计算机组织与体系结构

第十八讲

计算机科学与技术学院

舒燕君

Recap

- Tomasulo算法
  - ✓基本部件（保留站、CDB、load和store缓冲）
  - ✓ 执行过程（流出、执行、结果写回）
  - ✓ 动态循环展开（动态存储器地址判别技术）
- 动态调度方法中的异常行为
  - ✓精确异常和不精确异常
- 控制相关的动态解决技术
  - ✓分支预测缓冲（BPB，1位和2位）
  - ✓分支目标缓冲（BTB）

7.3 控制相关的动态解决技术

7.3.1 分支预测缓冲

7.3.2 分支目标缓冲

7.3.3 基于硬件的前瞻执行

7.3.3 基于硬件的前瞻执行

前瞻执行（Speculation）的基本思想：

- 对分支指令的结果进行猜测，并假设这个猜测总是对的，然后按这个猜测结果继续取、流出和执行后续的指令。
- 执行指令的结果不是写回到寄存器或存储器，而是写入一个称为再定序缓冲器ROB（ReOrder Buffer）中 。等到相应的指令得到“确认”（commit）（即确实是应该执行的）之后，才将结果写入寄存器或存储器。

- 基于硬件的前瞻执行结合了3种思想：
  - ✓ 采用动态分支预测来选择后续执行的指令。
  - ✓ 在控制相关的结果尚未出来之前，前瞻地执行后续指令。
  - ✓ 对基本块采用动态调度。

- 对Tomasulo算法加以扩充，就可以支持前瞻执行。
  - ✓ 把Tomasulo算法的写结果和指令完成加以区分，分成两个不同的段：

    写结果，指令确认

- 允许指令乱序执行，但必须顺序确认。

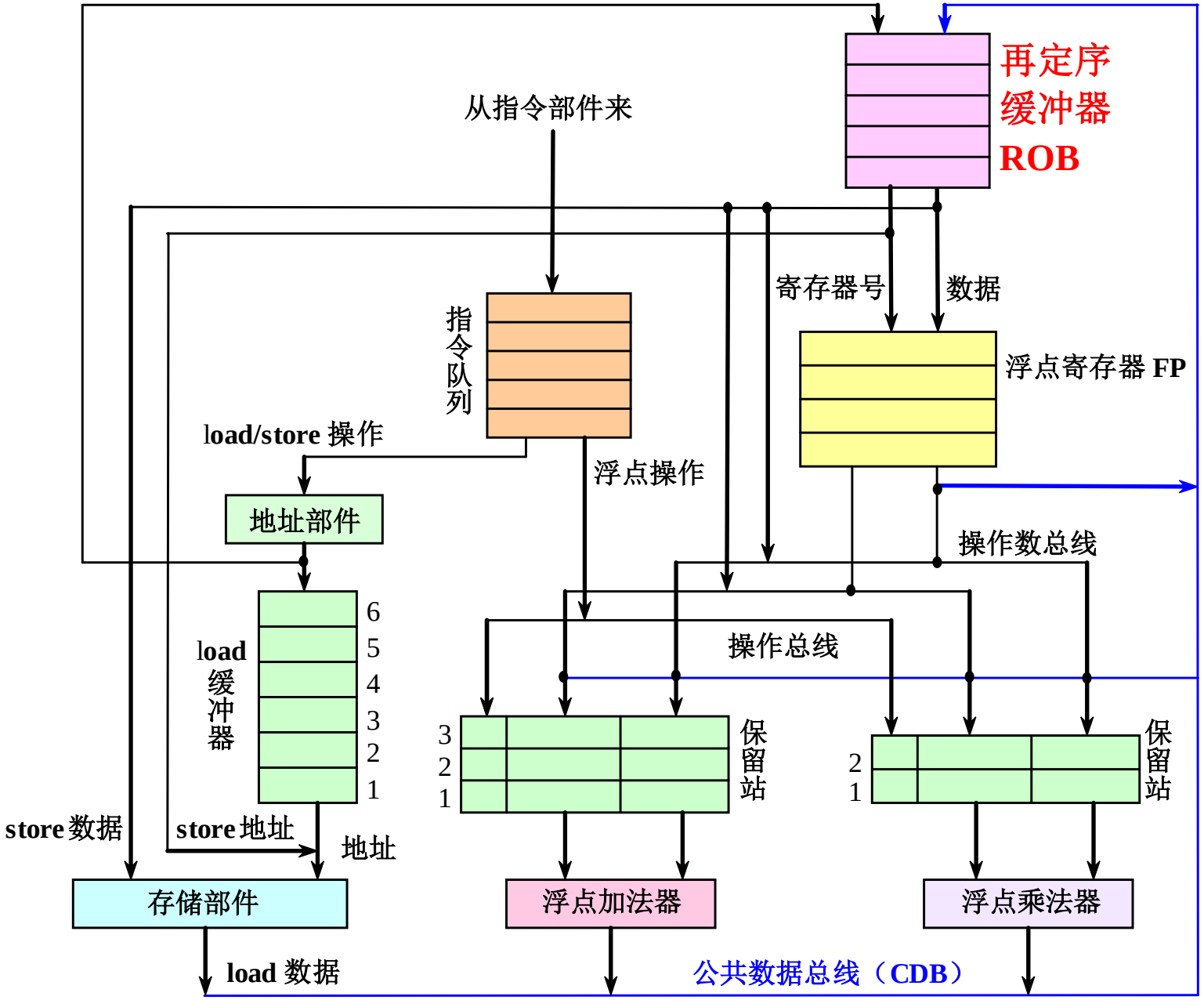
– 写结果段

- 把前瞻执行的结果写到**ROB**中；
- 通过**CDB**在指令之间传送结果，供需要用到这些结果的指令使用。

– 指令确认段

在分支指令的结果出来后，对相应指令的前瞻执行给予确认。

- 如果前面所做的猜测是对的，把在**ROB**中的结果写到寄存器或存储器。
- 如果发现前面对分支结果的猜测是错误的，那就不予以确认，刷新**ROB**，并从那条分支指令的另一条路径开始重新执行。


再定序
缓冲器
ROB
从指令部件来
指令队列
寄存器号
数据
浮点寄存器 FP
load/store 操作
浮点操作
地址部件
操作数总线
操作总线
load
缓冲器
6
5
4
3
2
1
保留站
3
2
1
保留站
2
1
store 数据
store 地址
地址
存储部件
浮点加法器
浮点乘法器
load 数据
公共数据总线（CDB）

– ROB中的每一项由以下4个字段组成：
  - 指令类型（instruction）
    指出该指令是分支指令、store指令或寄存器操作指令。
  - 状态（state）
    指出指令是否已经完成执行并且数据已就绪。
  - 目标地址（destination）
    给出指令执行结果应写入的目标寄存器号（load和ALU指令）或存储器单元的地址（store指令）。
  - 数据值（value）
    用来保存指令前瞻执行的结果，直到指令得到确认。

– 保留站RS中增加的域
  - 目标地址（destination）：对应的ROB编号

前瞻实例：Cycle 0

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Dest
0	Add1	No						
0	Add2	No						
0	Add3	No						
0	Mult1	No						
0	Mult2	No						

Reservation Stations

	Busy	Address
Load1		
Load2		
Load3		

Entry	Busy	Instruction	State	Destination	Value
1					
2					
3					
4					
5					
6					
7					
8					
9					
10					

Reorder Buffer

	F0	F2	F4	F6	F8	F10	F12	...	F30
Reorder #									
Busy	no	no	no	no	no	no	no		no

支持前瞻执行的 Tomasulo 算法的四阶段

1. Issue—get instruction from FP Op Queue
   - 如果RS和ROB有空闲单元就发射指令。如果寄存器或ROB中源操作数可用，就将其发送到RS，目的地址的ROB编号也发送给RS
2. Execution—operate on operands (EX)
   - 当操作数就绪后，开始执行。如果没有就绪，监测CDB，检查RAW相关
3. Write result—finish execution (WR)
   - 将运算结果通过CDB传送给所有等待结果的FU以及ROB单元，标识RS可用
4. Commit—update register with reorder result
   - ROB采用环形机制，按ROB中顺序，如果结果已有，就更新寄存器（或存储器），并将该指令从ROB表中删除
   - 预测失败或有中断时，清空ROB，从新的指令地址开始执行

假设执行阶段的周期数为：取数1个时钟周期，加减法2个时钟周期，乘法9个时钟周期，除法40个时钟周期。

```
LD      F6, 34(R2)
LD      F2, 45(R3)
MULTD   F0, F2, F4
SUBD    F8, F6, F2
DIVD    F10, F0, F6
ADDD    F6, F8, F2
```

前瞻实例：Cycle 1

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Dest
0	Add1	No						
0	Add2	No						
0	Add3	No						
0	Mult1	No						
0	Mult2	No						

