หน่วยความจำแคช (Cache Memory)

บทนำ

ปัญหาสำคัญเกี่ยวกับความเร็วในการทำงานเมื่อหน่วยความจำหลัก (Main Memory) มีขนาดใหญ่ คือ โปรเซสเซอร์ (Processor หรือ Central Processing Unit: CPU) จะเรียกใช้งานข้อมูลได้ช้าเนื่องจากโปรเซสเซอร์ทำงานโดยใช้ flip-flop ซึ่งมีความเร็วมากกว่าหน่วยความจำหลัก และเมื่อโปรเซสเซอร์ต้องการเรียกใช้ข้อมูลหรือคำสั่งจากหน่วยความจำหลักก็จะทำให้เกิดปัญหาในเรื่องความเร็วในการรับส่งข้อมูล

วิธีในการจัดการกับปัญหานี้มีอยู่หลายแบบ หนึ่งในวิธีที่ใช้ได้ผลคือ การลดเวลาในการเรียกค้นข้อมูลในหน่วยความจำหลัก ปัจจัยหนึ่งของปัญหาคือ การเข้าถึงหน่วยความจำที่มีขนาดใหญ่เพียงบางส่วนจำเป็นจะต้องมีการรอก่อนที่จะสามารถเข้าถึงได้ในครั้งต่อไป

แนวคิดที่เกี่ยวกับการแยกส่วนของหน่วยความจำหลักให้มีความอิสระมากขึ้นและการเรียกใช้หรือการดำเนินการต้องมีความอิสระระหว่างกัน การที่มีหน่วยจัดเก็บของข้อมูลแบบ flip-flop จะทำให้เรียกใช้ข้อมูลในหน่วยความจำมีความพร้อมอยู่เสมอ ซึ่งจะทำให้ความเร็วในการทำงานเพิ่มขึ้น นอกจากนี้แล้วจะต้องมีการจัดการเวลาในการเข้าถึงและหน่วยที่ทำหน้าที่ตรวจสอบความถูกต้องของการดำเนินการ แต่อย่างไรก็ตาม ยังมีปัญหาอีกอย่างหนึ่งเกี่ยวกับแนวคิดนี้นั่นคือ ในกรณีที่มีความต้องการเข้าถึงข้อมูลในหน่วยความจำย่อยส่วนใดส่วนหนึ่งพร้อมกัน ก็จะทำให้เกิดปัญหาการรอคอยเช่นเดียวกับหน่วยความจำรวม

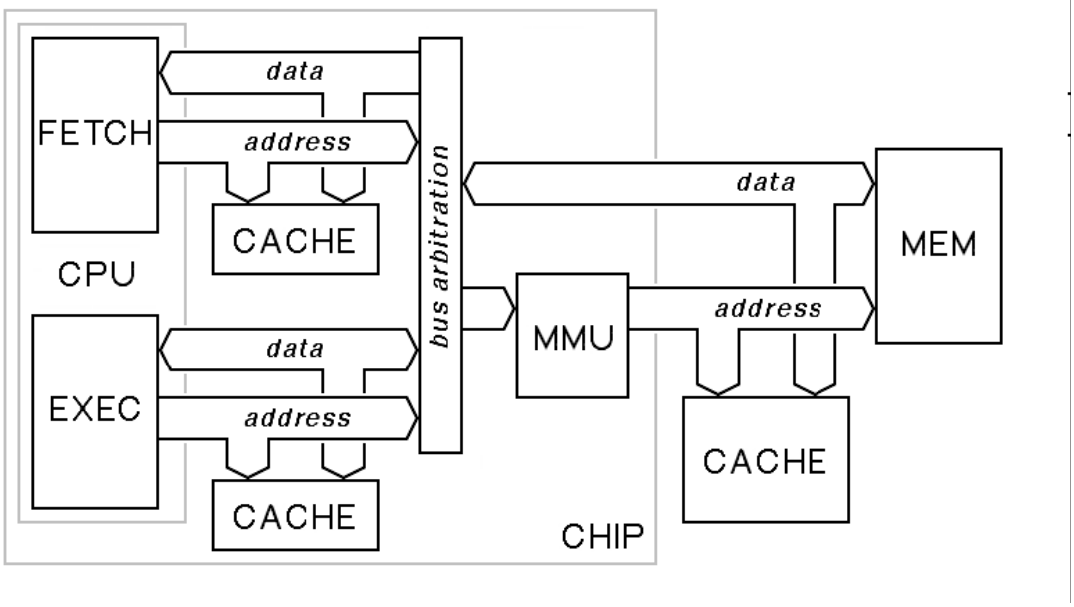
อีกวิธีการหนึ่งที่ใช้ในการแก้ปัญหาหน่วยความจำหลัก คือการเพิ่มหน่วยความจำขนาดเล็กที่เรียกว่า cache เข้าไปในโปรเซสเซอร์ ซึ่ง cache จะประกอบไปด้วยหน่วยจัดเก็บข้อมูลแบบ flip-flop และจะจัดเก็บข้อมูลหรือคำสั่งที่มีการเรียกใช้งานบ่อยๆ นอกจากนี้แล้วหากใน cache ไม่มีข้อมูลที่โปรเซสเซอร์ต้องการก็จะมีการอ้างไปยังหน่วยความจำหลักอีกครั้งหนึ่ง

ความหมาย

แคช (cache) คือ หน่วยความจำขนาดเล็กที่มีความเร็วสูงซึ่งเก็บข้อมูลหรือคำสั่งที่ถูกเรียกใช้หรือเรียกใช้บ่อย ๆ ข้อมูลและคำสั่งที่เก็บอยู่ในแคชซึ่งทำงานโดยใช้ SRAM (Static RAM) จะถูกดึงไปใช้งานได้เร็วกว่าการดึงข้อมูลจากหน่วยความจำหลักซึ่งใช้ DRAM (Dynamic RAM) หลายเท่าตัว

cache จะถูกพบบนระบบคอมพิวเตอร์ตั้งแต่ PCs ไปจนถึง Supercomputer ตัวแคชนั้นอาจจะถูกรวมไว้กับโปรเซสเซอร์และ Memory Management Unit (MMU) ในปัจจุบัน แต่หากไม่ได้รวมไว้ cache ก็จะถูกวางไว้บน CPU Board และเรียก cache ประเภทนี้ว่า cache ภายนอก

(External Cache) การที่ cache ถูกวางไว้ใกล้กับโปรเซสเซอร์ทำให้เป็นการลดเวลาในการเข้าถึงข้อมูลระหว่างโปรเซสเซอร์และ cache ศักยภาพในการทำงานก็จะสูงขึ้น และในทางกลับกันประสิทธิภาพในการทำงานจะลดลงหากนำ cache ไปไว้คนละที่กับโปรเซสเซอร์ แต่ในบางระบบก็ใช้ทั้ง On-chip Cache และ External Cache



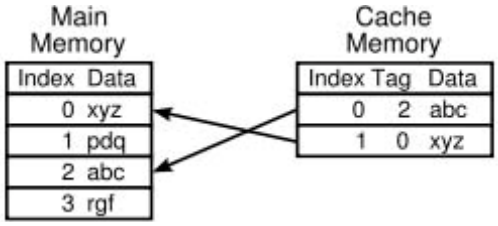
รูปที่ 1 แสดงสถาปัตยกรรมภายในระบบแบบสมบูรณ์

Cache L1

Cache L1 หรือแคชแบบไพรมารี หรือเรียกอีกอย่างหนึ่งว่า cache ภายใน (Internal Cache) ซึ่งติดตั้งอยู่ภายในตัวของซีพียูมาตั้งแต่เริ่มแรก โดยซีพียูที่เริ่มมีการติดตั้ง cache เป็นครั้งแรกนี้ คือ ซีพียูของอินเทล ต่อมาก็มีการติดตั้ง cache ให้กับซีพียูทุกตัวที่ผลิตมาหลังจากนี้

หน้าที่ของ Cache L1 นี้ก็คือใช้สำหรับเก็บข้อมูลและชุดคำสั่งชั่วคราวสำหรับซีพียูสำหรับนำไปประมวลผล cache ในซีพียูจะมีขนาดไม่ใหญ่มากนักเพราะหน่วยความจำ SRAM ที่นำมาทำนี้มีราคาแพง และพื้นที่ภายในชิปของโปรเซสเซอร์เองก็มีจำกัด ทำให้ขนาดของ Cache L1 จึงมีหน่วยเป็นกิโลไบต์ เช่น เพนเทียมคลาสลิกและเพนเทียม MMX จะมี Cache L1 ขนาดเท่ากับ 32 กิโลไบต์ หรือ AMD K6 และไซริกซ์ M2 ถึงแม้จะมีขนาดแคชใหญ่กว่าแต่ก็เท่ากับ 64 กิโลไบต์เท่านั้น

หน้าที่ของ cache

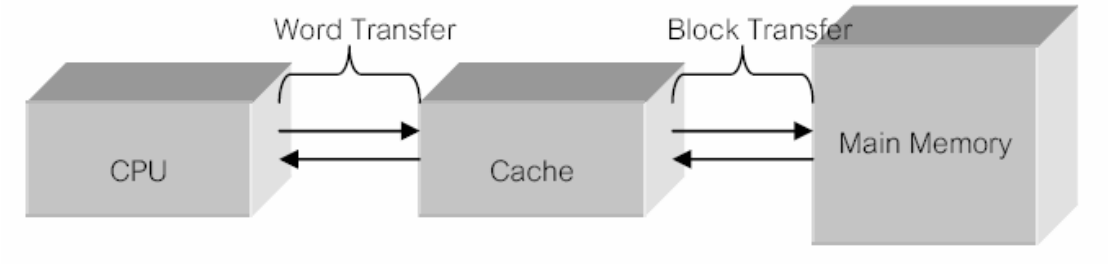


รูปที่ 2 แสดงตัวอย่างหน้าที่ของ Cache

จากรูปหน้าที่ของ cache เป็นที่เก็บอ้างอิงตำแหน่งของข้อมูลใน Ram ยกตัวอย่างดังภาพจะเห็นว่า Main Memory (Ram) จะมี Index อ้างอิงของข้อมูล เป็น 0 1 2 3 .... Cacheจะทำการเก็บ Index (tag) ของ Memory ไว้ เมื่อมีการร้องขอข้อมูลของโปรเซสเซอร์จะเข้ามาทำการค้นหาข้อมูลจาก cache L1 ก่อน หากไม่พบจะเข้าไปค้นหา cache L2 และ L3 (ถ้ามี) ตามลำดับ

