Cache Memories
Program Optimization

MADE BY: Zhong Zhineng & Song Yixin

Cache Memories

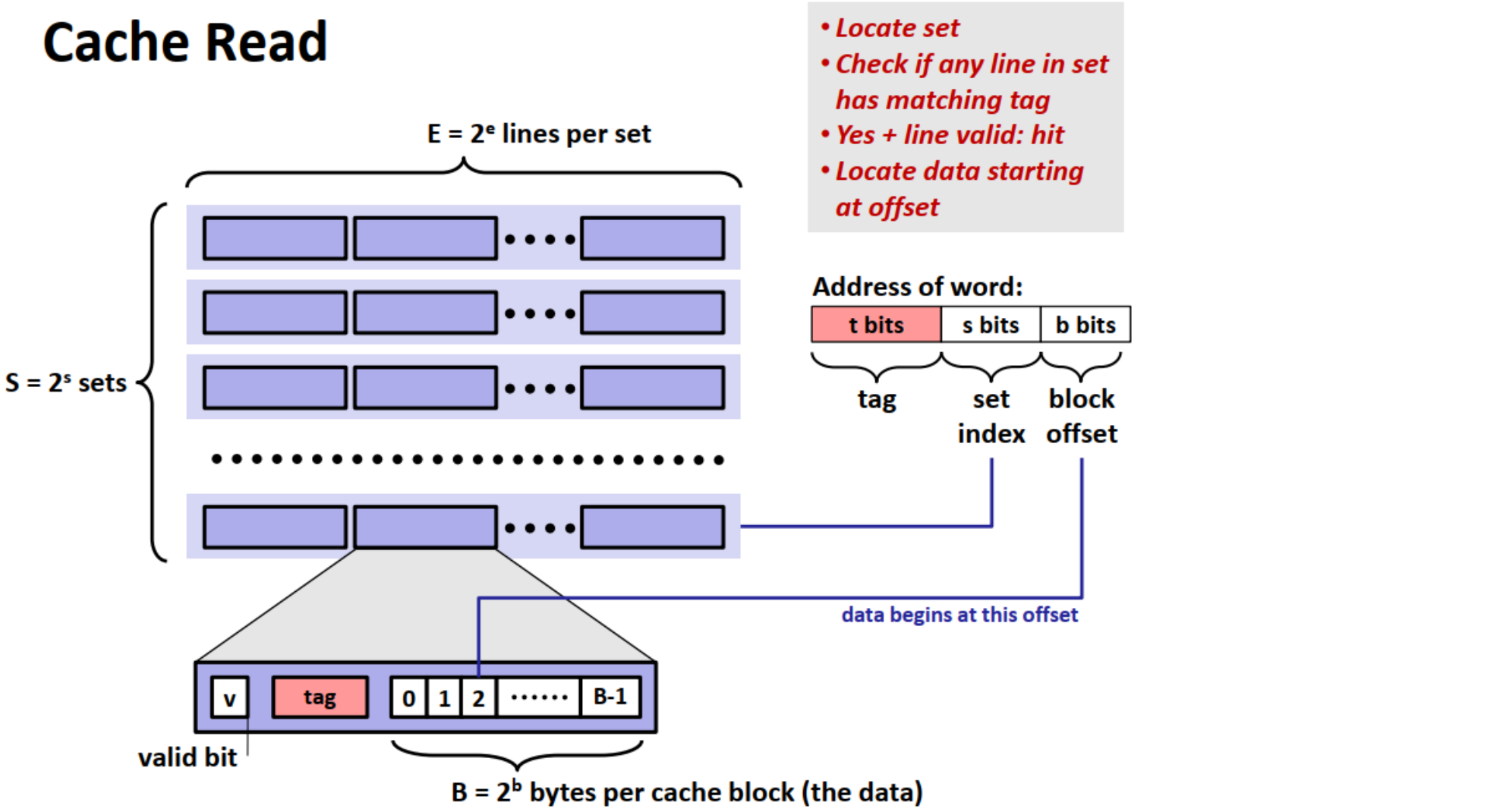
Cache Memories

- Small and fast SRAM memories
- Keep frequently accessed blocks of main memory
- Cache hit
- Cache miss: three types of miss

Cache Structure

- M, S, E, B, m, s
- S cache sets, E cache lines, B blocks
- t = m-(b+s) tag bits
- Valid bit
- C = S×E×B
- Address of word: tag + set index + block offset
- Bijection from middle s set index bits to S cache sets
- Bijection from low b block offset bits to B blocks
- 2^t address of a cache set => t tag bits