Reservation Stations

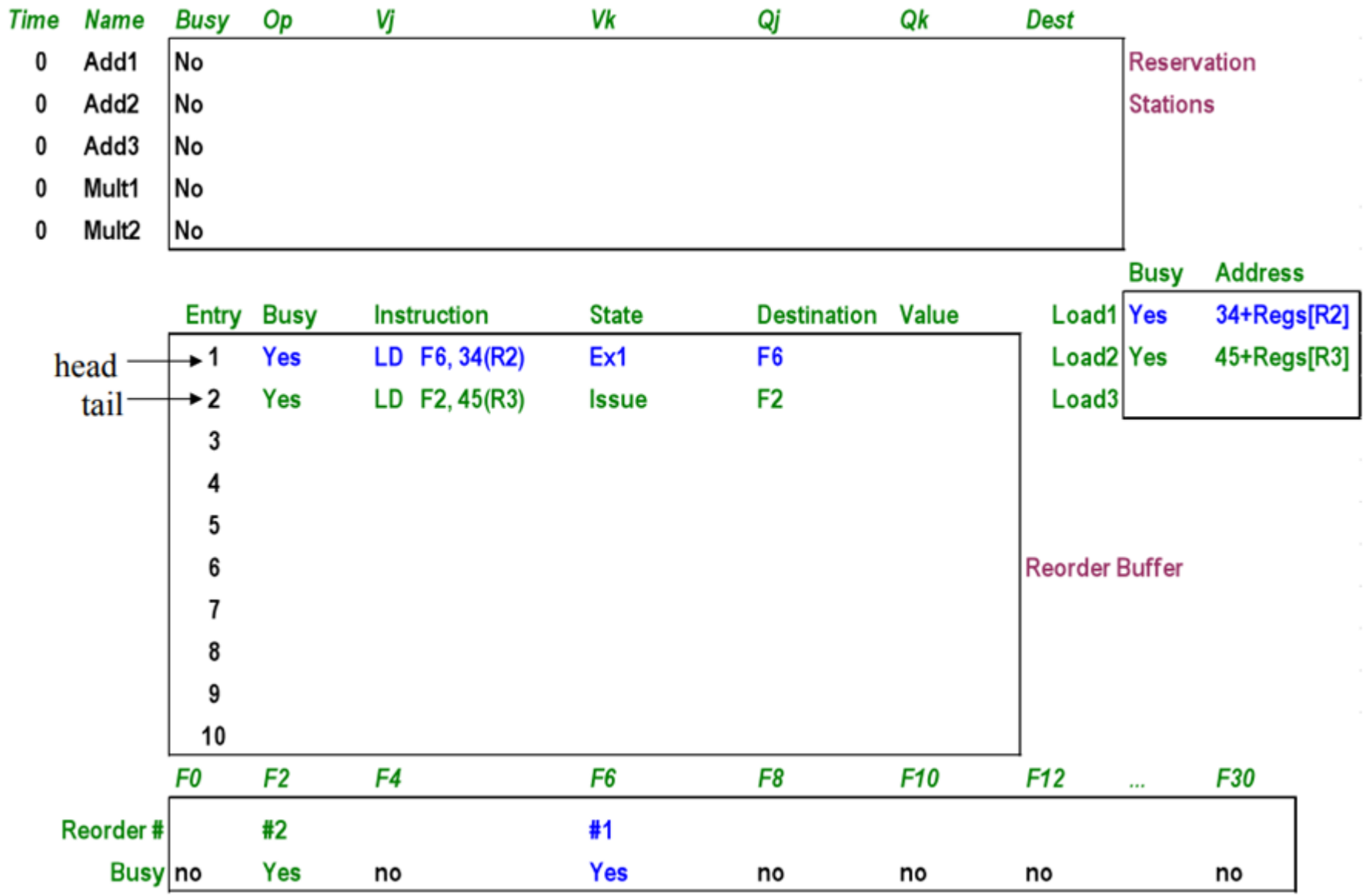
	Busy	Address
Load1	Yes	34+Regs[R2]
Load2		
Load3		

Entry	Busy	Instruction	State	Destination	Value
1	Yes	LD F6, 34(R2)	Issue	F6	
2					
3					
4					
5					
6					
7					
8					
9					
10					

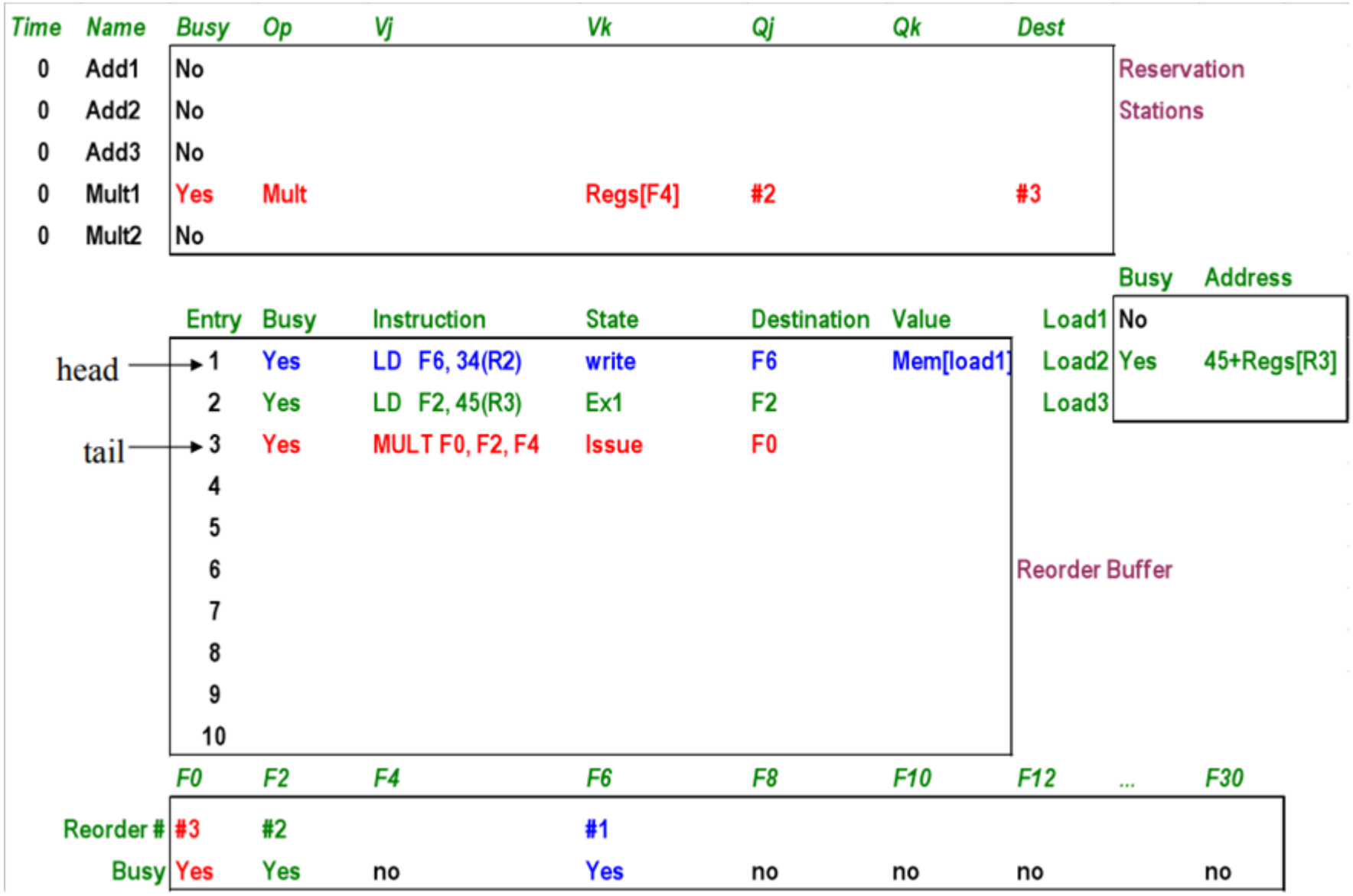
Reorder Buffer

	F0	F2	F4	F6	F8	F10	F12	...	F30
Reorder #				#1					
Busy	no	no	no	Yes	no	no	no		no