ลักษณะพื้นฐานของหน่วยความจำ cache

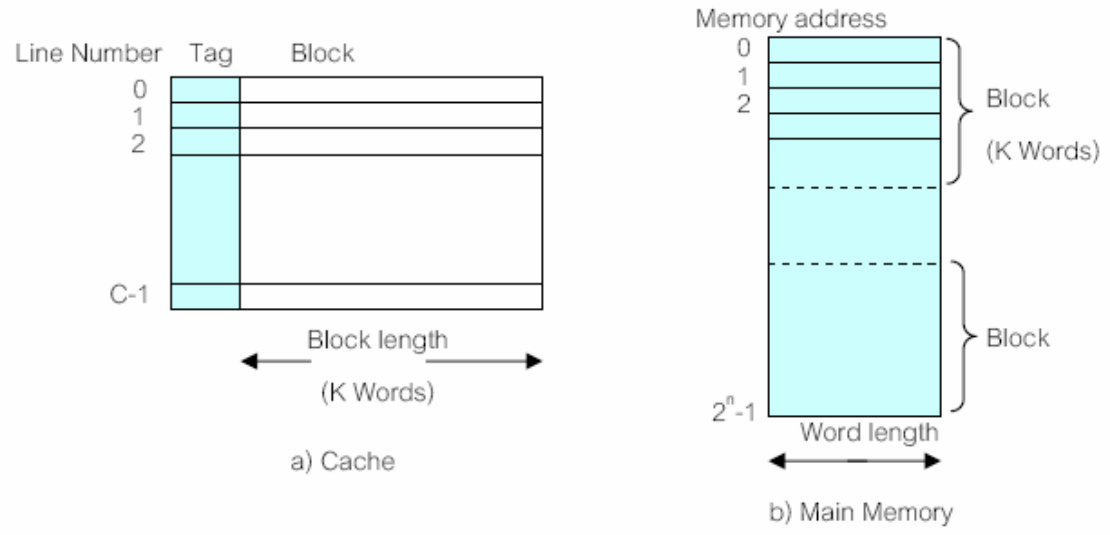
หน่วยความจำ cache สร้างขึ้นมาด้วยวัตถุประสงค์ที่จะให้เป็นหน่วยความจำที่ทำงานได้เร็วที่สุดเท่าที่เทคโนโลยีดีที่สุดในขณะนั้นจะทำได้ ในเวลาเดียวกันก็ต้องการให้มีขนาดใหญ่ที่สุดในราคาที่ไม่แพงจนเกินไปนัก แนวความคิดนี้ได้แสดงในรูปที่ 3



รูปที่ 3 แสดงหน่วยความจำ cache และหน่วยความจำหลัก

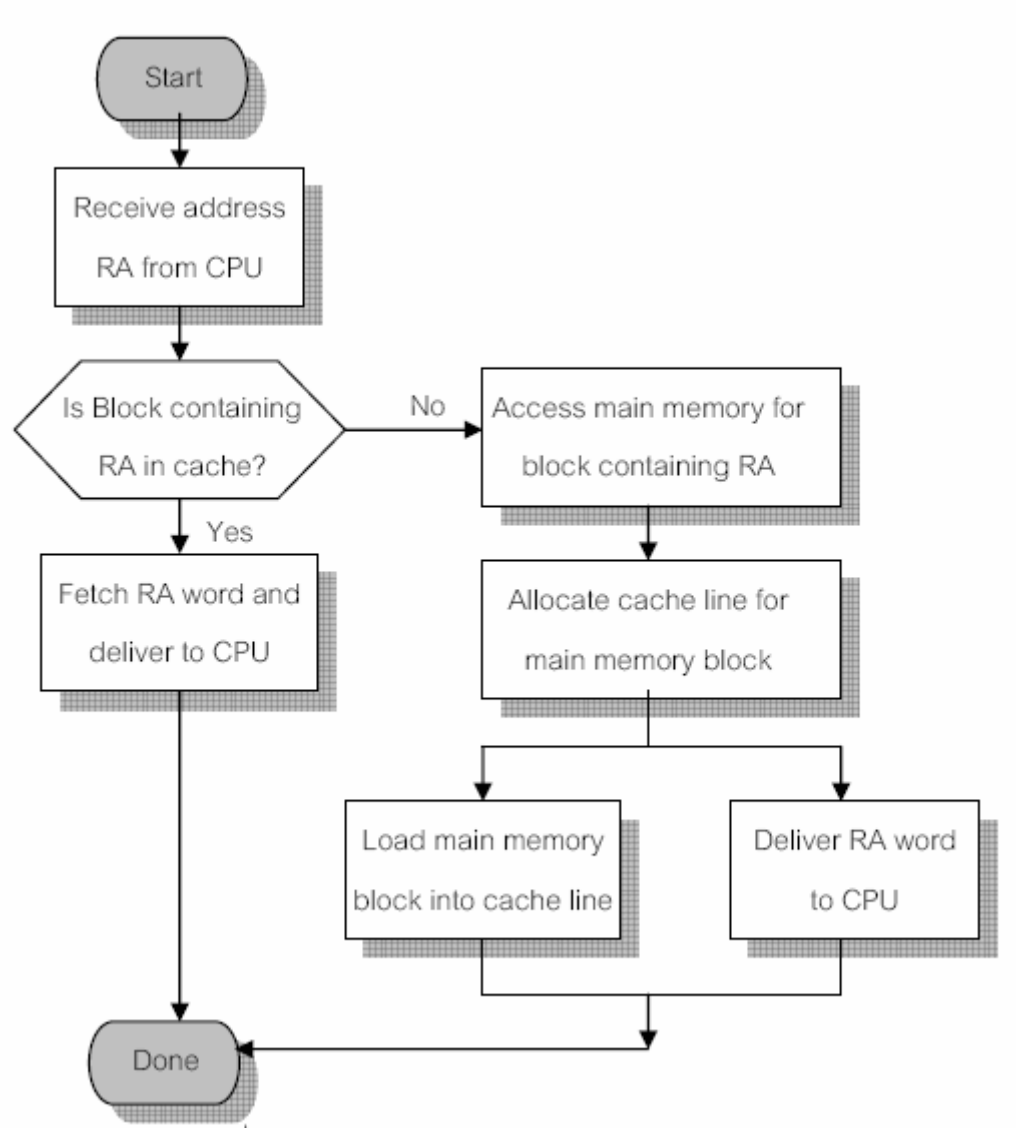
จากรูปในเครื่องคอมพิวเตอร์มีหน่วยความจำหลักที่มีความเร็วต่ำ (เมื่อเปรียบเทียบกับความเร็วของซีพียู)ที่มีปริมาณมากและมีหน่วยความจำ cache ที่เร็วกว่าแต่มีขนาดเล็กกว่า โดยปกติหน่วยความจำ cache จะเก็บสำเนาของข้อมูลบางส่วนในหน่วยความจำหลักเอาไว้ เมื่อโปรเซสเซอร์ต้องการอ่านข้อมูลจำนวนหนึ่ง word ในหน่วยความจำ ข้อมูลส่วนนั้นจะถูกตรวจสอบว่ามีอยู่ใน cache หรือไม่ ถ้ามีอยู่ก็จะนำข้อมูลใน cache ไปใช้ ถ้าไม่มีอยู่ก็จะเกิดการคัดลอกสำเนาข้อมูลหนึ่งบล็อกจากหน่วยความจำหลักมายัง cache แล้วจึงนำ word ที่ต้องการส่งต่อไปให้โปรเซสเซอร์ในภายหลัง เนื่องจากปรากฏการณ์การอ้างอิงในพื้นที่เดียวกัน (locality of reference) จะทำให้การอ้างอิงข้อมูลในหน่วยความจำครั้งต่อๆ ไปเป็นการอ้างอิงที่เดิมหรือที่ตำแหน่งใกล้เคียงจุดเดิม ดังนั้นการคัดลอก

ข้อมูลหนึ่งบล็อกจากหน่วยความจำหลักมายัง cache จะสามารถถูกนำมาใช้งานได้ในระยะหนึ่งก่อนที่จะมีการคัดลอกข้อมูลในครั้งต่อไป



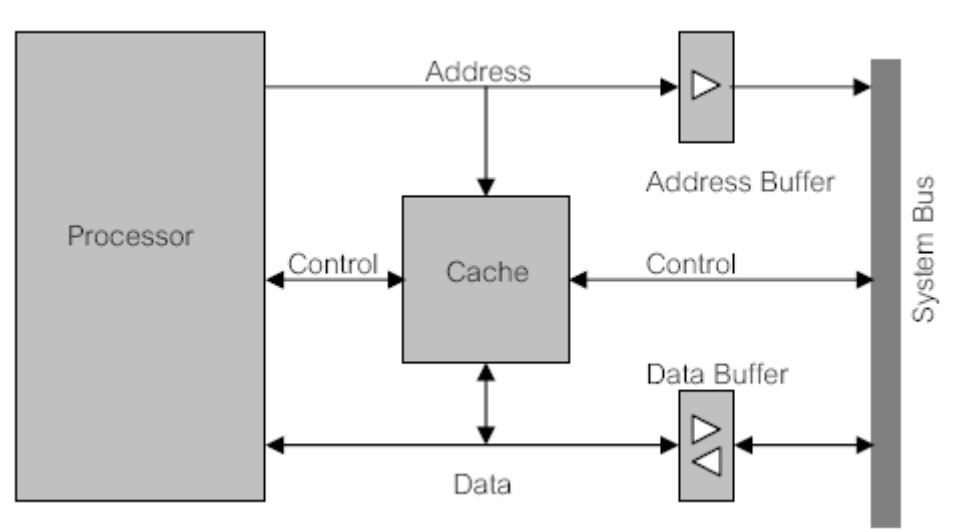
รูปที่ 4 แสดงโครงสร้างของหน่วยความจำ cache และหน่วยความจำหลัก

จากรูปหน่วยความจำหลักประกอบด้วยที่เก็บข้อมูล 2^n words ที่สามารถถูกอ้างอิงได้ โดยที่แต่ละ word จะมีหมายเลขที่อยู่เป็นเลขฐานสองจำนวน n-bit เป็นหมายเลขเฉพาะของตัวเอง หรือกล่าวอีกนัยหนึ่งว่า หน่วยความจำหลักประกอบด้วยบล็อกที่เก็บข้อมูลขนาด K-word จำนวนหนึ่ง นั่นคือมีจำนวนบล็อก (M) เท่ากับ $M=2^n/K$ บล็อกนั่นเอง ส่วน cache นั้นประกอบด้วยช่องสัญญาณจำนวน C ช่อง โดยที่แต่ละช่องมีขนาด K words และจำนวนช่องสายสัญญาณนั้นน้อยกว่าจำนวนบล็อกในหน่วยความจำหลักเป็นอย่างมาก ($C<<M$) ในเวลาใดก็ตาม ข้อมูลจำนวนหลายบล็อกจากหน่วยความจำหลักจะถูกคัดลอกสำเนามาเก็บไว้ที่ช่องสัญญาณใน cache ถ้าข้อมูลใน word ของบล็อกหนึ่งถูกเรียกใช้โดยโปรเซสเซอร์ ข้อมูลทั้งบล็อกนั้นจะถูกถ่ายโอนมาไว้ที่ cache เนื่องจากหน่วยความจำหลักมีจำนวนบล็อกมากกว่าจำนวนช่องสัญญาณใน cache ดังนั้นช่องสัญญาณแต่ละช่องจึงไม่ได้ถูกกำหนดไว้เป็นการเฉพาะสำหรับบล็อกใด ทำให้ช่องสัญญาณแต่ละช่องจะต้องมีป้ายบอกให้ทราบว่าข้อมูลในนั้นนำมาจากข้อมูลบล็อกใด โดยปกติป้ายมักจะประกอบด้วยตำแหน่งที่อยู่ของข้อมูลในหน่วยความจำหลัก