Cache Structure



Cache Classification

- Direct Mapped Cache (E = 1)
  - Not match: old line being replaced directly
  - Thrash
- Associative Cache (1 <E < C/B)
  - Not match: one line in the set is selected for eviction and replacement
  - Many replacement policies: Random, LRU, LFU, …
- Fully Associative Cache (S = 1, E = C/B)
  - Search for many matching tags in parallel => expensive and small
  - TLB

Cache Write

- Write hit
  - Write-through
  - Write-back
- Write miss
  - Write allocate
  - No write allocate
- Write through + No write allocate / Write back + Write allocate

Cache Friendly Code

- Make use of locality
- Make the most frequent the fast
- Decrease cache miss within loops
- Loop variable orders

Cache Mountain

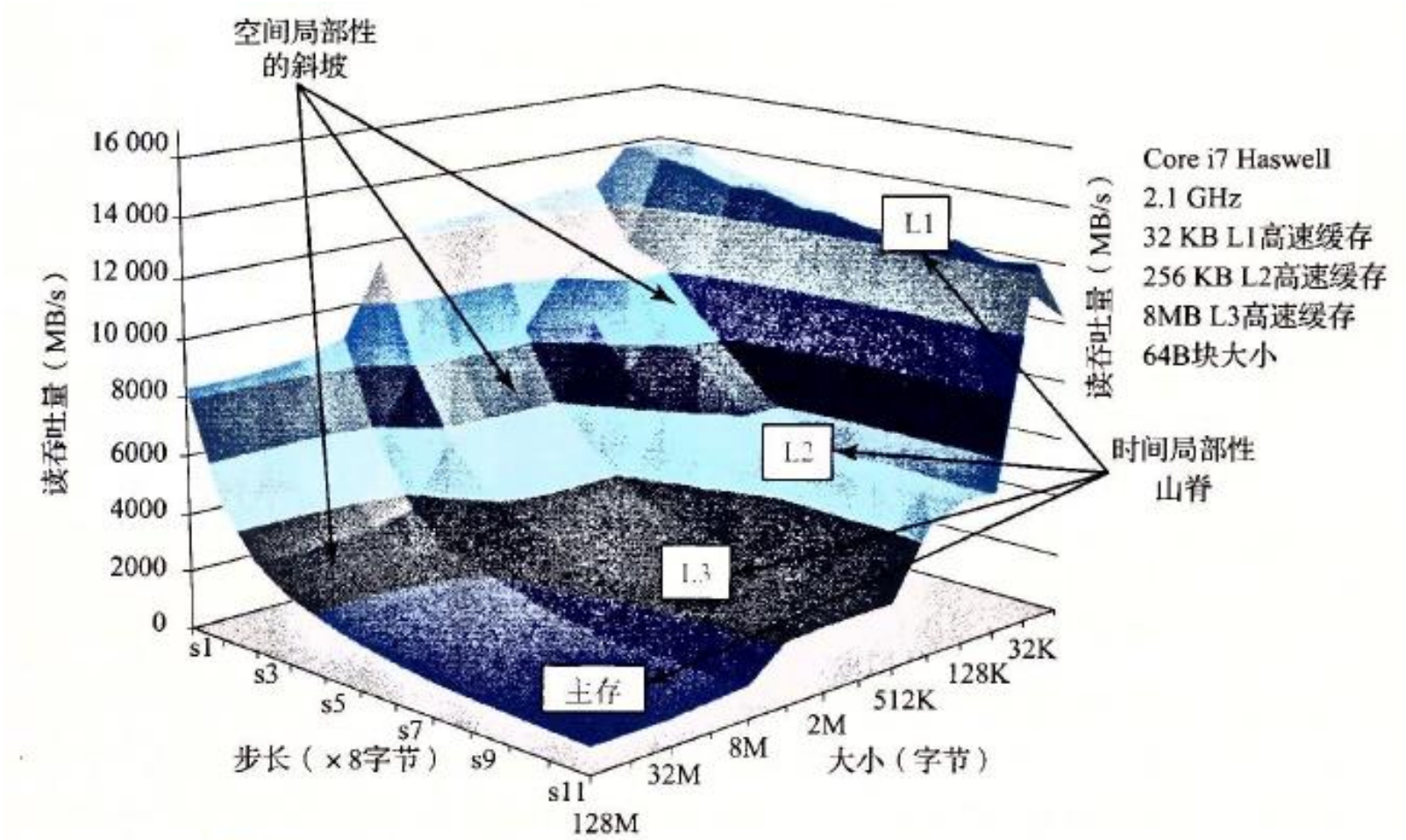


图 6-41　存储器山。展示了读吞吐量，它是时间和空间局部性的函数

Program Optimization

Compiler Optimization

- Some can be done by compiler
  - Code moving, such as moving the repeating calc inside the loop outside.
  - Use shift and add/sub to do mult/div
  - Expression reuse

- Some cannot be done by compiler (must be conservative)
  - Procedure may have side-effects
  - Pointer may have aliases

Simple Out-of-order Processor

- CPI may be less than 1
- In-order issue
- Out-of-order execution
- In-order commit

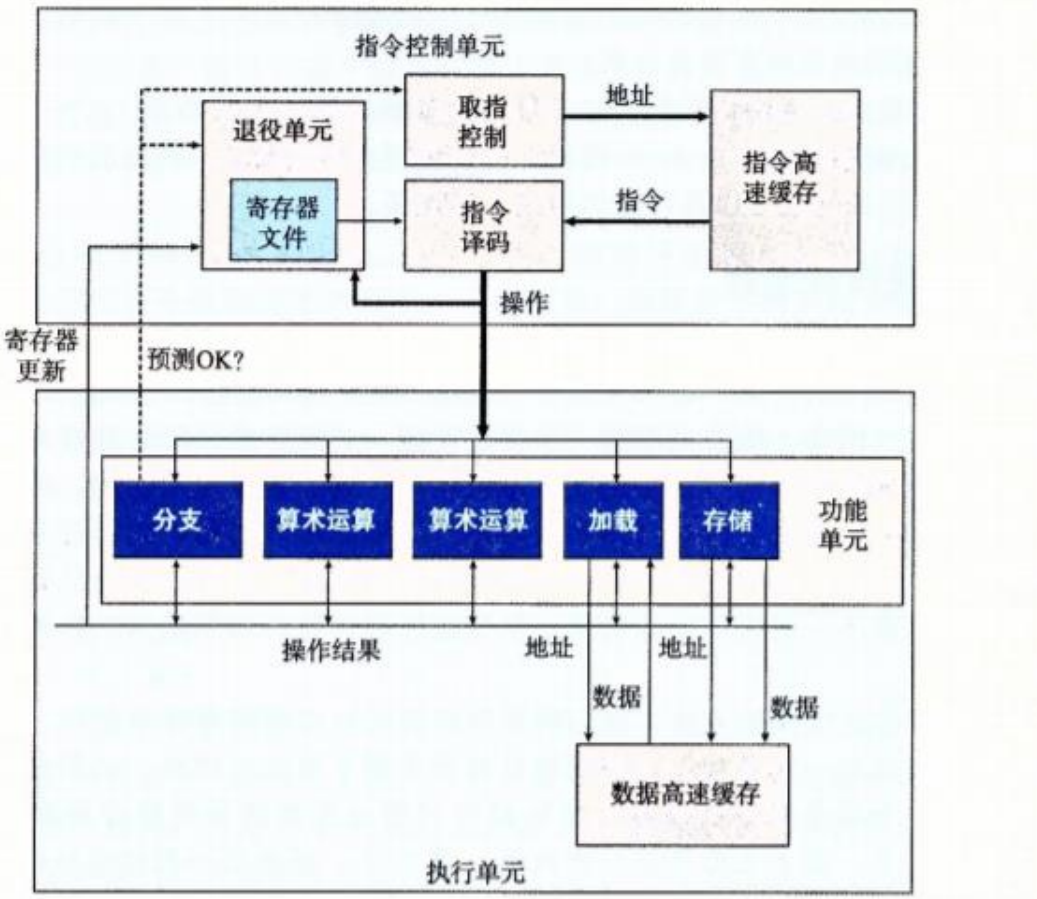


图 5-11 一个乱序处理器的框图。指令控制单元负责从内存中读出指令，并产生一系列基本操作。然后执行单元完成这些操作，以及指出分支预测是否正确

Modern CPU

- Out-of-order execution
- Multiple issue
- Dynamic Scheduling
- Speculative execution
- Register renaming