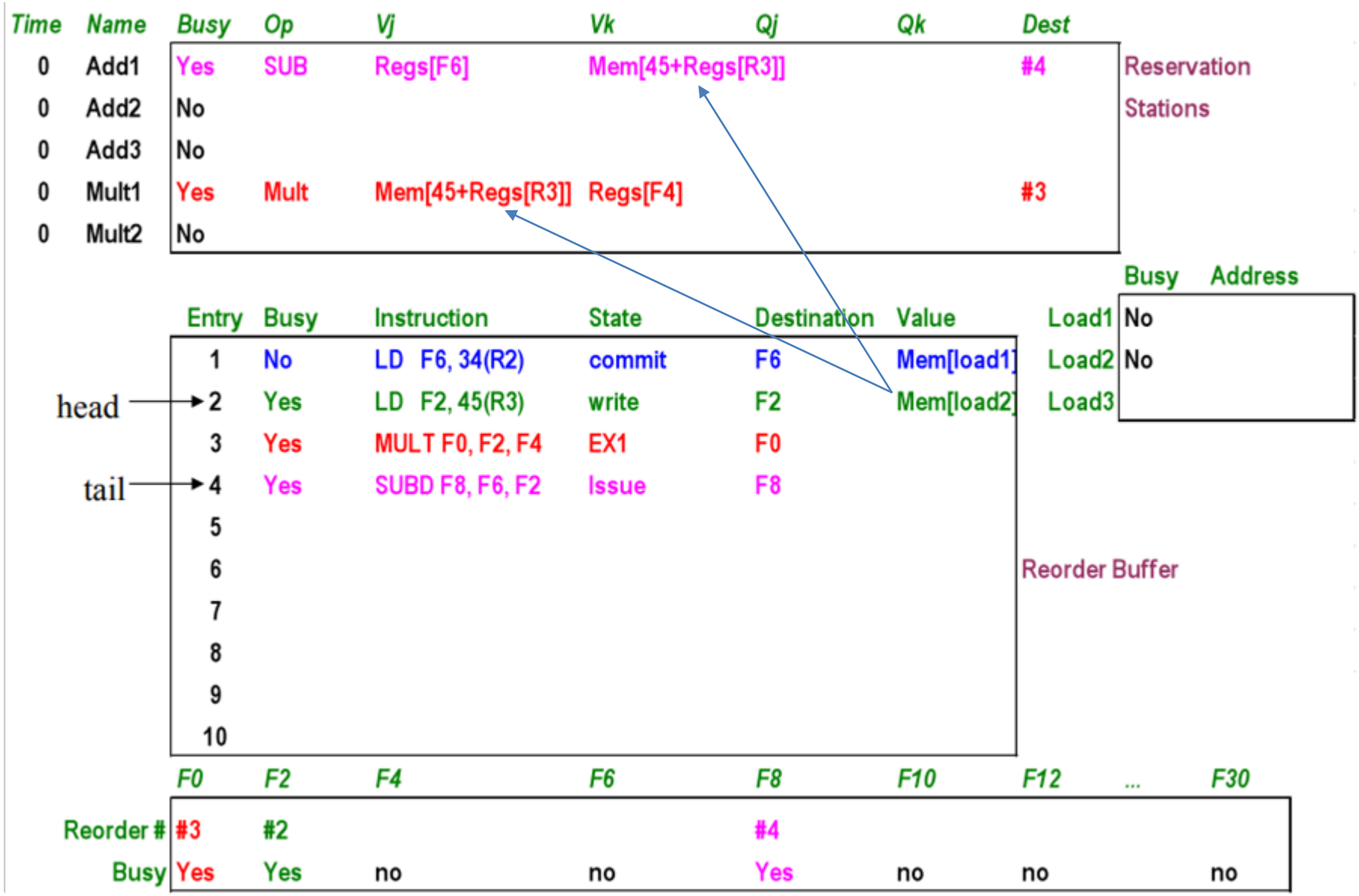
前瞻实例：Cycle 2


Time Name Busy Op Vj Vk Qj Qk Dest
0 Add1 No
0 Add2 No
0 Add3 No
0 Mult1 No
0 Mult2 No
Reservation Stations
Busy Address
Load1 Yes 34+Regs[R2]
Load2 Yes 45+Regs[R3]
Load3
Entry Busy Instruction State Destination Value
head → 1 Yes LD F6, 34(R2) Ex1 F6
tail → 2 Yes LD F2, 45(R3) Issue F2
3
4
5
6
7
8
9
10
Reorder Buffer
F0 F2 F4 F6 F8 F10 F12 ... F30
Reorder # #2 #1
Busy no Yes no Yes no no no no

前瞻实例：Cycle 3


Time Name Busy Op Vj Vk Qj Qk Dest
0 Add1 No
0 Add2 No
0 Add3 No
0 Mult1 Yes Mult Regs[F4] #2 #3
0 Mult2 No
Reservation Stations
Busy Address
Load1 No
Load2 Yes 45+Regs[R3]
Load3
Entry Busy Instruction State Destination Value
head 1 Yes LD F6, 34(R2) write F6 Mem[load1]
2 Yes LD F2, 45(R3) Ex1 F2
tail 3 Yes MULT F0, F2, F4 Issue F0
4
5
6
7
8
9
10
Reorder Buffer
F0 F2 F4 F6 F8 F10 F12 ... F30
Reorder # #3 #2 #1
Busy Yes Yes no Yes no no no no

前瞻实例：Cycle 4


Time Name Busy Op Vj Vk Qj Qk Dest
0 Add1 Yes SUB Regs[F6] Mem[45+Regs[R3]] #4
0 Add2 No
0 Add3 No
0 Mult1 Yes Mult Mem[45+Regs[R3]] Regs[F4] #3
0 Mult2 No
Reservation Stations
Busy Address
Load1 No
Load2 No
Load3
Entry Busy Instruction State Destination Value
1 No LD F6, 34(R2) commit F6 Mem[load1]
head 2 Yes LD F2, 45(R3) write F2 Mem[load2]
3 Yes MULT F0, F2, F4 EX1 F0
tail 4 Yes SUBD F8, F6, F2 Issue F8
5
6
7
8
9
10
Reorder Buffer
F0 F2 F4 F6 F8 F10 F12 ... F30
Reorder # #3 #2 #4
Busy Yes Yes no no Yes no no no

前瞻实例：Cycle 5

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Dest
0	Add1	Yes	SUB	Regs[F6]	Mem[45+Regs[R3]]			#4
0	Add2	No						
0	Add3	No						
0	Mult1	Yes	Mult	Mem[45+Regs[R3]]	Regs[F4]			#3
0	Mult2	Yes	DIV		Regs[F6]	#3		#5

Reservation Stations

	Busy	Address
Load1	No	
Load2	No	
Load3		

	Entry	Busy	Instruction	State	Destination	Value
	1	No	LD F6, 34(R2)	commit	F6	Mem[load1]
	2	No	LD F2, 45(R3)	commit	F2	Mem[load2]
head →	3	Yes	MULT F0, F2, F4	Ex2	F0	
	4	Yes	SUBD F8, F6, F2	Ex1	F8	
tail →	5	Yes	DIVD F10, F0, F6	Issue	F10	
	6					
	7					
	8					
	9					
	10					

Reorder Buffer

	F0	F2	F4	F6	F8	F10	F12	...	F30
Reorder #	#3				#4	#5			
Busy	Yes	no	no	no	Yes	Yes	no		no

前瞻实例：Cycle 6

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Dest
0	Add1	Yes	SUB	Regs[F6]	Mem[45+Regs[R3]]			#4
0	Add2	Yes	Add		Regs[F2]	#4		#6
0	Add3	No						
0	Mult1	Yes	Mult	Mem[45+Regs[R3]]	Regs[F4]			#3
0	Mult2	Yes	DIV		Regs[F6]	#3		#5

Reservation Stations

	Busy	Address
Load1	No	
Load2	No	
Load3		

	Entry	Busy	Instruction	State	Destination	Value
	1	No	LD F6, 34(R2)	commit	F6	Mem[load1]
	2	No	LD F2, 45(R3)	commit	F2	Mem[load2]
head →	3	Yes	MULT F0, F2, F4	Ex3	F0	
	4	Yes	SUBD F8, F6, F2	Ex2	F8	
	5	Yes	DIVD F10, F0, F6	Issue	F10	
tail →	6	Yes	ADDD F6, F8, F2	Issue	F6	
	7					
	8					
	9					
	10					

Reorder Buffer

	F0	F2	F4	F6	F8	F10	F12	...	F30
Reorder #	#3			#6	#4	#5			
Busy	Yes	no	no	Yes	Yes	Yes	no		no

前瞻实例：Cycle 7

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Dest
0	Add1	No						
0	Add2	Yes	Add	#4	Regs[F2]			#6
0	Add3	No						
0	Mult1	Yes	Mult	Mem[45+Regs[R3]]	Regs[F4]			#3
0	Mult2	Yes	DIV		Regs[F6]	#3		#5

Reservation Stations

	Busy	Address
Load1	No	
Load2	No	
Load3		

	Entry	Busy	Instruction	State	Destination	Value
	1	No	LD F6, 34(R2)	commit	F6	Mem[load1]
	2	No	LD F2, 45(R3)	commit	F2	Mem[load2]
head →	3	Yes	MULT F0, F2, F4	Ex4	F0	
	4	Yes	SUBD F8, F6, F2	write	F8	F6 - #2
	5	Yes	DIVD F10, F0, F6	Issue	F10	
tail →	6	Yes	ADDD F6, F8, F2	EX1	F6	
	7					
	8					
	9					
	10					

Reorder Buffer

	F0	F2	F4	F6	F8	F10	F12	...	F30
Reorder #	#3			#6	#4	#5			
Busy	Yes	no	no	Yes	Yes	Yes	no		no

前瞻实例：Cycle 8

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Dest
0	Add1	No						
0	Add2	Yes	Add	#4	Regs[F2]			#6
0	Add3	No						
0	Mult1	Yes	Mult	Mem[45+Regs[R3]]	Regs[F4]			#3
0	Mult2	Yes	DIV		Regs[F6]	#3		#5

Reservation Stations

	Busy	Address
Load1	No	
Load2	No	
Load3		

	Entry	Busy	Instruction	State	Destination	Value
	1	No	LD F6, 34(R2)	commit	F6	Mem[load1]
	2	No	LD F2, 45(R3)	commit	F2	Mem[load2]
head →	3	Yes	MULT F0, F2, F4	Ex5	F0	
	4	Yes	SUBD F8, F6, F2	write	F8	F6 - #2
	5	Yes	DIVD F10, F0, F6	Issue	F10	
tail →	6	Yes	ADDD F6, F8, F2	Ex2	F6	
	7					
	8					
	9					
	10					

Reorder Buffer

	F0	F2	F4	F6	F8	F10	F12	...	F30
Reorder #	#3			#6	#4	#5			
Busy	Yes	no	no	Yes	Yes	Yes	no		no

前瞻实例：Cycle 9

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Dest
0	Add1	No						
0	Add2	Yes	Add	#4	Regs[F2]			#6
0	Add3	No						
0	Mult1	Yes	Mult	Mem[45+Regs[R3]]	Regs[F4]			#3
0	Mult2	Yes	DIV		Regs[F6]	#3		#5

Reservation Stations

	Busy	Address
Load1	No	
Load2	No	
Load3		

	Entry	Busy	Instruction	State	Destination	Value
	1	No	LD F6, 34(R2)	commit	F6	Mem[load1]
	2	No	LD F2, 45(R3)	commit	F2	Mem[load2]
head →	3	Yes	MULT F0, F2, F4	Ex6	F0	
	4	Yes	SUBD F8, F6, F2	write	F8	F6 - #2
	5	Yes	DIVD F10, F0, F6	Issue	F10	
tail →	6	Yes	ADDD F6, F8, F2	write	F6	#4 + F2
	7					
	8					
	9					
	10					