รูปที่ 5 แสดงการอ่านข้อมูลจากหน่วยความจำ cache

จากรูปแสดงขั้นตอนในกระบวนการอ่านข้อมูล โปรเซสเซอร์สร้างตำแหน่งที่อยู่ของข้อมูลที่ต้องการอ่าน เรียกว่า RA ถ้าข้อมูล word นั้นมีอยู่ใน cache อยู่แล้วก็จะถูกนำส่งให้โปรเซสเซอร์ทำการประมวลผลต่อไป มิฉะนั้นบล็อกในหน่วยความจำหลักที่มีข้อมูล word นี้อยู่ก็จะถูกคัดลอกสำเนาเข้ามาใส่ไว้ใน cache และถูกนำส่งโปรเซสเซอร์ต่อไป โดยจากรูปแสดงให้เห็นขั้นตอนต่างๆ ที่เกิดขึ้นซึ่งสะท้อนให้เห็นโครงสร้างของส่วนประกอบภายในดังที่แสดงในรูปที่ 6



รูปที่ 6 แสดงโครงสร้างหน่วยความจำ cache โดยทั่วไป

จากรูปเป็นโครงสร้างที่ใช้กับ cache ที่มีใช้งานทั่วไป ในโครงสร้างนี้ cache เชื่อมต่อกับโปรเซสเซอร์ผ่านสายสัญญาณข้อมูล สายบอกตำแหน่งที่อยู่ และสายควบคุมการทำงาน สายสัญญาณข้อมูลและสายบอกตำแหน่งที่อยู่จะเชื่อมต่อเข้ากับบัฟเฟอร์ซึ่งเชื่อมต่อเข้ากับสายบัสหลักของระบบที่นำไปสู่หน่วยความจำหลัก เมื่อสามารถค้นพบข้อมูลที่ต้องการใน cache (เรียกว่า cache hit) บัฟเฟอร์สำหรับข้อมูลและตำแหน่งจะถูกสั่งไม่ให้ทำงาน (disable) และการสื่อสารจะเกิดขึ้นระหว่างโปรเซสเซอร์กับ cache โดยไม่มีการใช้บัสหลักด้วย แต่ถ้าไม่สามารถหาข้อมูลที่ต้องการใน cache ได้ (เรียกว่า cache miss) ตำแหน่งข้อมูลที่ต้องการจะถูกส่งเข้าไปในบัสหลัก ข้อมูลในหน่วยความจำหลักจะถูกส่งมาที่บัฟเฟอร์และจากบัฟเฟอร์ส่งไปยังโปรเซสเซอร์และส่งเข้าไปเก็บไว้ใน cache สำหรับโครงสร้างแบบอื่น cache จะถูกวางคั่นกลางไว้ระหว่างโปรเซสเซอร์และหน่วยความจำหลักซึ่งจะวางทับสายสัญญาณข้อมูล สายบอกตำแหน่งข้อมูล และสายสัญญาณควบคุมไว้ทั้งหมด ดังนั้นเมื่อเกิดกรณี cache miss ข้อมูลจากหน่วยความจำหลักจะถูกส่งมาที่ cache ก่อนแล้วจึงถูกส่งต่อไปยังโปรเซสเซอร์ในลำดับหลัง

องค์ประกอบในการออกแบบ cache

องค์ประกอบที่เข้ามาเกี่ยวข้องในการออกแบบ cache ประกอบด้วย

1. ขนาดของ cache (Cache Size)

โดยทั่วไปนักออกแบบต้องการให้ cache มีขนาดเล็กเพียงพอที่จะทำให้ราคาเฉลี่ยต่อบิตนั้นใกล้เคียงกับราคาของหน่วยความจำหลัก และต้องการให้มีขนาดใหญ่เพียงพอเพื่อให้ค่าเฉลี่ยของเวลาในการเข้าถึงข้อมูลใกล้เคียงกับระยะเวลาในการเข้าถึงข้อมูลของ cache แรงจูงใจอื่นที่ต้องการให้ cache มีขนาดเล็ก ได้แก่ ขนาดที่ใหญ่ขึ้นของ cache จะไปเพิ่มจำนวนของเกต (gates: อุปกรณ์

อิเล็กทรอนิกส์ตัวหนึ่งที่เป็นส่วนประกอบของ cache) ที่เกี่ยวข้องกับการค้นหาตำแหน่งข้อมูลใน cache ผลที่เกิดขึ้นทำให้ cache มีขนาดใหญ่ขึ้นซึ่งจะทำให้ cache ทำงานได้ช้าลง เนื้อที่บนแผงวงจรก็เป็นอีกส่วนหนึ่งที่บังคับขนาดของ cache ให้มีขนาดที่จำกัด เนื่องจากประสิทธิภาพของ cache มีความเกี่ยวข้องอย่างใกล้ชิดกับลักษณะของงาน จึงแทบที่จะเป็นไปไม่ได้เลยที่จะคำนวณขนาดที่เหมาะสมที่สุดของ cache สำหรับในทุกกรณี ตารางที่ 1 แสดงขนาดของ cache ในโปรเซสเซอร์ที่มีใช้งานจำนวนหนึ่ง

ตารางที่ 1 ขนาดของหน่วยความจำ cache ในโปรเซสเซอร์บางชนิด

Processor	**Type**	**Year of Introduction**	**L1 cache**[a]	**L2 cache**	**L3 cache**
IBM 360/85	Mainframe	1968	16 to 32 KB	N	N
PDP-11/70	Minicomputer	1975	1 KB	N	N
VAX 11/780	Minicomputer	1978	16 KB	N	N
IBM 3033	Mainframe	1978	64 KB	N	N
IBM 3090	Mainframe	1985	128 to 256 KB	N	N
Intel 80486	PC	1989	8 KB	N	N
Pentium	PC	1993	8 KB/8 KB	256 to 512 KB	N
PowerPC 601	PC	1993	32 KB	N	N
PowerPC 620	PC	1996	32 KB/32 KB	N	N
PowerPC G4	PC/server	1999	32 KB/32 KB	256 KB to 1 MB	2 MB
IBM S/390 G4	Mainframe	1997	32 KB	256 KB	2 MB
IBM S/390 G6	Mainframe	1999	256 KB	8 MB	N
Pentium 4	PC/server	2000	8 KB/8 KB	256 KB	N
IBM SP	High-end	2000	64 KB/32 KB	8 KB	N

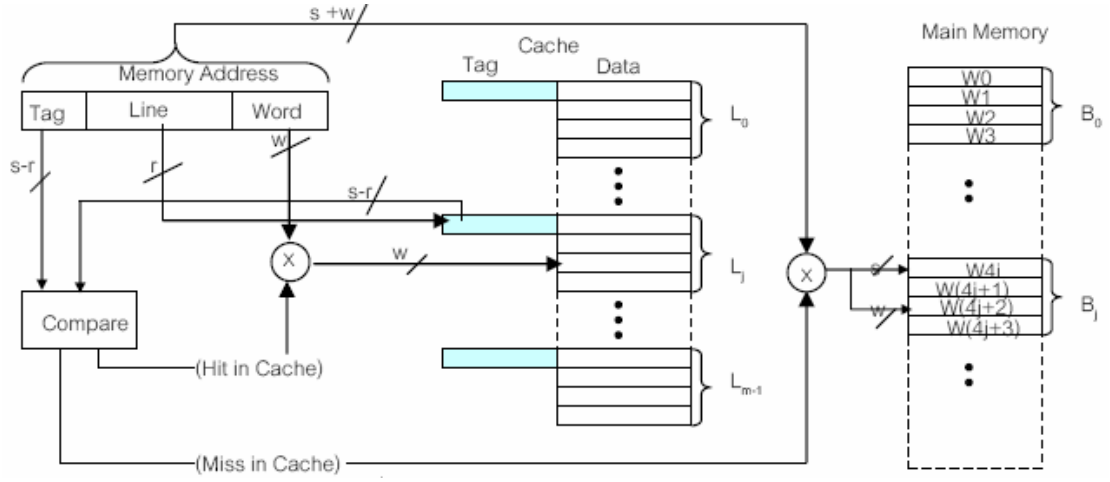
	server/supercomputer				
CRAY MTA[b]	PC/server	2001	16 KB/16 KB	96 KB	4 MB
Itanium	PC/server	2001	16 KB/16 KB	96 KB	4 MB
SGI Origin 2001	High-end server	2001	32 KB/32 KB	4 MB	N

2. ฟังก์ชันแม็บปิง (Mapping Function)

เนื่องจาก cache มีช่องสัญญาณน้อยกว่าจำนวนบล็อกในหน่วยความจำหลักจึงจำเป็นต้องมีอัลกอริทึมสำหรับการกำหนดตำแหน่งหน่วยความจำของแต่ละบล็อกไปยังช่องสัญญาณใน cache เรียกว่า การแม็บปิง (mapping) นอกจากนี้ ยังต้องมีวิธีการสำหรับการตรวจสอบว่าบล็อกใดที่ยังคงถูกเก็บไว้ในช่องสัญญาณ cache วิธีการกำหนดตำแหน่งหน่วยความจำไว้ในช่องสัญญาณ cache หรือการแม็บปิงจะเป็นสิ่งที่บอกให้ทราบถึงวิธีการวางโครงสร้างภายใน cache ซึ่งนิยมใช้ 3 วิธีคือ ไดเร็กแม็บปิง (Direct Mapping) แอสโซซิเอทีฟแม็บปิง (Associative Mapping) และเซ็ตแอสโซซิเอทีฟแม็บปิง เพื่อการเปรียบเทียบการทำงานของโครงสร้างทั้ง 3 แบบจึงกำหนดข้อสมมุติดังนี้

- ขนาดของ cache เป็น 64 Kbytes
- การถ่ายโอนข้อมูลระหว่างหน่วยความจำหลักกับ cache กระทำครั้งละหนึ่งบล็อก ซึ่งมีขนาด 4 ไบต์ (bytes) ซึ่งหมายความว่า จำนวนช่องสัญญาณใน cache นั้นเท่ากับ 16K ช่อง
- หน่วยความจำหลักมีขนาด 16 Mbytes ซึ่งใช้วิธีการกำหนดตำแหน่งข้อมูลโดยใช้หมายเลขฐานสองขนาด 24 บิต (2^{24}=16M) ดังนั้นเพื่อการกำหนดตำแหน่งบล็อกข้อมูลไปยัง cache จึงถือว่าหน่วยความจำหลักมีขนาด 16M บล็อก บล็อกละ 4 ไบต์

2.1 ไดเร็กแม็บปิง (Direct mapping) เป็นการจัดโครงสร้างภายใน cache แบบที่ง่ายที่สุด ซึ่งจะกำหนดตำแหน่งของแต่ละบล็อกไว้ที่ตำแหน่งช่องสัญญาณ cache ที่เดิมเสมอไม่มีการเปลี่ยนแปลงดังรูปที่ 7 ซึ่งแสดงกลไกการทำงานทั่วไป