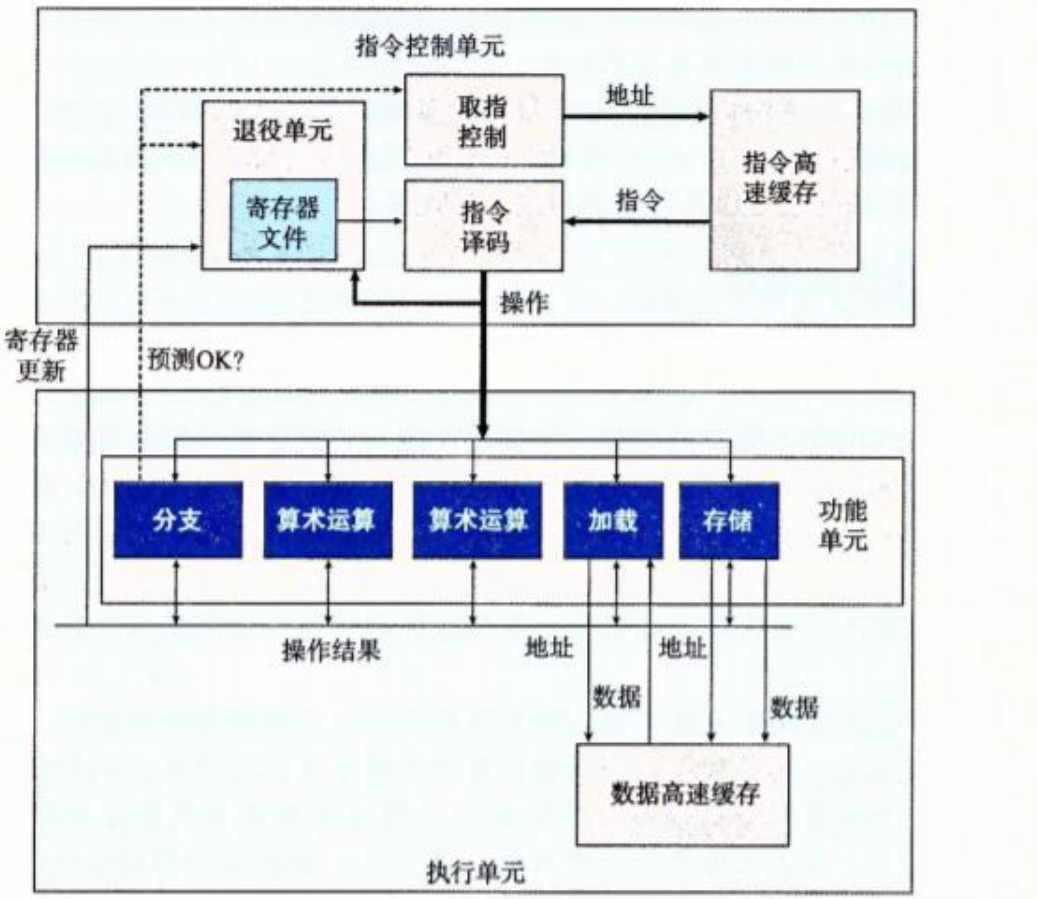


图 5-11 一个乱序处理器的框图。指令控制单元负责从内存中读出指令，并产生一系列基本操作。然后执行单元完成这些操作，以及指出分支预测是否正确

Critical Path

- Critical path: lower bound
- Number of function parts
- Data passing limitation