Reorder Buffer

	F0	F2	F4	F6	F8	F10	F12	...	F30
Reorder #	#3			#6	#4	#5			
Busy	Yes	no	no	Yes	Yes	Yes	no		no

前瞻实例：Cycle 10

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Dest
0	Add1	No						
0	Add2	No						
0	Add3	No						
0	Mult1	Yes	Mult	Mem[45+Regs[R3]]	Regs[F4]			#3
0	Mult2	Yes	DIV		Regs[F6]	#3		#5

Reservation Stations

	Busy	Address
Load1	No	
Load2	No	
Load3		

	Entry	Busy	Instruction	State	Destination	Value
	1	No	LD F6, 34(R2)	commit	F6	Mem[load1]
	2	No	LD F2, 45(R3)	commit	F2	Mem[load2]
head →	3	Yes	MULT F0, F2, F4	Ex7	F0	
	4	Yes	SUBD F8, F6, F2	write	F8	F6 - #2
	5	Yes	DIVD F10, F0, F6	Issue	F10	
tail →	6	Yes	ADDD F6, F8, F2	write	F6	#4 + F2
	7					
	8					
	9					
	10					

Reorder Buffer

	F0	F2	F4	F6	F8	F10	F12	...	F30
Reorder #	#3			#6	#4	#5			
Busy	Yes	no	no	Yes	Yes	Yes	no		no

前瞻实例：Cycle 11

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Dest
0	Add1	No						
0	Add2	No						
0	Add3	No						
0	Mult1	Yes	Mult	Mem[45+Regs[R3]]	Regs[F4]			#3
0	Mult2	Yes	DIV		Regs[F6]	#3		#5

Reservation Stations

	Busy	Address
Load1	No	
Load2	No	
Load3		

	Entry	Busy	Instruction	State	Destination	Value
	1	No	LD F6, 34(R2)	commit	F6	Mem[load1]
	2	No	LD F2, 45(R3)	commit	F2	Mem[load2]
head →	3	Yes	MULT F0, F2, F4	Ex8	F0	
	4	Yes	SUBD F8, F6, F2	write	F8	F6 - #2
	5	Yes	DIVD F10, F0, F6	Issue	F10	
tail →	6	Yes	ADDD F6, F8, F2	write	F6	#4 + F2
	7					
	8					
	9					
	10					

Reorder Buffer

	F0	F2	F4	F6	F8	F10	F12	...	F30
Reorder #	#3			#6	#4	#5			
Busy	Yes	no	no	Yes	Yes	Yes	no		no

前瞻实例：Cycle 12

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Dest
0	Add1	No						
0	Add2	No						
0	Add3	No						
0	Mult1	Yes	Mult	Mem[45+Regs[R3]]	Regs[F4]			#3
0	Mult2	Yes	DIV		Regs[F6]	#3		#5

Reservation Stations

	Busy	Address
Load1	No	
Load2	No	
Load3		

	Entry	Busy	Instruction	State	Destination	Value
	1	No	LD F6, 34(R2)	commit	F6	Mem[load1]
	2	No	LD F2, 45(R3)	commit	F2	Mem[load2]
head →	3	Yes	MULT F0, F2, F4	Ex9	F0	
	4	Yes	SUBD F8, F6, F2	write	F8	F6 - #2
	5	Yes	DIVD F10, F0, F6	Issue	F10	
tail →	6	Yes	ADDD F6, F8, F2	write	F6	#4 + F2
	7					
	8					
	9					
	10					

Reorder Buffer

	F0	F2	F4	F6	F8	F10	F12	...	F30
Reorder #	#3			#6	#4	#5			
Busy	Yes	no	no	Yes	Yes	Yes	no		no

前瞻实例：Cycle 13

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Dest
0	Add1	No						
0	Add2	No						
0	Add3	No						
0	Mult1	No						
0	Mult2	Yes	DIV	#2xRegs[F4]	Regs[F6]			#5

Reservation Stations

	Busy	Address
Load1	No	
Load2	No	
Load3		

	Entry	Busy	Instruction	State	Destination	Value
	1	No	LD F6, 34(R2)	commit	F6	Mem[load1]
	2	No	LD F2, 45(R3)	commit	F2	Mem[load2]
head →	3	Yes	MULT F0, F2, F4	write	F0	#2 x Regs[F4]
	4	Yes	SUBD F8, F6, F2	write	F8	F6 - #2
	5	Yes	DIVD F10, F0, F6	Ex1	F10	
tail →	6	Yes	ADDD F6, F8, F2	write	F6	#4 + F2
	7					
	8					
	9					
	10					

Reorder Buffer

	F0	F2	F4	F6	F8	F10	F12	...	F30
Reorder #	#3			#6	#4	#5			
Busy	Yes	no	no	Yes	Yes	Yes	no		no

前瞻实例：Cycle 14

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Dest
0	Add1	No						
0	Add2	No						
0	Add3	No						
0	Mult1	No						
0	Mult2	Yes	DIV	#2xRegs[F4]	Regs[F6]			#5

Reservation Stations

	Busy	Address
Load1	No	
Load2	No	
Load3		

	Entry	Busy	Instruction	State	Destination	Value
	1	No	LD F6, 34(R2)	commit	F6	Mem[load1]
	2	No	LD F2, 45(R3)	commit	F2	Mem[load2]
	3	No	MULT F0, F2, F4	commit	F0	#2 x Regs[F4]
head →	4	Yes	SUBD F8, F6, F2	write	F8	F6 - #2
	5	Yes	DIVD F10, F0, F6	Ex2	F10	
tail →	6	Yes	ADDD F6, F8, F2	write	F6	#4 + F2
	7					
	8					
	9					
	10					

Reorder Buffer

	F0	F2	F4	F6	F8	F10	F12	...	F30
Reorder #				#6	#4	#5			
Busy	No	no	no	Yes	Yes	Yes	no		no

前瞻实例：Cycle 15

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Dest
0	Add1	No						
0	Add2	No						
0	Add3	No						
0	Mult1	No						
0	Mult2	Yes	DIV	#2xRegs[F4]	Regs[F6]			#5

Reservation Stations

	Busy	Address
Load1	No	
Load2	No	
Load3		

	Entry	Busy	Instruction	State	Destination	Value
	1	No	LD F6, 34(R2)	commit	F6	Mem[load1]
	2	No	LD F2, 45(R3)	commit	F2	Mem[load2]
	3	No	MULT F0, F2, F4	commit	F0	#2 x Regs[F4]
	4	No	SUBD F8, F6, F2	commit	F8	F6 - #2
head →	5	Yes	DIVD F10, F0, F6	Ex3	F10	
tail →	6	Yes	ADDD F6, F8, F2	write	F6	#4 + F2
	7					
	8					
	9					
	10					

Reorder Buffer

	F0	F2	F4	F6	F8	F10	F12	...	F30
Reorder #				#6		#5			
Busy	no	no	no	Yes	no	Yes	no		no

前瞻实例：Cycle 16

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Dest
0	Add1	No						
0	Add2	No						
0	Add3	No						
0	Mult1	No						
0	Mult2	Yes	DIV	#2xRegs[F4]	Regs[F6]			#5

Reservation Stations

	Busy	Address
Load1	No	
Load2	No	
Load3		

	Entry	Busy	Instruction	State	Destination	Value
	1	No	LD F6, 34(R2)	commit	F6	Mem[load1]
	2	No	LD F2, 45(R3)	commit	F2	Mem[load2]
	3	No	MULT F0, F2, F4	commit	F0	#2 x Regs[F4]
	4	No	SUBD F8, F6, F2	commit	F8	F6 - #2
head →	5	Yes	DIVD F10, F0, F6	Ex4	F10	
tail →	6	Yes	ADDD F6, F8, F2	write	F6	#4 + F2
	7					
	8					
	9					
	10					