รูปที่ 7 โครงสร้างหน่วยความจำ cache แบบ direct-mapping

และสามารถอธิบายด้วยรูปแบบฟังก์ชันทางคณิตศาสตร์ได้คือ

$$i = j \bmod m$$

เมื่อ

i คือ หมายเลขช่องสัญญาณ cache

j คือ หมายเลขบล็อก

m คือ จำนวนช่องสัญญาณของ cache

ฟังก์ชันสำหรับการแม็บปิงสามารถสร้างขึ้นมาได้ง่ายโดยใช้ตำแหน่งของบล็อกเป็นฐานในการคำนวณ การคำนวณให้พิจารณาว่า ตำแหน่งของข้อมูลในหน่วยความจำประกอบขึ้นด้วย 3 ส่วน บิตจากด้านซ้ายมือ (เริ่มจากบิตที่มีค่าน้อยที่สุด) จำนวน w บิตใช้ในการอ้างอิง word แต่ละ word ที่อยู่ในบล็อกหนึ่ง (ซึ่งในเครื่องคอมพิวเตอร์ทั่วไปมักจะหมายถึง การอ้างอิงไบต์แต่ละไบต์ในบล็อก) ที่เหลืออีกจำนวน s บิต หมายถึงตำแหน่งหรือหมายเลขของบล็อกจากจำนวนทั้งหมด 2^s บล็อก หน่วยความจำ cache จะแบ่งข้อมูล s บิตนี้ออกเป็นสองส่วน คือ เป็นบิตที่ใช้บอกหมายเลขช่องสัญญาณ cache จำนวน r บิต (แสดงว่ามีจำนวนช่องสัญญาณทั้งสิ้น 2^r ช่อง) และที่เหลืออีก s-r บิต ใช้เป็น tag หรือป้ายบอกตำแหน่งการใช้งานของบล็อกนั้น(อยู่ใน cache หรือไม่) ดังนั้น

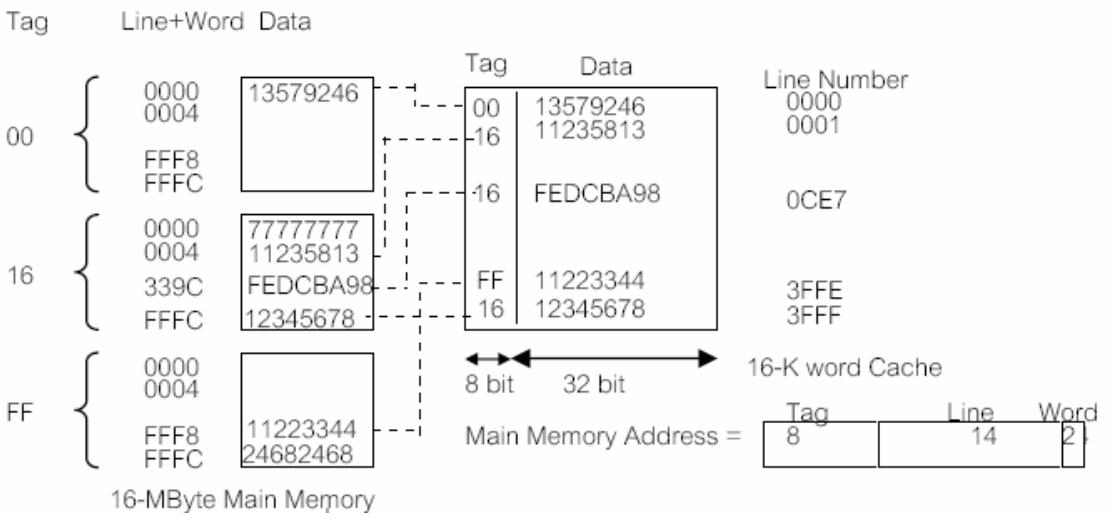
- ความยาวของหมายเลขที่มีอยู่ = (s+w) บิต
- ปริมาณหน่วยความจำที่สามารถอ้างอิงได้ทั้งหมด = 2^{s+w} words หรือไบต์
- จำนวนบล็อกทั้งหมด = 2^w words หรือไบต์
- จำนวนบล็อกในหน่วยความจำหลัก = $2^{s+w} / 2^w = 2^s$ บล็อก
- จำนวนช่องสัญญาณ cache = 2^r = m ช่อง
- ขนาดของป้ายบอกตำแหน่ง = (s-r) บิต

ผลของการกำหนดความสัมพันธ์ของตำแหน่งข้อมูลในหน่วยความจำหลักกับช่องสัญญาณ cache เป็นดังตารางที่ 2

ตารางที่ 2 แสดงความสัมพันธ์ของตำแหน่งข้อมูลในหน่วยความจำหลักกับช่องสัญญาณ cache

Cache line	Main Memory block assigned
0	0,m,2m,…,2^s-m
1	1, m+1, 2m+1,…, 2^s-m+1
…	
m-1	m-1, 2m-1, 3m-1,…., 2^s-1

การนำตัวเลขบางส่วนของหมายเลขที่อยู่มาใช้เป็นหมายเลขช่องสัญญาณ cache ทำให้เกิดการแปลงตำแหน่งแต่ละบล็อกในหน่วยความจำหลักมายัง cache เป็นค่าเฉพาะที่ไม่ซ้ำกัน เมื่อมีการอ่านข้อมูลจากบล็อกในหน่วยความจำหลักมาไว้ที่ cache ตามตำแหน่งที่กำหนดก็จะต้องเปลี่ยนค่าของป้ายบอกตำแหน่งของข้อมูลให้สอดคล้องกันด้วยเพื่อจะได้สามารถแยกความแตกต่างออกจากบล็อกที่ยังไม่เกิดการอ่านได้ ข้อมูล s-r บิตแรกจะถูกนำมาใช้เป็นป้ายบอกตำแหน่งข้อมูลนี้



รูปที่ 8 ตัวอย่างโครงสร้างหน่วยความจำ cache แบบ direct-mapping

จากรูปที่ 8 แสดงตัวอย่างการใช้วิธีไดเร็กแม็บปิง ซึ่งกำหนดให้ m = 16K = 2^{14} และ i = j modulo 2^{14} การแปลงที่อยู่จะได้ผลลัพธ์ดังนี้

Cache line	Main Memory block assigned
0	000000,010000,…,FF0000
1	000004, 010004,…, FF0004
…	
m-1	00FFFC,01FFFC, ..., FFFFFC

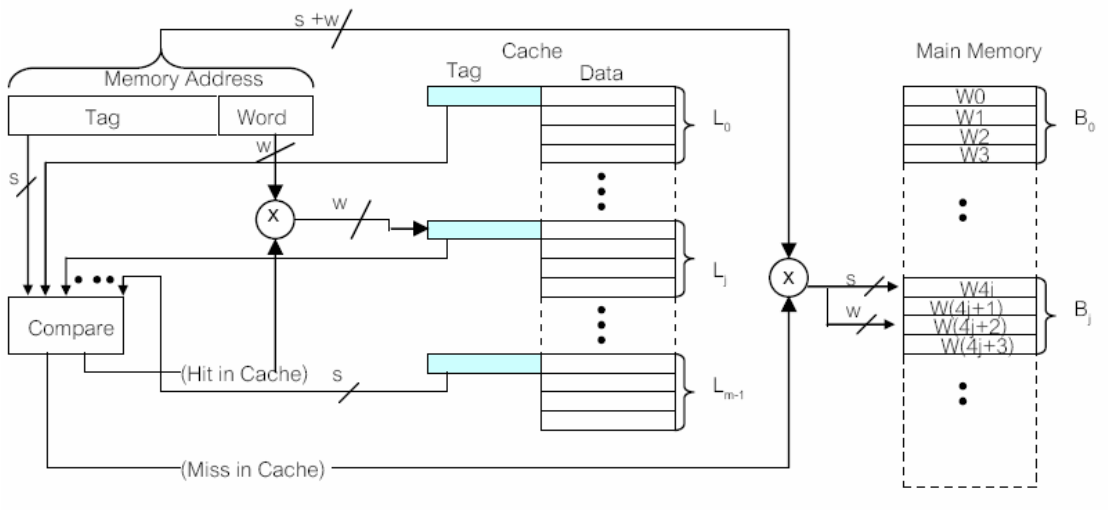
สังเกตว่าจะไม่มีบล็อกใดๆ ที่ถูกแปลงที่อยู่หมายเลขช่องสัญญาณ cache เดียวกันแล้วมีหมายเลขป้ายบอกตำแหน่งเหมือนกัน เช่น บล็อกที่เริ่มต้นด้วยหมายเลขที่อยู่ 000000,010000,…..,FF0000 จะมีค่าของป้ายบอกตำแหน่งเป็น 00,01,FF ตามลำดับ

เมื่อกลับไปพิจารณารูปที่ 5 จะพบว่ากระบวนการอ่านข้อมูลหน่วยความจำหลักทำงานดังนี้ ข้อมูลตำแหน่งที่อยู่ขนาด 24 บิตถูกส่งมาที่ระบบหน่วยความจำ cache ข้อมูลช่องสัญญาณ cache ขนาด 14 บิต (line number) ถูกนำไปใช้ในการค้นหาตำแหน่งช่องสัญญาณ cache ต่อไป ข้อมูลป้ายบอกตำแหน่ง 8 บิตแรก (tag) จะถูกนำไปเปรียบเทียบกับหมายเลขของป้ายบอกตำแหน่งที่เก็บอยู่ในตำแหน่งช่องสัญญาณ cache ที่พบนั้น ซึ่งถ้าตรงกันก็แสดงว่าข้อมูลบล็อกที่ต้องการนั้นอยู่ใน cache จึงนำข้อมูล 2 บิตสุดท้าย (word) มาใช้เลือกข้อมูลในบล็อกนั้น แต่ถ้าข้อมูลป้ายบอกตำแหน่งมีค่าไม่เท่ากันแสดงว่าข้อมูลในบล็อกนั้นยังไม่ถูกคัดลอกสำเนาจากหน่วยความจำหลักมาเก็บไว้ใน cache ก็จะนำข้อมูลป้ายบอกตำแหน่งรวมทั้งข้อมูลช่องสัญญาณ cache จำนวน 22 บิตมารวมกันกลายเป็นตำแหน่งบล็อกข้อมูลในหน่วยความจำหลักแล้วคัดลอกสำเนามาเก็บไว้ใน cache หมายเลขที่อยู่จริงๆ ในหน่วยความจำหลักนั้นมาจากข้อมูล 22 บิตที่กล่าวถึงและต่อท้ายด้วย 00 ทำให้กลายเป็นหลายเลขที่อยู่เริ่มต้นของข้อมูลในแต่ละบล็อกขนาด 24 บิตที่เก็บอยู่ในหน่วยความจำหลัก

เทคนิคไดเร็กแม็ปปิงนั้นมีข้อดีที่เป็นวิธีการง่ายและสามารถสร้างขึ้นมาใช้ได้ด้วยต้นทุนต่ำกว่าวิธีการแบบอื่น จุดด้อยที่สำคัญคือ การเปลี่ยนตำแหน่งบล็อกข้อมูลจากหน่วยความจำหลักมาที่ cache นั้นเป็นการกำหนดตำแหน่งคงที่ ดังนั้นถ้าในโปรแกรมเกิดมีการอ้างอิงถึงข้อมูลจาก 2 บล็อก(หรือมากกว่า) ที่อยู่ในตำแหน่งที่ต่างกันในหน่วยความจำหลัก แต่บังเอิญอยู่ในตำแหน่งเดียวกันใน cache แล้ว ข้อมูลทั้ง 2 บล็อกนั้นจะต้องสับเปลี่ยนกันถูกคัดลอกเข้ามาไว้ใน cache ที่ตำแหน่งเดียวกันเสมอทั้งๆ ที่ cache ส่วนอื่นอาจไม่มีข้อมูลอยู่เลยก็ได้ ปรากฏการณ์นี้เรียกว่า “trashing”