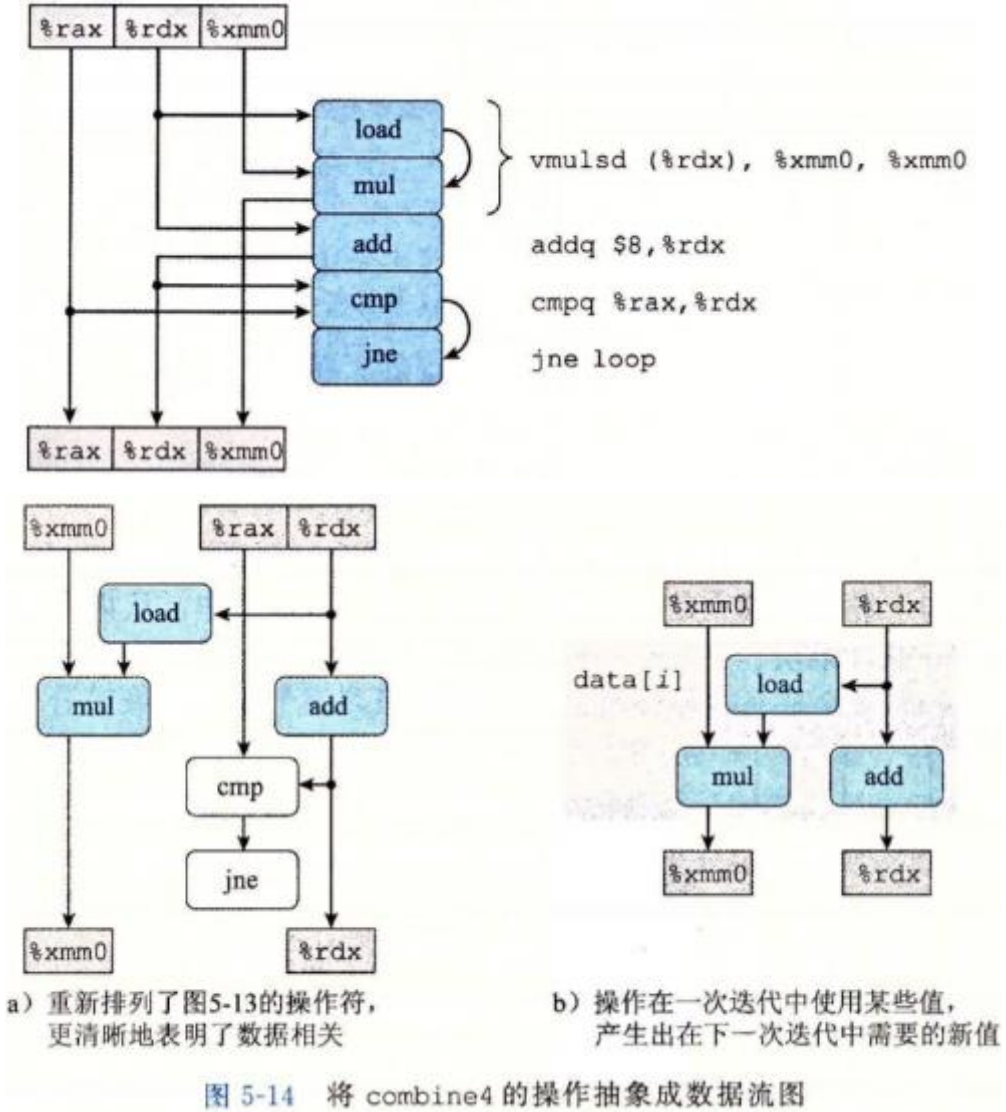
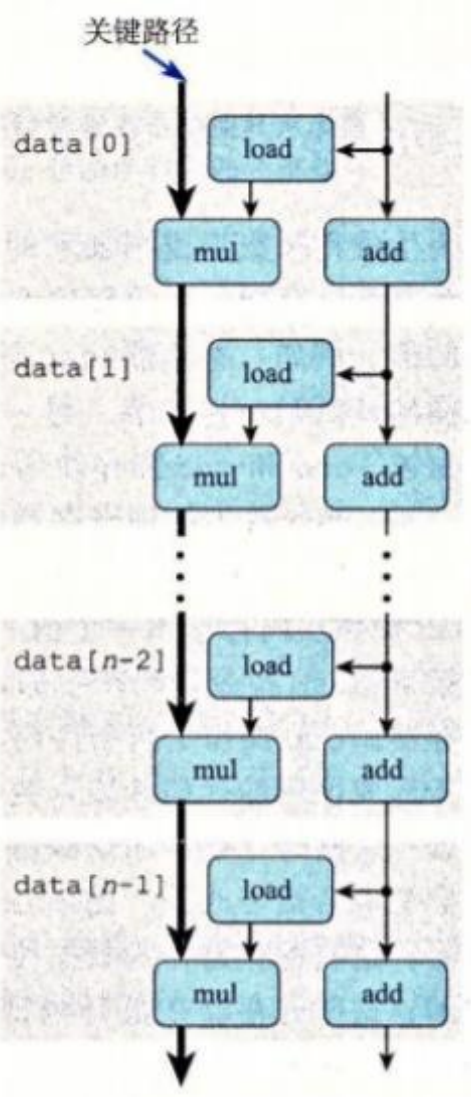