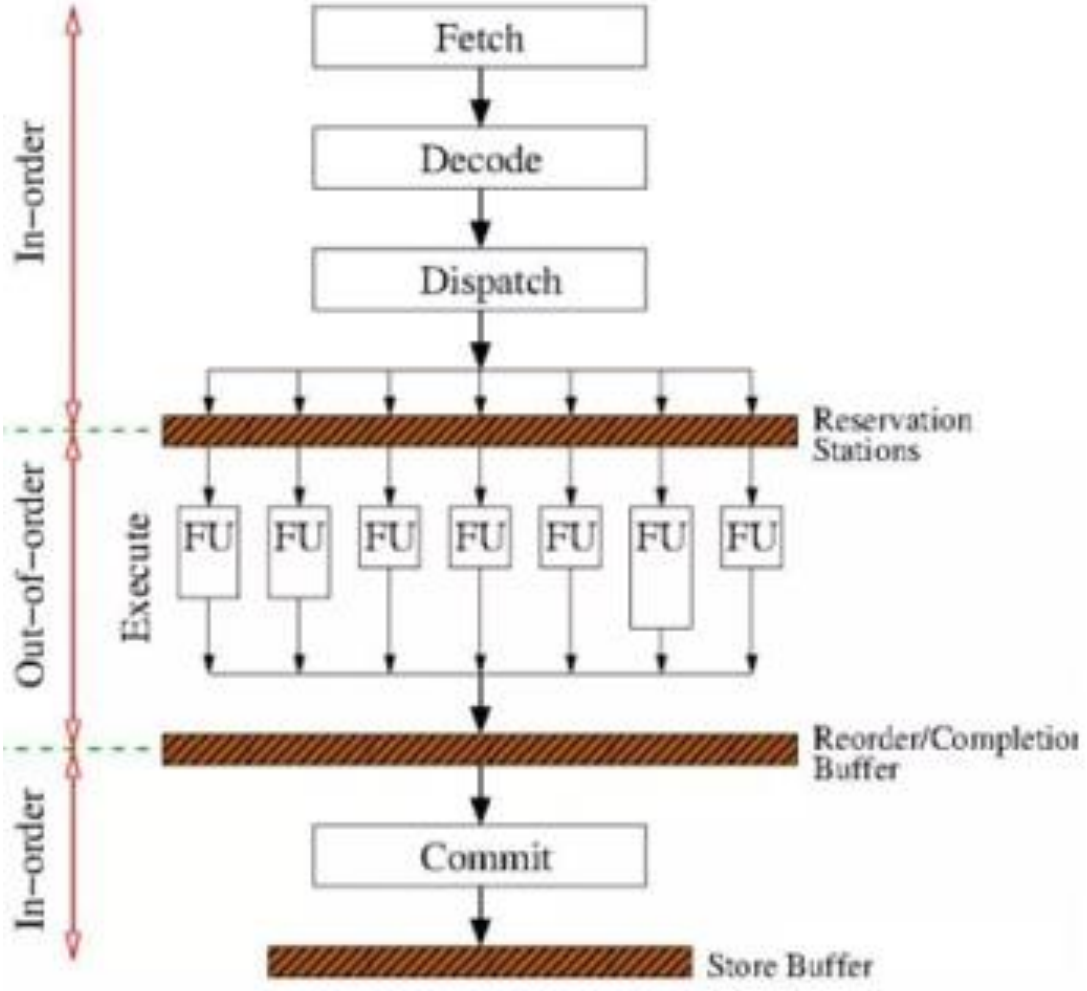
Reorder Buffer

	F0	F2	F4	F6	F8	F10	F12	...	F30
Reorder #				#6		#5			
Busy	no	no	no	Yes	no	Yes	no		no

Instruction	Issue	Exec Comp	Writeback	Commit
LD F6, 34(R2)	1	2	3	4
LD F2, 45(R3)	2	3	4	5
MULT F0, F2, F4	3	12	13	14
SUBD F8, F6, F2	4	6	7	15
DIVD F10, F0, F6	5	52	53	54
ADDD F6, F8, F2	6	8	9	55

- **In-order Issue/Commit**
- **Out-of-Order Execution/Writeback**

前瞻执行的流程


Fetch
Decode
Dispatch
In-order
Out-of-order
In-order
Execute
FU
FU
FU
FU
FU
FU
FU
Reservation Stations
Reorder/Completio
Buffer
Commit
Store Buffer

前瞻执行的循环展开

0	Add1	No					Reservation
0	Add2	No					Stations
0	Add3	No					
0	Mult1	No	MULT	Mem[0+Regs[R1]]	Regs[F2]	#2	
0	Mult2	No	MULT	Mem[0+Regs[R1]]	Regs[F2]	#7	

	Busy	Address
Load1	No	
Load2	No	
Load3		

First loop

Second loop

Entry	Busy	Instruction	State	Destination	Value	
1	No	LD F0, 0(R1)	commit	F0	Mem[0+R1]	
2	No	MULT F4, F0, F2	commit	F4	F0 x F2	
3	Yes	SD 0(R1), F4	write	0+Reg[R1]	#2	
4	Yes	SUBI R1, R1, 8	write	R1	R1 - 8	
5	Yes	BNEZ R1, Loop	write			
6	Yes	LD F0, 0(R1)	write	F0	Mem[#4]	Reorder Buffer
7	Yes	MULT F4, F0, F2	write	F4	#6 X F2	
8	Yes	SD 0(R1), F4	write	0+Regs[R1]	#7	
9	Yes	SUBI R1, R1, 8	write	R1	#4 - 8	
10	Yes	BNEZ R1, Loop	write			

	F0	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
Reorder #	6		7						
Busy	yes	no	yes	no	no	no	no		no

支持前瞻执行的 Tomasulo 算法的总结

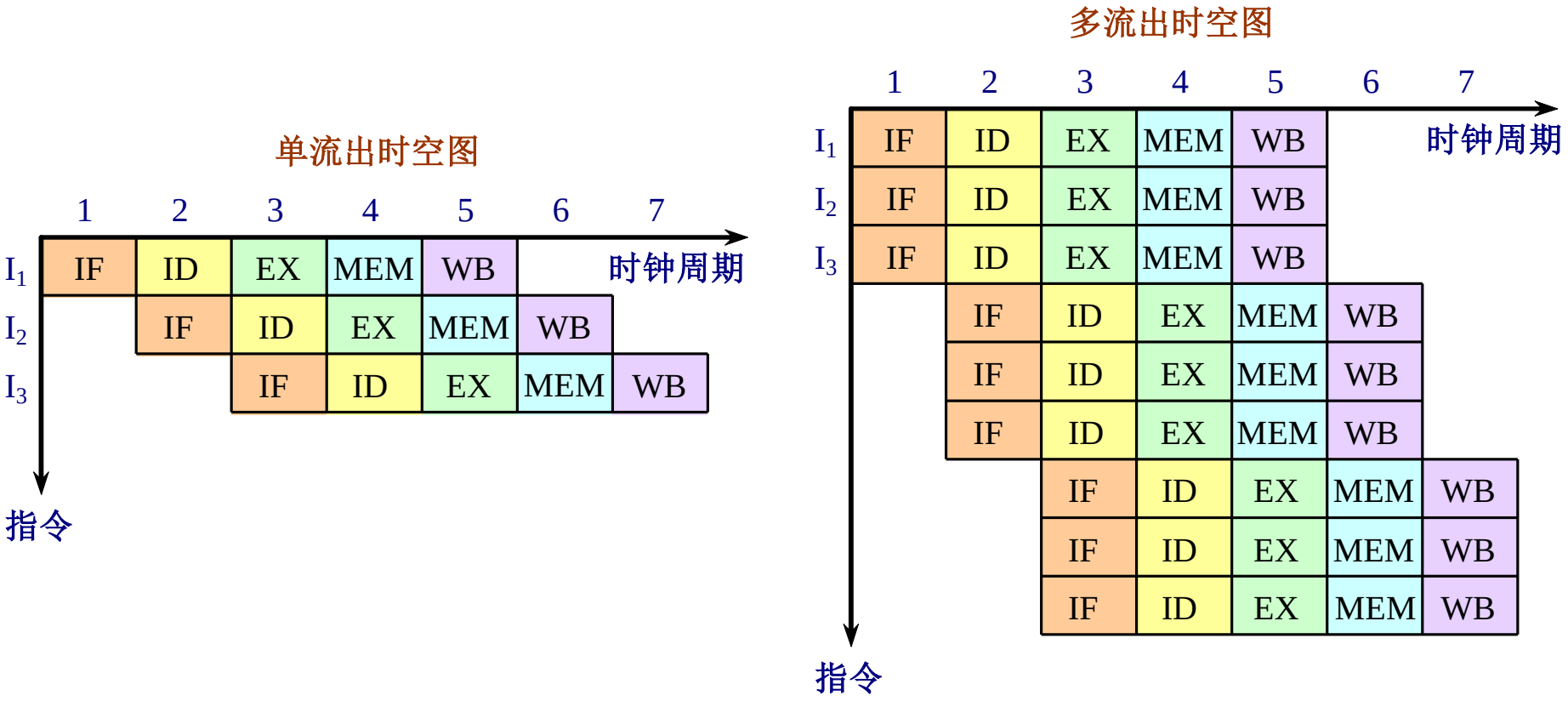
- 实际系统的实现非常相似
  - **Pentium P6, PowerPC, MIPS多种型号等**
- 不足之处：硬件非常复杂
  - **Too many value copy operations**
    - **Register File →RS→ROB→Register File**
  - **Too many muxes/buses (CDB)**
    - **Values are from everywhere to everywhere else!**
  - **Reservation Stations mix values(data) and tags (control)**
    - **Slow down max clock frequency**

7.4 多指令流出技术

7.4.1 静态超标量技术

7.4.2 动态多指令流出技术

7.4.3 超长指令字技术



单流出和多流出处理机执行指令的时空图

7.4 多指令流出技术

- 多指令流出处理器
  - 实现一个时钟周期内流出多条指令时
  - 达到CPI小于1
- 多流出处理器2种基本结构
  - 超标量（Superscalar）
    - 超标量每个时钟周期流出的指令数不定
    - 可以编译器静态调度，也可以硬件动态调度
  - 超长指令字（ VLIW，Very long Instruction Word）
    - 每个时钟周期流出的指令数是固定的，只能通过编译静态调度

技 术	流出结构	冲突检测	调 度	主要特点	处理机实例
超标量（静态）	动态	硬件	静态	按序执行	Sun UltraSPARCII/III
超标量（动态）	动态	硬件	动态	部分乱序执行	IBM Power2
超标量（前瞻）	动态	硬件	带有前瞻的动态调度	带有前瞻的乱序执行	Pentium III/4, MIPS R10K, Alpha 21264, HP PA 8500, IBM RS64III
VLIW /LIW	静态	软件	静态	流出包指令之间没有冲突	Trimedia，i860
EPIC	主要是静态	主要是软件	主要是静态	相关性被编译器显式地标记出来	Itanium (IA64)

超标量技术

- 超标量处理机的原型来自于IBM实验室的“America”处理器（John Cocke）
  - RS/6000第一个采用超标量技术
  - 现在，几乎所有高性能处理器都是用该技术