2.2 **แอสโซซิเอทีฟแม็บปิง (Associative mapping)** ได้แก้ปัญหาของไดเร็กแม็บปิงโดยการยินยอมให้หน่วยความจำในแต่ละบล็อกสามารถถูกอ่านเข้ามาไว้ใน cache ช่องใดก็ได้ ในกรณีนี้การแปลงตำแหน่งที่อยู่จะแบ่งออกเป็น 2 ส่วนคือ ป้ายบอกตำแหน่ง (tag) และตำแหน่งของ word ป้ายบอกตำแหน่งใช้ในการบอกหมายเลขบล็อกของหน่วยความจำหลัก การตรวจสอบว่าบล็อกนั้น

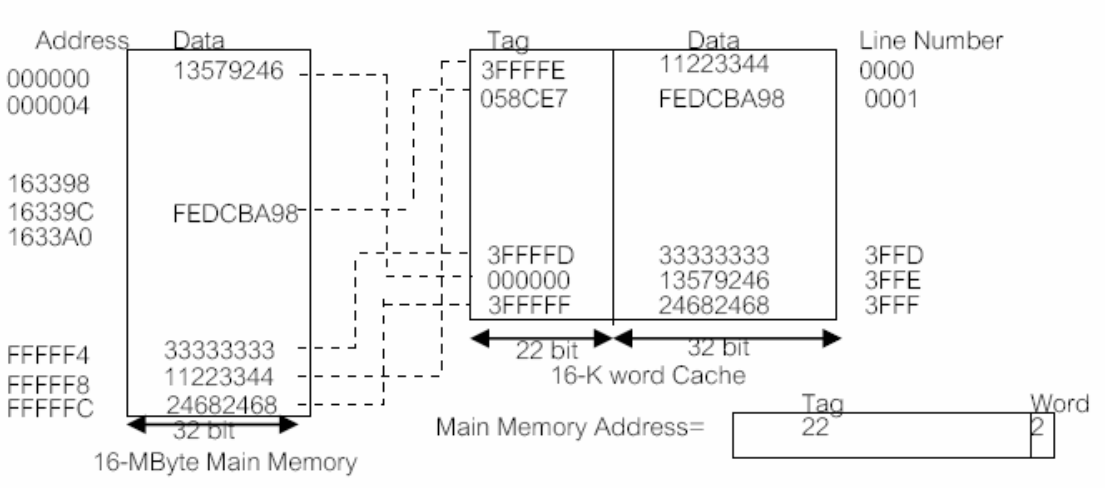
อยู่ใน cache หรือไม่ยังคงใช้วิธีเดิมคือ การเปรียบเทียบค่าของป้ายบอกตำแหน่งที่มากับข้อมูลกับค่าของป้ายบอกตำแหน่งที่เก็บอยู่ใน cache ดังรูปที่ 9



รูปที่ 9 โครงสร้างหน่วยความจำ cache แบบ Fully Associative

จากรูปแสดงถึงการทำงานของ**แอสโซซิเอทีฟแม็บปิง** สังเกตว่าไม่มีส่วนใดของตำแหน่งที่อยู่ข้อมูลเป็นส่วนที่บอกตำแหน่งใน cache (line number) ดังนั้นจึงไม่มีการตรวจสอบหมายเลขช่องสัญญาณใน cache ค่าต่าง ๆ สรุปได้ดังนี้

- ความยาวของหมายเลขตำแหน่งทีอยู่ (address length) = (s+w) บิต
- จำนวน word (หรือ ไบต์) ในหน่วยความจำ = 2^{s+w} words
- ขนาดของบล็อก = 2^{w} words (หรือ ไบต์)
- จำนวนบล็อกในหน่วยความจำ = $2^{s+w} / 2^{w} = 2^{s}$
- จำนวนช่องสัญญาณใน cache = ไม่มีการกำหนดตายตัว
- ขนาดของป้ายบอกตำแหน่ง = s บิต



รูปที่ 10 โครงสร้างหน่วยความจำ cache แบบ associative

จากรูปแสดงตัวอย่างในการใช้แอสโซซิเอทีฟแม็บปิงที่อยู่ในหน่วยความจำ หลักประกอบขึ้นจากป้ายบอกตำแหน่งขนาด 22 บิตและเลขบอกตำแหน่งไบต์ขนาด 2 บิตรวมเป็น 24 บิต ป้ายบอกตำแหน่งทั้ง 22 บิต (เริ่มจากบิตซ้ายสุด) จะต้องใช้ในการเก็บข้อมูลบล็อกละ 22 บิต (เริ่มจากบิตซ้ายสุด) จะต้องใช้ในการเก็บข้อมูลบล็อกละ 32 บิตสำหรับแต่ละช่องสัญญาณใน cache เช่น

Memory Address	0001	0110	0011	0011	1001	1100	(Binary)
	1	6	3	3	9	C	(Hex)
Tag(left-most 22 bit)	00	0101	1000	1100	1110	0111	(Binary)
	0	5	8	C	E	7	(Hex)

นั่นคือที่อยู่อ้างอิงในหน่วยความจำหลัก 16339C จะมีหมายเลข tag เป็น 058CE7

วิธีการแบบแอสโซซีเอทีฟแม็บปิงมีความอ่อนตัวเมื่อบล็อกในหน่วยความจำจะต้องถูกแทนที่ด้วยบล็อกใหม่ อัลกอริทึมสำหรับการแทนที่ (Replacement Algorithm) ถูกออกแบบมาเพื่อเพิ่มอัตราการค้นพบข้อมูลที่ต้องการใน cache โดยลดอัตราการอ้างอิงที่หน่วยความจำหลักลง ข้อเสียของวิธีการนี้ คือ ความซับซ้อนที่เกิดขึ้นในระหว่างการตรวจสอบป้ายบอกตำแหน่งข้อมูลใน cache ที่จะต้องทำการตรวจสอบพร้อมกันทุกตำแหน่ง

2.3 เซ็ตแอสโซซิเอทีฟแมบปิง(Set associative mapping) เป็นวิธีการผสมที่ได้นำข้อดีของทั้งแบบไดเร็กแมบปิงและแอสโซซิเอทีฟแม็บปิงมาใช้ ในขณะที่พยายามลดข้อด้อยให้มีผลกระทบน้อยลง ในกรณีนี้หน่วยความจำ cache จะถูกแบ่งเป็นเซ็ตจำนวน v เซ็ต แต่ละเซ็ตประกอบด้วยช่องสัญญาณจำนวน k ช่อง สรุปความสัมพันธ์ได้ดังนี้

$$m = v \times k$$

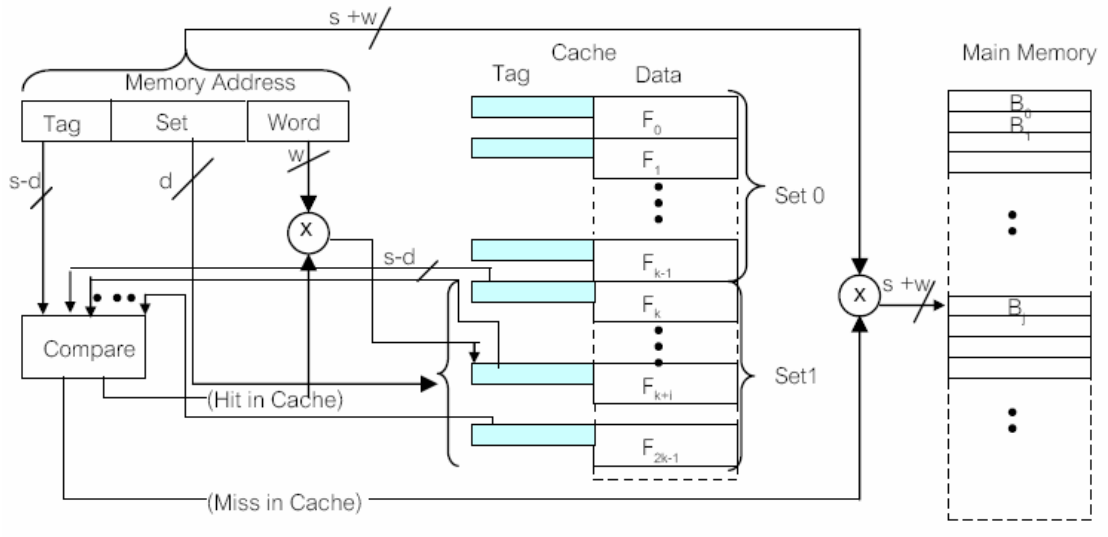
$$i = j \text{ modulo } v$$

เมื่อ i = หมายเลขเซ็ตใน cache

j = หมายเลขบล็อกในหน่วยความจำ

m = จำนวนช่องสัญญาณใน cache

ความสัมพันธ์ที่กล่าวถึงข้างบนนี้เรียกเป็นชื่อเฉพาะว่า “k-way set associative mapping” วิธีการเซ็ตแอสโซซิเอทีฟนี้จะแปลงที่อยู่ของบล็อก B_j ให้ไปอยู่ในช่องสัญญาณใดก็ได้ในเซ็ต i ของหน่วยความจำ cache โครงสร้างของหมายเลขที่อยู่ประกอบด้วย 3 ส่วนคือ ป้ายบอกตำแหน่ง (tag) หมายเลขเซ็ต และตำแหน่งใน word หมายเลขเซ็ตมีขนาด d บิต ($v=2^d$ set) ใช้ในการกำหนดเซ็ตที่ต้องใช้ใน cache ส่วนป้ายบอกตำแหน่งขนาด s บิตและข้อมูลในเซ็ตจะนำมาใช้บอกตำแหน่งบล็อกข้อมูลในหน่วยความจำ