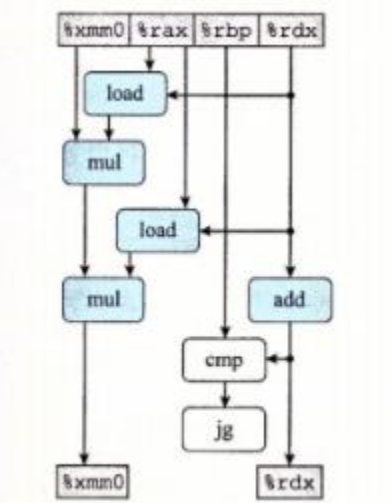
图 5-14　将 combine4 的操作抽象成数据流图



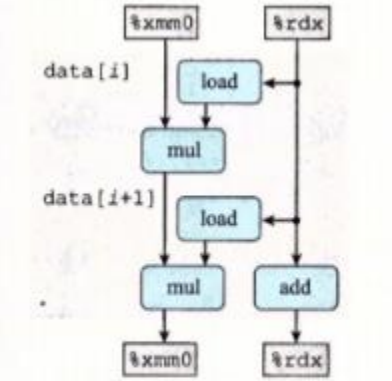
Loop Unrolling: Improve Parallelism

```
1   /* 2 x 1 loop unrolling */
2   void combine5(vec_ptr v, data_t *dest)
3   {
4       long i;
5       long length = vec_length(v);
6       long limit = length-1;
7       data_t *data = get_vec_start(v);
8       data_t acc = IDENT;
9
10      /* Combine 2 elements at a time */
11      for (i = 0; i < limit; i+=2) {
12          acc = (acc OP data[i]) OP data[i+1];
13      }
14
15      /* Finish any remaining elements */
16      for (; i < length; i++) {
17          acc = acc OP data[i];
18      }
19      *dest = acc;
20  }
```

函数	方法	整数		浮点数	
		+	*	+	*
combine4	无展开	1.27	3.01	3.01	5.01
combine5	2×1 展开	1.01	3.01	3.01	5.01
	3×1 展开	1.01	3.01	3.01	5.01
延迟界限		1.00	3.00	3.00	5.00
吞吐量界限		0.50	1.00	1.00	0.50