- 超标量处理机的硬件支持每个时钟周期发出1-8条不存在相关的指令
  - 如果指令流中的指令相关或不满足限制条件，则只能流出这条指令前面的指令，因此超标量处理器流出的指令数是不定的

7.4.1 静态超标量技术

- 假设有这样一个简单的超标量处理器，每个时钟周期它可以流出两条指令
  - 一条指令可以是取指令、存指令、分支指令或整数运算操作
  - 另一条指令可以是任意的浮点操作
  - 这种配置与**HP 7100**结构类似

超标量处理机的理想执行情况

指令	流水线工作情况							
	1	2	3	4	5	6	7	8
整数指令	IF	ID	EX	MEM	WB			
浮点指令	IF	ID	EX	MEM	WB			
整数指令		IF	ID	EX	MEM	WB		
浮点指令		IF	ID	EX	MEM	WB		
整数指令			IF	ID	EX	MEM	WB	
浮点指令			IF	ID	EX	MEM	WB	
整数指令				IF	ID	EX	MEM	WB
浮点指令				IF	ID	EX	MEM	WB

超标量处理机的技术问题

- 每个时钟周期流出两条指令意味着同时取两条指令（64位），译码两条指令（64位）
  - 假设：指令按要求组合成对，且与 64位边界对其，整数指令顺序在前
  - 需要使得浮点部件流水化或增加相关检测部件来减少结构相关
- 另一个限制超标量流水线性能发挥的障碍是取操作和分支操作的延迟
  - 取操作指令的结果不能在本周期和下一个周期使用
  - 分支指令肯定是指令组合的第一条指令，影响配对指令和后续两条指令，分支延迟也变为3条指令

超标量：例子

- 为了能有效利用超标量处理器的可获得的并行度，需要采用更有效的编译技术、硬件调度技术和更复杂的指令译码技术
  - 循环展开成5个副本
- 使用先前我们用到的代码作为例子

```
LOOP：LD      F0, 0(R1)    ;F0=数组元素
      ADDD    F4, F0, F2   ;与F2寄存器中的标量相加
      SD      0(RI), F4    ;存结果
      SUBI    R1, R1, #8   ;R1寄存器值减8
      BNEZ    R1, LOOP     ;如果R1值不为0，则跳转
```

超标量：例子执行情况

- 在超标量流水线上对上述代码进行调度，以获取更多地指令机并行度

	整数指令		浮点指令		时钟周期
Loop:	LD	F0(R1)			1
	LD	F6,-8(R1)			2
	LD	F10,-16(R1)	ADDD	F4,F0,F2	3
	LD	F14,-24(R1)	ADDD	F8,F6,F2	4
	LD	F18,-32(R1)	ADDD	F12,F10,F2	5
	SD	0(R1),F4	ADDD	F16,F14,F2	6
	SD	-8(R1),F8	ADDD	F20,F18,F2	7
	SD	-16(R1),F12			8
	SD	-24(R1),F16			9
	SUBI	R1,R1,#40			10
	BNEZ	R1,Loop			11
	SD	8 (R1),F20			12

超标量：例子结果分析

- 每次循环需12个时钟周期
- 每个迭代时2.4个时钟周期
  - 前面在普通的流水线上，通过循环展和调度，可以达到每个迭代为3.5个时钟周期

- 超标量可以获得更好地性能，代价是硬件复杂性大幅度增加

7.4.2 动态多指令流出技术

- 扩展Tomasulo算法：支持双流出超标量流水线
  - 每个时钟周期流出两条指令；
  - 一条是整数指令，另一条是浮点指令。

采用一种比较简单的方法：

- 指令按顺序流向保留站
- 将整数寄存器与浮点寄存器分开，只要不使用相同的寄存器，同时地流出一条浮点指令和一条整数指令到各自的保留站。

7.4.2 动态多指令流出技术

- 有两种不同的方法可以实现多流出
  - 在半个时钟周期里完成流出步骤，这样一个时钟周期就能处理两条指令。
  - 对流出指令的组合进行限制，设置一次能处理两条指令的逻辑电路。

现代的流出4条或4条以上指令的超标量处理机经常是两种方法都采用，不仅采用流水而且还把流出电路加宽。

对于采用了Tomasulo算法和多流出技术的MIPS流水线，考虑以下简单循环的执行，列表表示前面三遍循环的各指令流出、开始执行、访存、结果写到CDB的时间。

```
LOOP：LD      F0, 0(R1)      ;F0=数组元素.
      ADDD    F4, F0, F2     ;与F2寄存器中的标量相加
      SD      F4, 0(RI)      ;存结果
      SUBI    R1, R1, #8     ;R1寄存器值减8
      BNEZ    R1, LOOP       ;如果R1值不为0，则跳转
```

（1）无论是否相关，每个时钟周期流出一条整数指令和一条浮点数指令；

（2）有一个整数部件，用于整数运算和地址计算；有一个独立的浮点功能部件；

（3）指令流出和写结果各占用1个时钟周期；

（4）有1个具有独立分支预测能力的分支预测部件，分支指令只能单独流出，没有分支延迟；

（5）因为写结果占用1个时钟周期，所以产生结果的延迟为：整数运算1个周期，存储器取数操作2个周期，浮点运算3个周期。

遍数	指　　令		流出	执行	访存	写CDB	说明
1	LD	F0,0(R1)	1	2	3	4	流出第一条指令
1	ADDD	F4,F0,F2	1	5		8	等待LD的结果
1	SD	0(R1),F4	2	3	9		等待ADDD的结果
1	SUBI	R1,R1,#-8	2	4		5	等待计算的ALU
1	BNEZ	R1,Loop	3	6			等待SUBI的结果
2	LD	F0,0(R1)	4	7	8	9	等待BNEZ完成
2	ADDD	F4,F0,F2	4	10		13	等待LD的结果
2	SD	0(R1),F4	5	8	14		等待ADDD的结果
2	SUBI	R1,R1,#-8	5	9		10	等待ALU
2	BNEZ	R1,Loop	6	11			等待SUBI的结果
3	LD	F0,0(R1)	7	12	13	14	等待BNEZ完成
3	ADDD	F4,F0,F2	7	15		18	等待LD的结果
3	SD	0(R1),F4	8	13	19		等待ADDD的结果
3	SUBI	R1,R1,#-8	8	14		15	等待ALU
3	BNEZ	R1,Loop	9	16			等待SUBI的结果

可以看出：

- 每3个时钟周期就执行一个新循环迭代，每个循环5条指令。

  IPC＝5/3＝1.67条/拍

- 虽然指令的流出率比较高，但是执行效率并不是很高。
  - 16拍共执行15条指令，
  - 平均指令执行速度为15/16＝0.94条/拍。
- 原因是浮点运算少，ALU部件成了瓶颈。

解决方法：增加一个加法器，把ALU功能和地址运算功能分开。

- 上述双流出动态调度流水线的性能受限于以下3个因素：
  - 整数部件和浮点部件的工作负载不平衡，没有充分发挥出浮点部件的作用。
    - ✓ 应该设法减少循环中整数型指令的数量。
  - 每个循环叠代中的控制开销太大。
    - 5条指令中有两条指令是辅助指令；
    - 应该设法减少或消除这些指令。
  - 控制相关使得处理机必须等到分支指令的结果出来后才能开始下一条LD指令的执行。

7.4.3 超长指令字技术

- 一台超标量机器每周期能够流出4-8条指令
  - 由于必须要用硬件分析指令间的相关，为其实现带来了困难
- 另一种选择：长指令字（LIW，Long Instruction Word）或称为超长指令字（VLIW，Very Long Instruction Word）体系结构
  - 第一种商用LIW机器是AP-120B，由Floating Point Systems Inc.开发
  - FPS-164是较新的机器，它的每个指令字含有对应于10个不同功能单元的10条指令

VLIW基本机构

- VLIW采用多个独立的功能单元，多个不同的操作封装在一条长指令字中，每个功能单元在VLIW指令中都有一定的对应区域
  - 一般每个功能单元占用16-24位
  - 例如：2个整数、2个浮点、2个访存、1个分支，则该指令的长度为112-168位
- VLIW硬件只是简单地将指令字中对应的部分送给各个功能单元，功能单元在哪一个时钟周期执行什么操作由编译器来确定
  - 如果某个功能单元在某个周期没有任务，则执行NOP指令

VLIW例子

- 再次使用先前在解释循环展开及超标量机器时使用过的那段循环代码，来解释VLIW如何工作这一次，我们将循环展开n个副本
- VLIW机器每条指令字包含
  - 两个访存操作
  - 两个浮点操作
  - 一个整数或分支操作
- 得到如下指令序列

VLIW例子的指令执行情况，n=5

访存操作 1	访存操作 2	浮点操作 1	浮点操作 2	整数操作/分支
LD F0,0(R1)	LD F6,-8(R1)			
LD F10,-16(R1)	LD F14,-24(R1)			
LD F18,-32(R1)		ADDD F4,F0,F2	ADDD F8,F6,F2	
		ADDD F12,F10,F2	ADDD F16,F14,F2	
		ADDD F20,F18,F2		
SD 0(R1),F4	SD -8(R1),F8			
SD -16(R1),F12	SD -24(R1),F16			SUBI R1,R1,#40
SD 8(R1),F20				BNEZ R1,Loop