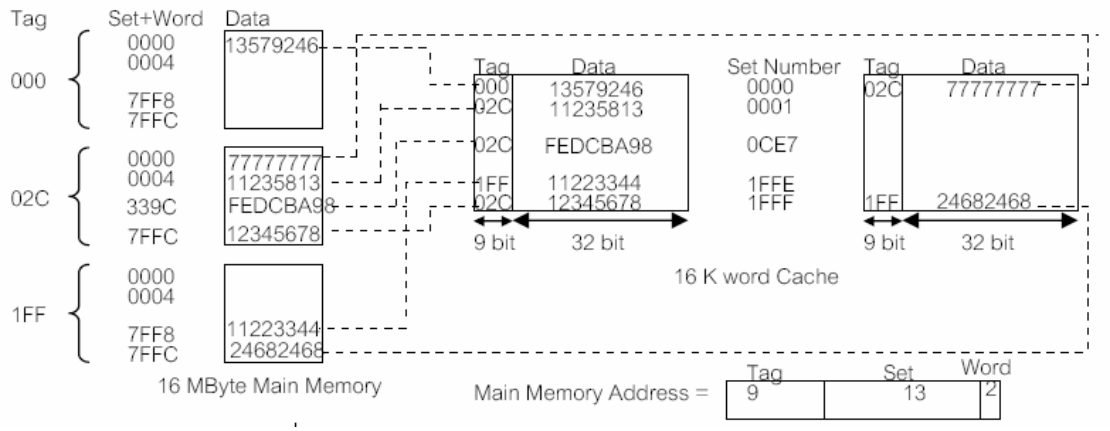


รูปที่ 11 โครงสร้างหน่วยความจำ cache แบบ k-way set associative

จากรูปแสดงการทำงานของเซ็ตแอสโซซิเอทีฟกรณีพิเศษกรณีหนึ่งเรียกว่า fully associative mapping มีขนาดของป้ายบอกตำแหน่งมากจนต้องทำการเปรียบเทียบกับทุกช่องสัญญาณใน cache (คือมีจำนวนเซ็ตทั้งหมดเป็น 1 เซ็ตเท่านั้น) แต่ k-way set associative จะมีขนาดของป้ายบอกตำแหน่งที่เล็กกว่ามาก ซึ่งจะทำการเปรียบเทียบกับช่องสัญญาณเพียง k ช่องในเซ็ตหนึ่งๆ เท่านั้นซึ่งสามารถสรุปความสัมพันธ์ได้ดังนี้

- ความยาวของตำแหน่งข้อมูล(address length) = s + w บิต
- ขนาดหน่วยความจำทั้งหมด = 2^{s+w} word or bytes
- ขนาดของบล็อก = 2^w word or bytes
- จำนวนบล็อกในหน่วยความจำหลัก = $2^{s+w} / 2^w = 2^s$ บล็อก
- จำนวนช่องสัญญาณในแต่ละเซ็ต = k
- จำนวนเซ็ต = 2^d = v จำนวน

- จำนวนช่องสัญญาณทั้งหมดใน cache = kv = k x 2^d
- ขนาดของป้ายบอกตำแหน่ง = (s-d) บิต



รูปที่ 12 ตัวอย่างโครงสร้างหน่วยความจำ cache แบบ 2-way set associative

จากรูปแสดงตัวอย่างการทำงานของเซ็ตแอสโซซิเอทีฟที่แต่ละเซ็ตประกอบด้วย 2 ช่องสัญญาณ (2-way set associative) หมายเลขเซ็ตขนาด 13 บิตใช้ระบุหมายเลขเซ็ตและหมายเลขบล็อกในหน่วยความจำหลัก ดังนั้น บล็อกหมายเลข 000000,008000,…,FF8000 ของหน่วยความจำหลักจะถูกแปลงให้ไปอยู่ในเซ็ต 0 ใน cache แต่ละบล็อกสามารถถูกอ่านเข้าไปเก็บไว้ในช่องสัญญาณใด (1 ใน 2 ช่อง) ของเซ็ต 0 ก็ได้ แต่จะไม่สามารถเก็บไว้ในช่องสัญญาณเดียวกันในเซ็ตเดียวกันได้ จากนั้นค่าของป้ายบอกตำแหน่งที่ตำแหน่งข้อมูลที่ถูกอ้างอิงจะถูกนำมาเปรียบเทียบกับค่าของป้ายบอกตำแหน่งที่เก็บที่อยู่ใน cache พร้อมกันทั้ง 2 ช่องสัญญาณเพื่อค้นหาบล็อกข้อมูลที่ต้องการ

ในกรณีที่ v = m และ k = 1 จะทำให้วิธีนี้กลายเป็นวิธีไดเร็กแม็บปิงไปโดยปริยาย และถ้า v = 1 และ k = m ก็จะกลายเป็นวิธีแอสโซซิเอทีฟแม็บปิง การกำหนดให้แต่ละเซ็ตมี 2 ช่องสัญญาณ (v=m/2 และ k=2) เป็นวิธีการที่นิยมใช้กันโดยทั่วไป ซึ่งสามารถเพิ่มประสิทธิภาพของอัตราการค้นพบข้อมูลใน cache ได้ดีกว่าวิธีไดเร็กแม็บปิงมาก และถ้ากำหนดให้แต่ละเซ็ตมี 4 ช่องสัญญาณ (v=m/4 และ k=4) ซึ่งเรียกว่า 4-way set associative ก็จะเพิ่มประสิทธิภาพให้ดียิ่งขึ้นไปอีก โดยที่มีค่าใช้จ่ายเพิ่มขึ้นเล็กน้อย แต่ถ้าเพิ่มจำนวนช่องสัญญาณต่อเซ็ตให้สูงกว่านี้แล้วประสิทธิภาพที่ได้รับจะเพิ่มขึ้นเพียงเล็กน้อยเท่านั้นซึ่งไม่คุ้มกับค่าใช้จ่ายที่เพิ่มสูงขึ้น

3. อัลกอริทึมการแทนที่ (Replacement Algorithm)

เมื่อข้อมูลบล็อกใหม่ถูกอ่านขึ้นมาจากหน่วยความจำ หนึ่งในบล็อกเดิมที่ถูกอ่านเข้ามาไว้ใน cache จะต้องถูกแทนที่ด้วยบล็อกใหม่ สำหรับไดเร็กแม็บปิงจะมีช่องสัญญาณเพียงช่องเดียวเท่านั้นที่จะเป็นที่อยู่ใน cache สำหรับข้อมูลแต่ละบล็อก จึงไม่จำเป็นต้องคิดหาวิธีการแทนที่ใดๆ แต่สำหรับแอสโซซิเอทีฟและเซ็ตแอสโซซิเอทีฟนั้นจะต้องนำวิธีการสำหรับการแทนที่

(Replacement Algorithm) มาใช้งานเพื่อตอบสนองความรวดเร็วในการทำงาน วิธีการดังกล่าวจะต้องถูกสร้างขึ้นด้วยฮาร์ดแวร์ วิธีการที่นิยมนำมาใช้กันโดยทั่วไปมี 4 วิธี

3.1 แอลอาร์ยู (Least Recently Used: LRU) เป็นวิธีที่นิยมใช้กันมากที่สุดคือ ซึ่งจะนำบล็อกใหม่มาแทนที่บล็อกเก่าที่ถูกเก็บไว้ใน cache เป็นเวลานานที่สุด โดยที่มีการนำไปใช้งานน้อยที่สุด สำหรับ 2-way set associative สามารถสร้างขึ้นมาใช้งานได้ง่ายมาก แต่ละช่องสัญญาณในเซ็ตจะมีบิตพิเศษเรียกว่า USE บิต เมื่อมีการอ้างอิงถึงช่องสัญญาณหนึ่งก็จะเปลี่ยนค่า USE ให้เป็น “1” และกำหนดให้ USE บิตของอีกช่องสัญญาณให้เป็น “0” เมื่อต้องการนำข้อมูลบล็อกใหม่เข้ามาในเซ็ตนั้นก็จะเลือกใช้ช่องสัญญาณที่มีค่า USE เป็น “0” ทั้งนี้แนวความคิดนี้คาดว่าข้อมูลที่ถูกอ้างอิงถึงในลำดับหลังมีโอกาสที่จะถูกอ้างอิงมากกว่าข้อมูลที่ไม่ถูกอ้างอิงมาเป็นเวลานานกว่า

3.2 วิธีการแบบเข้าคิว (First-in-First-out: FIFO) ซึ่งจะทดแทนบล็อกที่ถูกอ่านขึ้นมานานมากที่สุด

3.3 แอลเอฟยู (Lease Frequently Used: LFU) จะทำการทดแทนบล็อกที่ถูกอ้างอิงเป็นจำนวนครั้งน้อยที่สุด การสร้างขึ้นมาใช้งานจำเป็นจะต้องมีตัวนับการใช้งานข้อมูลแต่ละช่องสัญญาณ เพื่อหาช่องสัญญาณที่มีการนำมาใช้งานน้อยที่สุด

3.4 วิธีการสุ่ม (Random) เป็นวิธีที่ไม่เกี่ยวข้องกับการใช้งานเลย คือจะเลือกช่องสัญญาณที่จะถูกทดแทนด้วยวิธีการสุ่มหมายเลขช่องสัญญาณขึ้นมาในแต่ละครั้งที่มีความต้องการเกิดขึ้น ผลจากการสร้างระบบจำลองการทำงานพบว่า วิธีการนี้มีประสิทธิภาพด้อยกว่าวิธีการที่นำข้อมูลเกี่ยวกับการใช้งานมาพิจารณาแต่ก็ไม่มากนัก

4. นโยบายการบันทึกลงหน่วยความจำหลัก (Write Policy)

ก่อนที่บล็อกข้อมูลใน cache จะถูกแทนที่ด้วยข้อมูลบล็อกใหม่จำเป็นจะต้องตรวจสอบว่าข้อมูลในบล็อกนั้นถูกแก้ไขเปลี่ยนแปลงไปบ้างแล้วหรือไม่ ซึ่งถ้าไม่มีการแก้ไขเกิดขึ้นก็สามารถบันทึกข้อมูลบล็อกใหม่แทนที่ได้เลย มิฉะนั้นแสดงว่าจะต้องมีการบันทึกข้อมูล (write operation) เกิดขึ้นอย่างน้อยหนึ่งครั้งกับข้อมูลนั้น ทำให้จะต้องมีการปรับปรุงข้อมูลในหน่วยความจำหลักให้เหมือนกับข้อมูลใน cache มีวิธีการกำหนดนโยบายการบันทึกข้อมูลจาก cache ลงหน่วยความจำหลัก (write policy) อยู่หลายวิธี ซึ่งมีความแตกต่างกันในเรื่องประสิทธิภาพและค่าใช้จ่าย ปัญหาที่เกิดขึ้นมีอยู่ 2 ประการคือ ประการแรก อาจมีอุปกรณ์มากกว่าหนึ่งอย่างที่ทำการอ่านหรือบันทึกข้อมูลในหน่วยความจำ เช่น อุปกรณ์ไอโอ ได้รับอนุญาตให้อ่านหรือบันทึกข้อมูลไปยังหน่วยความจำได้โดยตรง ถ้าข้อมูลบล็อกนั้นถูกแก้ไขใน cache อุปกรณ์ไอโอก็จะอ่านหรือบันทึกข้อมูลเก่าในหน่วยความจำหลักซึ่งไม่ถูกต้อง หรือในทางกลับกัน ถ้าอุปกรณ์ไอโอมีการแก้ไขข้อมูลในหน่วยความจำหลัก บล็อกข้อมูลที่อยู่ใน cache ก็จะเป็นข้อมูลเก่าซึ่งก็ไม่ถูกต้องเช่นกัน ปัญหาที่อาจซับซ้อนมากกว่านี้เกิดขึ้นในระบบคอมพิวเตอร์ที่มีโปรเซสเซอร์หลายตัว ซึ่ง

โปรเซสเซอร์แต่ละตัวต่างก็มี cache เป็นของตนเองก็จะทำให้ข้อมูลบล็อกเดียวกันแต่มีอยู่หลายสำเนาซึ่งอาจมีค่าแตกต่างกันทั้งหมด