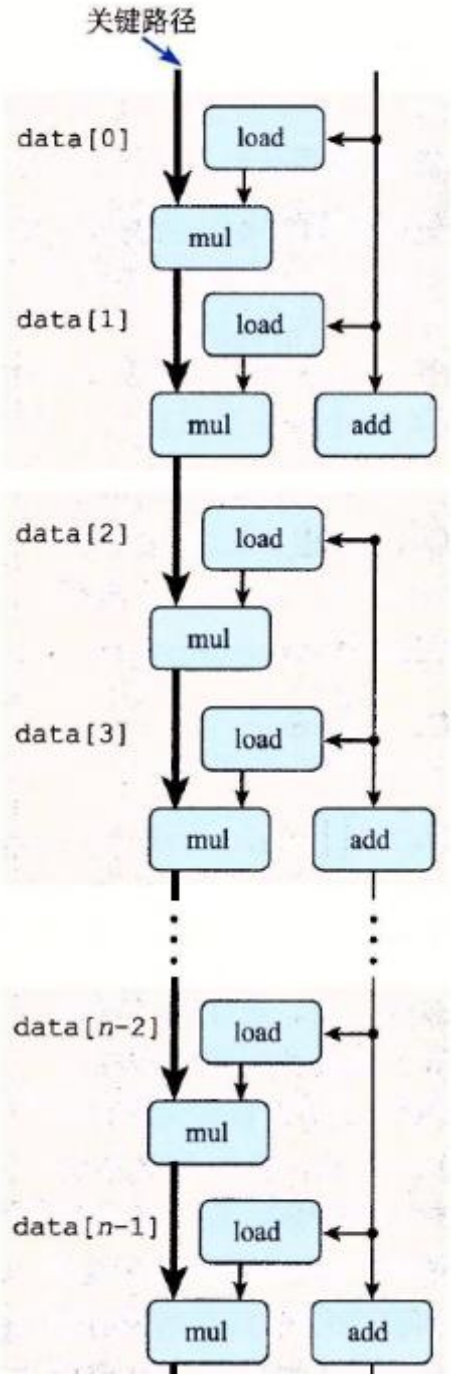


a) 重新排列、简化和抽象图5-18的表示，给出连续迭代之间的数据相关



b) 每次迭代必须顺序地执行两个乘法

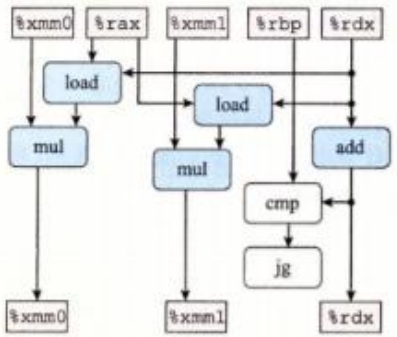
图 5-19 将 combine5 的操作抽象成数据流图



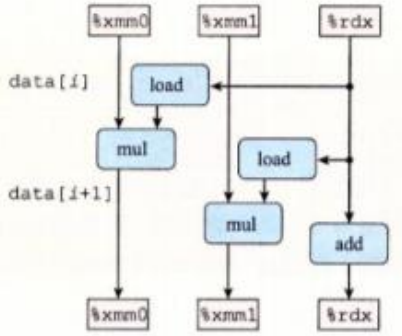
Separate Accumulators

```
1   /* 2 x 2 loop unrolling */
2   void combine6(vec_ptr v, data_t *dest)
3   {
4       long i;
5       long length = vec_length(v);
6       long limit = length-1;
7       data_t *data = get_vec_start(v);
8       data_t acc0 = IDENT;
9       data_t acc1 = IDENT;
10
11      /* Combine 2 elements at a time */
12      for (i = 0; i < limit; i+=2) {
13          acc0 = acc0 OP data[i];
14          acc1 = acc1 OP data[i+1];
15      }
16
17      /* Finish any remaining elements */
18      for (; i < length; i++) {
19          acc0 = acc0 OP data[i];
20      }
21      *dest = acc0 OP acc1;
22  }
```

图 5-21　运用 2×2 循环展开。通过维护多个累积变量，这种方法利用了多个功能单元以及它们的流水线能力



a）重新排列、简化和抽象图5-22的表示，给出连续迭代之间的数据相关



b）两个mul操作之间没有相关

图 5-23　将 combine6 的运算抽象成数据流图

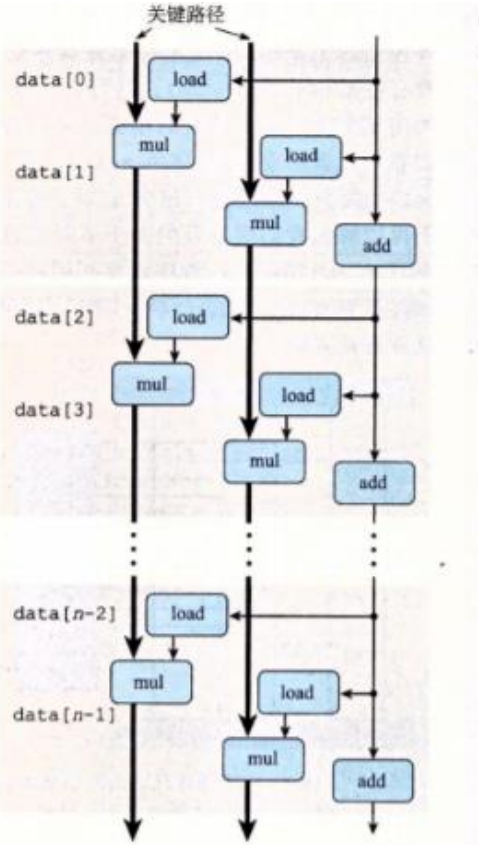


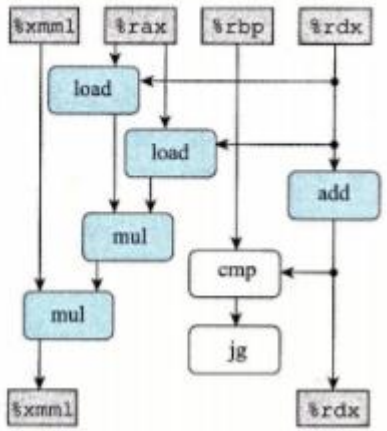
图 5-24　combine6 对一个长度为 n 的向量进行操作的数据流表示。现在有两条关键路径，每条关键路径包含 $n/2$ 个操作