- **5条结果在8个时钟周期中完成计算**
  - **每条结果花费1.3时钟周期**
  - **有17/40的指令槽被放入了有效的操作**

执行情况，n=7

Mem Ref1	Mem Ref2	FP1	FP2	Int/branch
LD F0,0(R1)	LD F6,-8(R1)			
LD F10,-16(R1)	LD F14,-24(R1)			
LD F18,-32(R1)	LD F22,-40(R1)	ADDD F4,F0,F2	ADDD F8,F6,F2	
LD F26,-48(R1)		ADDD F12,F10,F2	ADDD F16,F14,F2	
		ADDD F20,F18,F2	ADDD F24,F22,F2	
SD F4,0(R1)	SD F8,-8(R1)	ADDD F28,F26,F2		
SD F12,-16(R1)	SD F16,-24(R1)			SUBI R1,R1,#56
SD F20,24(R1)	SD F24,16(R1)			
SD F28,8(R1)				BNEZ R1, Loop

结果分析

- **9拍产生7个结果**
  - **每个结果1.29拍**
  - **比前面的超标量，每个结果2.4拍，快接近2倍**
- **9拍里面执行了23个操作**
  - **2.5操作/拍**
  - **指令槽利用率不高， 只有51% (23/45)**
- **使用大量寄存器**
  - **7个循环，共计使用2×7+1=15个64位浮点寄存器**

没有空指令的调度, n=3

Mem Ref1	Mem Ref2	FP1	FP2	Int/branch
LD F0, 0(R1)	LD F6,-8(R1)			
LD F10, -16(R1)				
		ADDD F4,F0,F2	ADDD F8,F6,F2	
		ADDD F12,F10,F2		
				SUBI R1, R1, #24
SD F4,24(R1)	SD F8,16(R1)			
SD F12,8(R1)				
				BNEZ R1, Loop

- 8拍3个结果，每个结果2.66拍
- 指令槽利用率: 11/40 = 27.5%
- 寄存器：3×2+1=7个64位浮点寄存器

尽可能指令空间充满的调度, n=10

Mem Ref1	Mem Ref2	FP1	FP2	Int/branch
LD F0,0(R1)	LD F6,-8(R1)			
LD F10,-16(R1)	LD F14,-24(R1)			
LD F18,-32(R1)	LD F22,-40(R1)	ADDD F4,F0,F2	ADDD F8,F6,F2	
LD F26,-48(R1)	LD F30,-56(R1)	ADDD F12,F10,F2	ADDD F16,F14,F2	
LD F34,-64(R1)	LD F38,-72(R1)	ADDD F20,F18,F2	ADDD F24,F22,F2	SUBI R1,R1,#80
SD F4,80(R1)	SD F8,72(R1)	ADDD F28,F26,F2	ADDD F32,F30,F2	
SD F12,64(R1)	SD F16,56(R1)	ADDD F36,F34,F2	ADDD F40,F38,F2	
SD F20,48(R1)	SD F20,40(R1)			
SD F20,32(R1)	SD F20,24(R1)			
SD F20,16(R1)	SD F20,8(R1)			BNEZ R1,Loop

- 10拍10个结果，每个结果使用1拍
- 指令槽利用率：36/50 =72%
- 寄存器需求：10×2+1=21个64位浮点寄存器

VLIW的技术难题

- 第一，从线性代码片段中产生足够的操作需要进行激进的循环展开，这增大了代码大小
- 第二，无论指令是否被充满，没有被使用的功能单元也在指令字编码过程中占据了相应的位。将近一半的指令是被浪费掉的
  - 在主存中压缩指令，在cache中解压缩指令
- 第三，VLIW带来了二进制代码兼容性问题
  - 采用机器代码翻译或仿真模拟的方法解决移植的问题

多指令流出的技术难题

指令多流出处理器受哪些因素的限制呢？

主要受以下3个方面的影响：

- 程序所固有的指令级并行性；
- 硬件实现上的困难；
- 超标量和超长指令字处理器固有的技术限制。

– 设计多流出处理器的难点：

- 访存开销
- 硬件复杂性
- 编译器技术

ILP的展望

2000年以后4种IBM Power处理器特性

	Power4	Power5	Power6	Power7
发布时间	2001	2004	2007	2010
最初时钟频（GHz）	1.3	1.9	4.7	3.6
晶体管数量（百万）	174	276	790	1200
每时钟周期的发射数	5	5	7	6
功能单元	8	8	9	12
核心数	2	2	2	8
总的片上缓存（MB）	1.4	2	4.1	32.3

多核系统结构

- 多核技术是指在一枚处理器中集成两个或多个完整的计算内核，从而提高计算能力的技术。
- 按计算内核的对等与否，多核系统结构又可以分为同构多核结构和异构多核结构两种。
  - 计算内核相同，地位对等的称为同构多核，反之称为异构多核。

需要注意的是，多核系统结构与多处理器不同，多处理器指多个CPU，每个CPU可以是单核或多核的。

多核系统结构

- 多核处理器的组织架构主要包括：片上核心处理器的个数、多少级Cache、共享Cache的容量和内部互连结构等。
- 多核系统的4种典型的组织结构：
  - 专用L1 Cache多核系统结构
  - 专用L2 Cache多核系统结构
  - 共享L2 Cache多核系统结构
  - 共享L3 Cache多核系统结构