วิธีการที่ง่ายที่สุดเป็นการบันทึกข้อมูลลงหน่วยความจำในทันที หรือ “write through” ทุกครั้งที่มีการบันทึกข้อมูลลงใน cache ก็จะถูกบังคับให้เกิดการบันทึกลงในหน่วยความจำหลักในทันที เพื่อเป็นการรับประกันว่าข้อมูลทั้งที่อยู่ในหน่วยความจำหลักและที่อยู่ในทุกส่วนของ cache นั้นมีค่าเท่ากันเสมอ อุปกรณ์อื่นที่มีการเชื่อมต่อระหว่างโปรเซสเซอร์และ cache ก็จะคอยตรวจตราดูความเคลื่อนไหวของหน่วยความจำหลัก และจะต้องคอยปรับปรุงข้อมูลในส่วนของตนเองให้สอดคล้องกับข้อมูลในหน่วยความจำหลักเสมอ ข้อเสียของวิธีการนี้คือ เป็นวิธีที่ทำให้เกิดการส่งข้อมูลผ่านบัส โดยเฉพาะส่วนที่เชื่อมต่อกับหน่วยความจำหลักเป็นอย่างมากซึ่งอาจทำให้เกิดเป็นจุดปัญหาคอขวดของระบบขึ้นมาได้ อีกวิธีการหนึ่งเป็นการบันทึกข้อมูลในภายหลัง หรือ “write back” ซึ่งช่วยลดจำนวนครั้งในการบันทึกข้อมูลลงหน่วยความจำหลักให้เหลือน้อยที่สุด วิธีนี้จะปล่อยให้มีการปรับปรุงแก้ไขข้อมูลเกิดขึ้นใน cache เท่านั้น ซึ่งจะไปกำหนดค่าของบิตพิเศษเรียกว่า UPDATE บิต เมื่อมีการแก้ไขเกิดขึ้นจริง ถ้าข้อมูลในบล็อกนั้นจะต้องถูกแทนที่ด้วยข้อมูลบล็อกใหม่ระบบก็จะตรวจสอบ UPDATE บิต และจะทำการบันทึกข้อมูลลงในหน่วยความจำหลักก็ต่อเมื่อบิตนี้มีค่าเปลี่ยนแปลงไปจากเดิมเท่านั้น ข้อเสียของวิธีการนี้คือ ข้อมูลใน cache และในหน่วยความจำอาจจะไม่เหมือนกัน ดังนั้นอุปกรณ์ไอโอทั้งหลายก็จะต้องติดต่อหน่วยความจำหลักผ่านระบบ cache เท่านั้น ซึ่งทำให้วงจรการทำงานซับซ้อนและอาจทำให้เกิดเป็นจุดคอขวดขึ้นได้เหมือนกัน

ในระบบที่มีโครงสร้างของบัสที่มีอุปกรณ์เชื่อมต่อมากกว่าหนึ่งอย่างที่มี cache เป็นของตนเอง (เช่น โปรเซสเซอร์) และมีการใช้หน่วยความจำร่วมกันจะทำให้เกิดปัญหาใหม่ขึ้นมา ถ้าข้อมูลใน cache ของอุปกรณ์ตัวหนึ่งถูกแก้ไขก็จะทำให้ข้อมูลบล็อกนั้นในหน่วยความจำหลักและข้อมูลนั้นที่เก็บอยู่ใน cache ส่วนอื่นๆไม่ถูกต้องตามไปด้วย ถึงแม้ว่าจะแก้ปัญหานี้ด้วยการใช้วิธี write through ก็ตาม ข้อมูลใน cache ของอุปกรณ์ตัวอื่นก็จะยังคงไม่ถูกต้องอยู่ดี ระบบที่มีการป้องกันและแก้ปัญหานี้เรียกว่ามีการทำ cache coherency ซึ่งมีวิธีการต่างๆ ดังนี้

- **Bus watching with write through:** หน่วยควบคุม cache จะคอยตรวจสายสัญญาณตำแหน่งที่อยู่ (address lines) เพื่อค้นหาการบันทึกข้อมูลไปยังหน่วยความจำหลักของอุปกรณ์ที่กำลังใช้งานบัสอยู่ ถ้าเกิดมีอุปกรณ์กำลังบันทึกข้อมูลลงไปในพื้นที่ในหน่วยความจำหลักที่มีการใช้งานร่วมกันซึ่งก็ถูกอ่านเข้ามาไว้ใน cache ของตนเองแล้วหน่วยควบคุม cache ก็จะไม่อนุญาตให้ใช้ข้อมูลใน cache ส่วนนั้นในทันที (เสมือนว่าข้อมูลส่วนนั้นยังไม่ได้ถูกอ่านเข้ามาใน cache) วิธีการนี้จะต้องใช้งานร่วมกับ write through เท่านั้น

- **Hardware transparency:** เป็นการเพิ่มอุปกรณ์เข้าในระบบคอมพิวเตอร์เพื่อทำให้แน่ใจว่าการปรับปรุงข้อมูลทั้งหมดที่เกิดขึ้นกับหน่วยความจำหลักจะถูกตรวจพบ และข้อมูลที่ถูกแก้ไข

นั้นจะต้องนำไปแก้ไขใน cache ทั้งหมด (ที่เกี่ยวข้อง) ด้วย ดังนั้นถ้ามีโปรเซสเซอร์ตัวหนึ่งทำการแก้ไขข้อมูลใน cache ของตนเอง ข้อมูลนั้นจะถูกแก้ไขในหน่วยความจำหลักและจะถูกนำไปปรับปรุง cache ของโปรเซสเซอร์อื่นที่มีการคัดลอกสำเนาข้อมูลนั้นไปใช้

- Noncacheable memory: กำหนดให้พื้นที่บางส่วนของหน่วยความจำหลักเป็นพื้นที่ที่อนุญาตให้มีการใช้งานร่วมกันได้ระหว่างโปรเซสเซอร์ต่างๆ แต่ไม่อนุญาตให้หน่วยควบคุม cache ของโปรเซสเซอร์ทุกตัวเข้ามาควบคุมพื้นที่นี้ (คือไม่ยอมให้ทำ cache กับข้อมูลในพื้นที่นี้) ในระบบนี้จะกำหนดให้การอ้างอิงถึงข้อมูลในพื้นที่พิเศษนี้ของโปรเซสเซอร์ทุกตัวเป็น cache miss เสมอ คือจะต้องอ่านจากหน่วยความจำโดยตรงทุกครั้งที่มีการอ้างอิงถึง

5. **ขนาดของบล็อกใน cache (Line Size)**

องค์ประกอบที่สำคัญอีกอย่างหนึ่งคือขนาดของบล็อกใน cache เมื่อเกิดการอ่านบล็อกข้อมูลมาจากหน่วยความจำหลักและใส่เข้าไปใน cache นั้น นอกเหนือจาก word ที่ต้องการแล้วยังได้รับข้อมูลอื่นที่อยู่ติดกับ word นั้นๆ มาด้วยอีกจำนวนหนึ่ง ถ้าบล็อกมีขนาดใหญ่ขึ้นอัตราการพบข้อมูลใน cache จะเพิ่มขึ้นเป็นระยะเวลาหนึ่งเนื่องจากกฎการอ้างอิงพื้นที่ใกล้เคียง (locality of reference) ซึ่งกล่าวว่า การอ้างอิงข้อมูลในครั้งต่อๆ ไปจะอ้างอิงข้อมูลเดิมหรือข้อมูลที่อยู่ใกล้เคียงกับที่เดิม ดังนั้นบล็อกที่มีขนาดใหญ่ขึ้นจึงสามารถนำข้อมูลที่เป็นประโยชน์เข้ามาสู่ cache ได้มากขึ้น อย่างไรก็ตาม ถ้าบล็อกมีขนาดใหญ่ขึ้นเรื่อยๆ โอกาสที่จะพบข้อมูลที่ต้องการใน cache จะลดลง ถ้าบล็อกมีขนาดใหญ่ขึ้นไปอีกโอกาสที่จะใช้ข้อมูลในบล็อกใหม่ที่พึ่งจะอ่านขึ้นมานั้นกลับน้อยกว่าโอกาสที่จะใช้ข้อมูลจากบล็อกเก่าที่ถูกแทนที่ไปแล้ว ซึ่งสรุปความสัมพันธ์ได้ว่า

- บล็อกที่มีขนาดใหญ่จะลดจำนวนบล็อกที่จะสามารถอ่านเข้ามาเก็บไว้ใน cache เนื่องจากบล็อกใหม่ที่อ่านเข้ามาจะถูกบันทึกแทนที่บล็อกข้อมูลเก่าที่มีอยู่ก่อนหน้า ดังนั้นยิ่งมีจำนวนบล็อกน้อยลงก็จะยิ่งทำให้บล็อกเก่าถูกลบทิ้งเร็วขึ้น

- ถ้าบล็อกมีขนาดใหญ่ขึ้น ข้อมูลที่อ่านเข้ามาเพิ่มขึ้นนั้นจะยิ่งอยู่ในตำแหน่งที่ห่างไกลจากตำแหน่งข้อมูลที่กำลังถูกเรียกใช้งานมากยิ่งขึ้น ซึ่งเป็นการขัดต่อกฎการอ้างอิงพื้นที่ใกล้เคียง ดังนั้นจึงมีโอกาสที่จะถูกนำมาใช้งานน้อยลง

ความสัมพันธ์ระหว่างขนาดของบล็อกและอัตราการค้นพบข้อมูลที่ต้องการใน cache (hit ratio) นั้นมีความซับซ้อนมาก ซึ่งขึ้นอยู่กับลักษณะการทำงานของแต่ละโปรแกรมเป็นหลักและไม่มีผู้ใดสามารถให้นิยามของความพอดีได้เลย อย่างไรก็ตาม บล็อกขนาด 8 ถึง 32 ไบต์นั้นเป็นค่าที่ดีสำหรับการงานทั่วไป และบล็อกขนาด 64 ถึง 128 ไบต์นั้นเหมาะสมกับเครื่องคอมพิวเตอร์ความสามารถสูงสำหรับใช้ในงานวิจัยชั้นสูง

6. จำนวนของ cache

ในตอนที่ cache ถูกนำมาใช้งานเป็นครั้งแรก ระบบคอมพิวเตอร์ทั่วไปจะมี cache เพียงระดับเดียว เมื่อเร็วๆ นี้การใช้ cache หลายระดับได้รับความนิยมนำมาใช้งานทั่วไป การออกแบบ

cache หลายระดับมีมุมมองที่สำคัญ 2 ด้านคือ จำนวนระดับของ cache ที่นำมาใช้ และการใช้ cache แบบรวมเป็นหนึ่งเดียวหรือแบบแยกส่วน