函数	方法	整数		浮点数	
		+	*	+	*
combine4	在临时变量中累积	1.27	3.01	3.01	5.01
combine5	2×1 展开	1.01	3.01	3.01	5.01
combine6	2×2 展开	0.81	1.51	1.51	2.51
延迟界限		1.00	3.00	3.00	5.00
吞吐量界限		0.50	1.00	1.00	0.50

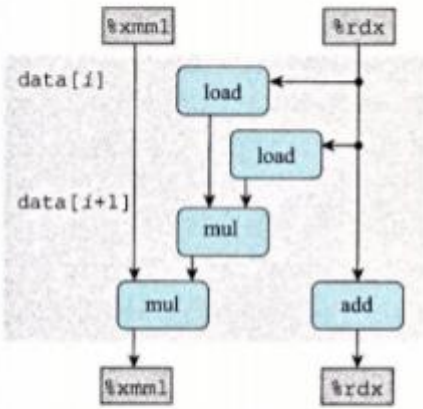
Reassociation

```
1   /* 2 x 1a loop unrolling */
2   void combine7(vec_ptr v, data_t *dest)
3   {
4       long i;
5       long length = vec_length(v);
6       long limit = length-1;
7       data_t *data = get_vec_start(v);
8       data_t acc = IDENT;
9
10      /* Combine 2 elements at a time */
11      for (i = 0; i < limit; i+=2) {
12          acc = acc OP (data[i] OP data[i+1]);
13      }
14
15      /* Finish any remaining elements */
16      for (; i < length; i++) {
17          acc = acc OP data[i];
18      }
19      *dest = acc;
20  }
```

函数	方法	整数		浮点数	
		+	*	+	*
combine4	累积在临时变量中	1.27	3.01	3.01	5.01
combine5	2×1 展开	1.01	3.01	3.01	5.01
combine6	2×2 展开	0.81	1.51	1.51	2.51
combine7	2×1a 展开	1.01	1.51	1.51	2.51
延迟界限		1.00	3.00	3.00	5.00
吞吐量界限		0.50	1.00	1.00	0.50



a）重新排列、简化和抽象图5-27的表示，给出连续迭代之间的数据相关



b）上面的mul操作让两个二向量元素相乘，而下面的mul操作将前面的结果乘以循环变量acc

图 5-28 将 combine7 的操作抽象成数据流图

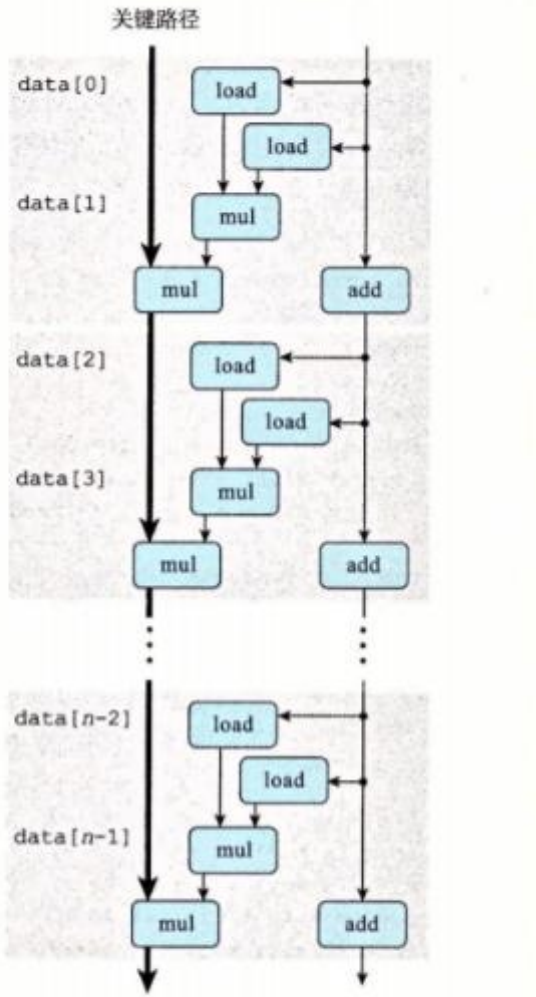


图 5-29 combine7 对一个长度为 n 的向量进行操作的数据流表示。我们只有一条关键路径，它只包含 $n/2$ 个操作

Limitations

- Register spilling: k = 20, use stack to store
- Floating operation does not have associative law, so changing operation orders may not apply.

Thanks for listening.