่วยความจำหลักและเก็บไว้ในเวิร์ดดังกล่าง และมีค่าเป็น 1 ตลอดไปจนกระทั่งแคชถูกเริ่มทำงานใหม่หรือมีการรีสตาร์ทเครื่อง
- การนำบิตตรวจสอบความถูกต้องมาใช้งาน หมายความว่าเวิร์ดในแคชดังกล่าวจะไม่ถูกสับเปลี่ยนโดยบล็อกของข้อมูลใหม่ ถ้าบิตตรวจสอบความถูกต้องมีค่าเท่ากับ 0 แต่ข้อมูลดังกล่าวสามารถเข้าไปแทนที่ข้อมูลนั้นได้ทันที เพราะเป็นข้อมูลที่ไม่ถูกต้อง