6.1 cache แบบหลายระดับ

เทคโนโลยีที่ก้าวหน้ามากขึ้นช่วยให้สามารถใส่ cache เข้าไปเป็นส่วนหนึ่งของชิพโปรเซสเซอร์ได้ ซึ่งเรียกว่า on-chip cache เมื่อเปรียบเทียบกับ cache ภายนอกซึ่งจะต้องติดต่อผ่านบัส on-chip cache ช่วยลดงานของโปรเซสเซอร์ในการใช้บัสให้น้อยลงจึงช่วยเพิ่มความเร็วในการประมวลผลของโปรเซสเซอร์ซึ่งเป็นการเพิ่มประสิทธิภาพโดยรวมของระบบ ในกรณีที่ข้อมูลหรือคำสั่งที่ต้องการถูกตรวจพบว่าอยู่ใน on-chip cache ก็ไม่มีความจำเป็นจะต้องใช้บริการของบัส และเนื่องจากระยะทางที่สั้นกว่าจึงทำให้การส่งข้อมูลไปยังโปรเซสเซอร์ทำได้เร็วกว่ามากแม้ว่าจะเปรียบเทียบกับการใช้งานผ่านบัสของ cache ภายนอกที่สามารถใช้ได้ในทันทีที่ต้องการก็ตาม (zero-wait state cycle) ยิ่งกว่านี้บัสที่ว่างงานก็สามารถไปให้บริการอุปกรณ์อื่นได้

ในระบบที่มี on-chip cache นั้นไม่ได้บังคับให้มีหรือไม่มี cache ภายนอกใช้งาน แต่โดยทั่วไปโดยเฉพาะเครื่องคอมพิวเตอร์ในยุคปัจจุบันมีการนำ cache ทั้งสองชนิดมาใช้งานร่วมกันและเรียกโครงสร้างประเภทนี้ว่า cache 2 ระดับ โดย cache ระดับที่ 1 (L1) หมายถึง on-chip cache และระดับที่ 2 (L2) หมายถึง cache ภายนอก เหตุผลที่ต้องมี L2 cache เนื่องจากถ้าไม่มี L2 cache แล้วในทุกครั้งที่โปรเซสเซอร์ไม่สามารถหาข้อมูลได้จาก L1 cache ก็จะเกิดการอ้างอิงไปที่หน่วยความจำหลัก ผ่านบัสหลักของระบบคอมพิวเตอร์ ซึ่งโดยปกติจะทำให้เสียเวลานานมาก (เมื่อเปรียบเทียบกับความเร็วในการประมวลผลของซีพียู) และทำให้ทั้งระบบมีประสิทธิภาพการทำงานลดลง แต่ถ้ามีการนำ L2 cache มาใช้ข้อมูลส่วนหนึ่งจะถูกพบที่นี่ ซึ่งช่วยให้การเข้าถึงข้อมูลนั้นรวดเร็วมากกว่าการเข้าถึงหน่วยความจำหลักโดยตรง โดยเฉพาะถ้าชิพที่นำมาสร้าง L2 cache เป็นแบบ SRAM ที่มีความเร็วเท่ากับบัสแล้ว ข้อมูลในนี้จะถูกนำส่งโปรเซสเซอร์ได้โดยไม่มีการรอจังหวะสัญญาณนาฬิกา (zero-wait state transaction) เกิดขึ้นเลยซึ่งเป็น L2 cache ชนิดที่เร็วที่สุด

การออกแบบ cache 2 ระดับสำหรับใช้งานมีข้อสังเกต 2 ประการ ประการแรก สำหรับ L2 cache มักจะไม่เชื่อมโยงเข้ากับบัสหลัก แต่จะมีบัสแยกต่างหาก (local bus) ที่เชื่อมโยง L2 cache เข้ากับโปรเซสเซอร์โดยตรง ทั้งนี้เพื่อเป็นการลดปริมาณข้อมูลที่จะต้องส่งผ่านบัสหลัก ประการที่สอง เนื่องจากเทคโนโลยีใหม่ๆ ช่วยลดขนาดของอุปกรณ์ภายในโปรเซสเซอร์ให้เล็กลงจึงเริ่มมีความพยายามที่จะบรรจุ L2 cache เข้าไปไว้ในชิพโปรเซสเซอร์เช่นเดียวกับ L1 cache ซึ่งจะช่วยเพิ่มประสิทธิภาพให้สูงขึ้นไปอีก

แนวโน้มของประโยชน์ที่จะได้รับจากการใช้ L2 cache ขึ้นอยู่กับอัตราการค้นพบข้อมูล (hit ratio) ใน L1 และ L2 cache จากการศึกษาหลายแห่งพบว่า L2 cache นั้นช่วยเพิ่มประสิทธิภาพในการทำงานจริง อย่างไรก็ตาม การใช้ cache 2 ระดับทำให้การออกแบบ cache มีความซับซ้อนเพิ่มขึ้นเป็นอย่างมาก ทั้งในเรื่องขนาด อัลกอริทึมการแทนที่ และนโยบายในการบันทึกข้อมูล

6.2 cache แบบ Unified และแบบ Split

เมื่อ on-chip cache เริ่มถูกนำมาใช้งาน การออกแบบ cache นิยมใช้ cache ที่สามารถเก็บได้ทั้งข้อมูลและคำสั่งปนกัน (เรียกว่า unified cache) ต่อมาจึงได้มีแนวความคิดในการแยก cache สำหรับเก็บคำสั่งและสำหรับเก็บข้อมูลออกจากกัน (เรียกว่า split cache)

Unified cache มีข้อดี 2 ประการคือ

- สำหรับ cache จำนวนหนึ่ง unified cache มีอัตราการค้นพบข้อมูลที่ต้องการมากกว่า split cache เนื่องจากสามารถรักษาสมดุลระหว่างการเรียกใช้คำสั่งและข้อมูลได้โดยอัตโนมัติ นั่นคือ ถ้าการอ้างอิงมีแนวโน้มในการเรียกใช้คำสั่งมากกว่าข้อมูล cache ก็จะถูกบันทึกไว้ด้วยคำสั่งในปริมาณที่มากกว่าข้อมูล และในทางกลับกัน cache ก็จะถูกบันทึกไว้ด้วยคำสั่งในปริมาณที่มากกว่าข้อมูล และในทางกลับกัน cache จะมีข้อมูลมากกว่าคำสั่งถ้าโปรแกรมที่กำลังทำงานอยู่นั้นอ้างอิงข้อมูลจำนวนมากโดยใช้คำสั่งซ้ำๆ
- การออกแบบและสร้างจะง่ายกว่าเพราะไม่ต้องแยกชนิดของ cache

แม้ว่า unified cache จะมีข้อดีมากกว่า แต่แนวโน้มก็เริ่มนิยมใช้ split cache มากขึ้นทุกขณะ โดยเฉพาะเครื่องคอมพิวเตอร์ประสิทธิภาพสูงประเภท superscalar machine เช่น Pentium และ PowerPC ซึ่งเน้นการประมวลผลแบบขนานและการดึงคำสั่งล่วงหน้าสำหรับคำสั่งที่คาดว่าจะถูกนำมาประมวลผลในลำดับต่อไป ข้อดีประการหนึ่งของการใช้ split cache คือวิธีการนี้ช่วยกำจัดความคับคั่งของ cache ระหว่างหน่วยที่ดึงและแปลความหมายคำสั่ง (instruction prefetcher) และหน่วยที่ทำการประมวลผล (execution unit) ซึ่งเป็นสิ่งที่สำคัญมากในการออกแบบคำสั่งแบบไปป์ไลน์ (pipelined instruction) กล่าวคือ โดยปกติโปรเซสเซอร์จะดึงคำสั่งเข้ามาล่วงหน้าและคอยเติมบัฟเฟอร์หรือไปป์ไลน์ด้วยคำสั่งที่จะประมวลผลให้ต่อเนื่องอยู่เสมอ ในกรณีที่ใช้ unified cache เมื่อการประมวลผลมีการอ้างอิงถึงการอ่านหรือบันทึกข้อมูลลงในหน่วยความจำ ความต้องการนี้จะถูกส่งไปที่ unified cache ถ้าในเวลาเดียวกันหน่วยที่ดึงและแปลความหมายคำสั่งก็ออกคำสั่งให้ unified cache อ่านคำสั่งต่อไปเข้ามา ก็จะถูกสั่งให้หยุดรอเป็นการชั่วคราว เพื่อรอให้การส่งข้อมูลไปยังหน่วยประมวลผลเสร็จสิ้นก่อนจึงจะส่งคำสั่งต่อไปให้ ปัญหาความคับคั่งในการใช้งานนี้ จะลดประสิทธิภาพการทำงานของไปป์ไลน์ลงไปอย่างมาก ซึ่ง split cache สามารถแก้ปัญหานี้ได้

เทคโนโลยีและการพัฒนา cache

1. Asynchronous Static RAM ถูกนำมาใช้เป็น cache ตั้งแต่ CPU 80386 มีความเร็วในการเข้าถึงข้อมูลรวดเร็วกว่า DRAM ที่นิยมใช้กันมีอยู่ 3 รุ่น แบ่งตามอัตราความเร็วในการเข้าถึงข้อมูล มี 20, 15, 12 นาโนวินาที (ns) แต่ไม่สามารถที่จะทำงานที่ความเร็วเท่าๆกับความเร็วของ CPU ทำให้ CPU ต้องเสียเวลาคอยข้อมูล (Wait State) จาก SRAM

2. Synchronous Burst Static RAM มีคุณสมบัติ Burst คือ ใช้สัญญาณนาฬิกาในการทำงานให้น้อยที่สุด โดยใช้สัญญาณนาฬิกาของตัว staticหน่วยความจำเอง ในการอ่านข้อมูลที่ต่อเนื่องกันสามารถรองรับความเร็วของสัญญาณนาฬิกาของระบบบัส 66 MHz ไม่ต้องเสียเวลาคอยข้อมูล (Wait State) ในท้องตลาดมีอยู่ 2 รุ่นคือ 8.5 และ 12 นาโนวินาที (ns) เหมาะกับเมนบอร์ดเพนเทียม

3. Pipelined Burst Static RAM (PB SRAM) มีหน่วยความจำรีจิสเตอร์พิเศษอยู่ในชิปหน่วยความจำ ช่วยให้สามารถที่จะโอนข้อมูลพร้อมๆ กับการระบุตำแหน่งและอ้างถึงข้อมูลในสแตติกหน่วยความจำตำแหน่งต่อไป สามารถรองรับความเร็วของสัญญาณนาฬิกาที่สูงสุด 133 MHz ในท้องตลาดมีอยู่ 2 รุ่นคือ 4.5 และ 8 นาโนวินาที (ns) เหมาะกับระบบใหม่ที่มีความเร็วของสัญญาณนาฬิกา 75 - 100 MHz

บรรณานุกรม

[1] สัลยุทธ์ สว่างวรรณ.สถาปัตยกรรมคอมพิวเตอร์.กรุงเทพฯ:เพียร์สัน เอ็ดดูเคชั่น อินโดไชน่า,2546

[2] Memory part 2: CPU caches**.[Online] Available :** http://lwn.net/Articles/252125/

[3] CPU cache**.[Online] Available :** http://en.wikipedia.org/wiki/CPU_cache

[4] Cache นั้นสำคัญไฉน**.[Online] Available :**
http://irrigation.rid.go.th/rid15/ppn/Knowledge/Cache/Cache.htm

[5] ความรู้เรื่อง Cache Memory**.[Online] Available :** http://www.se-ed.net/sanambin/h-cache.html

[6] CPU CACHE L1&L2**.[Online] Available :**
http://202.28.94.55/web/322461/2550/report/g5/index1_2.html