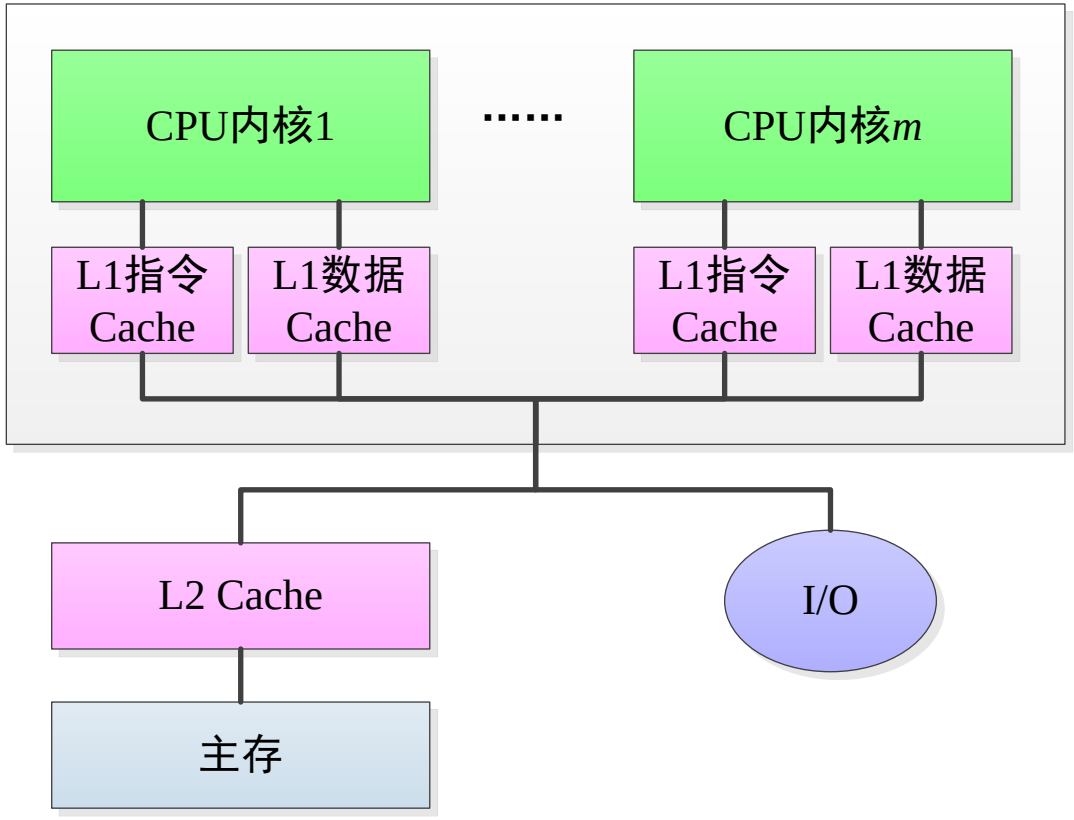
多核系统结构

（A）专用L1 Cache多核系统结构

早期多核处理器的一种组织架构，现在在嵌入式芯片中仍能见到。

在这种组织方式中，只有一级片内Cache，每个核带有自己的专用L1 Cache，分成指令Cache和数据Cache。

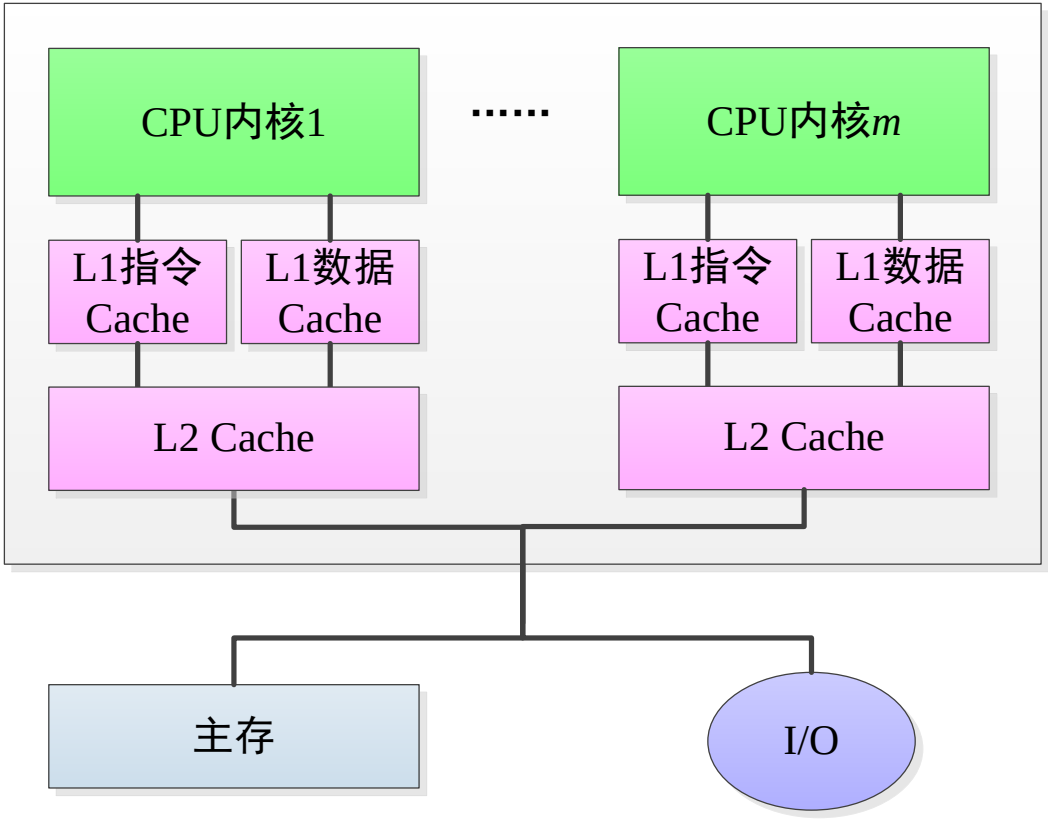
这种组织的一个典型实例是ARM11 MPCore。



多核系统结构

（B）专用L2 Cache多核系统结构

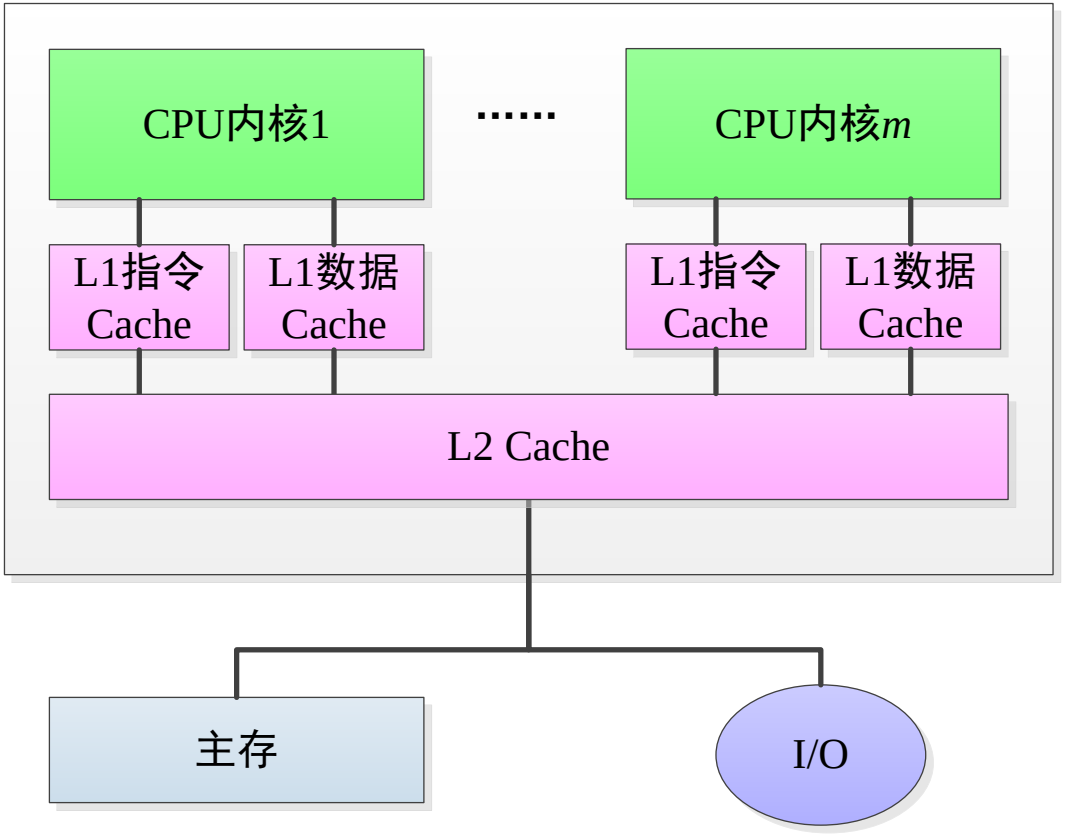
专用L2 Cache多核系统结构无片内共享Cache，在这种结构里，片内有足够的可用面积容纳多个L2 Cache。这种组织的一个典型实例是**AMD Opteron**。



多核系统结构

（C）共享L2 Cache多核系统结构

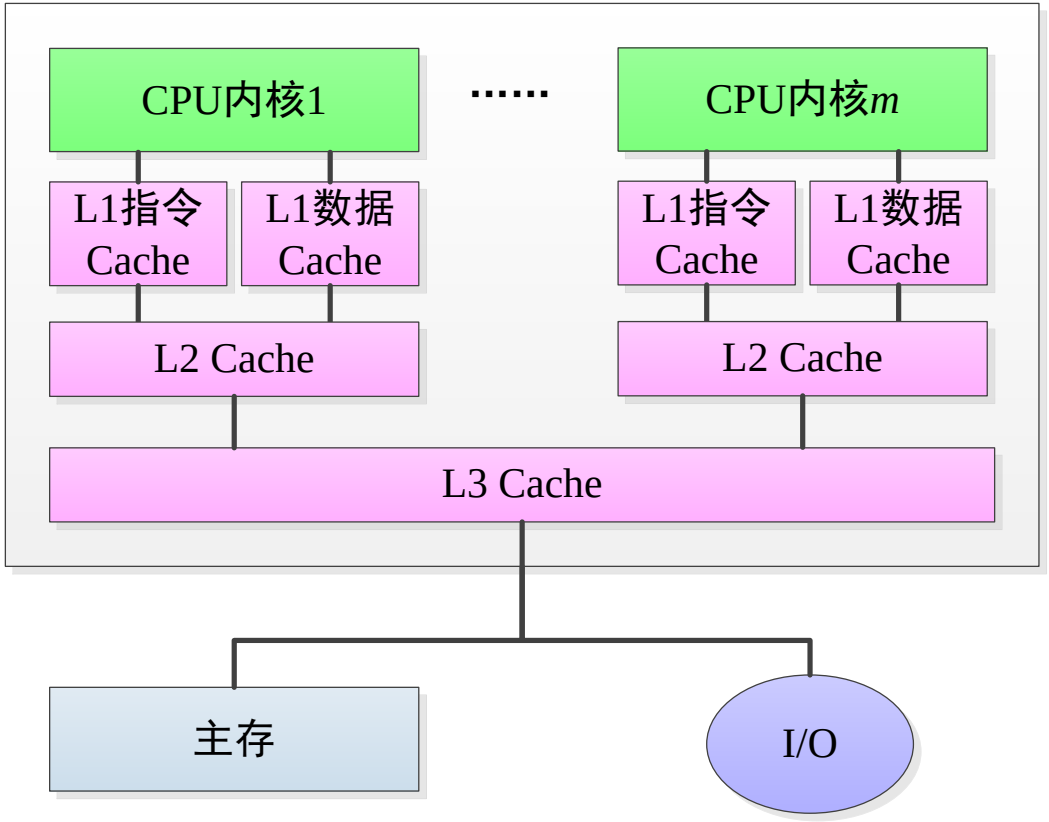
共享L2 Cache多核系统结构采用了和专用L2 Cache多核结构类似的存储空间分配，不同的是该处理器架构拥有共享L2 Cache，Intel的Core Duo处理器就是这种结构。



多核系统结构

（D）共享L3 Cache多核系统结构

共享L3 Cache多核系统结构出于性能上的考虑，分离出一个独立的三级Cache，每个CPU计算内核除了拥有专用的一、二级Cache外，还共享L3 Cache；**Intel Core i7**就是这种结构。



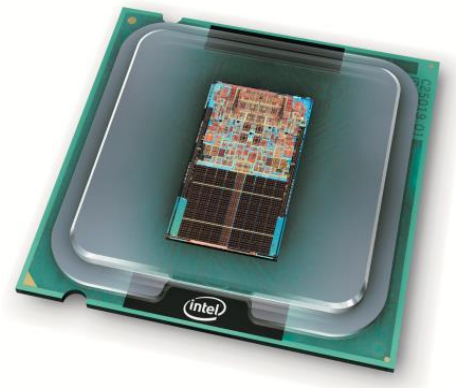
多核系统结构

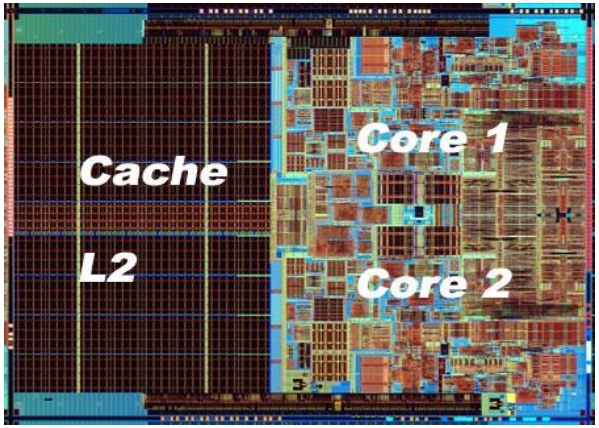
多核CPU产品有很多，几乎所有的厂商都推出了自己的多核产品：

- **Intel x86多核系统结构**
  - **Core Duo**
  - **Intel Core i7**
- 基于ARM多核系统结构，华为鲲鹏系列
- 自主国产龙芯Loongson多核系统结构

Intel x86多核系统结构 - Core Duo

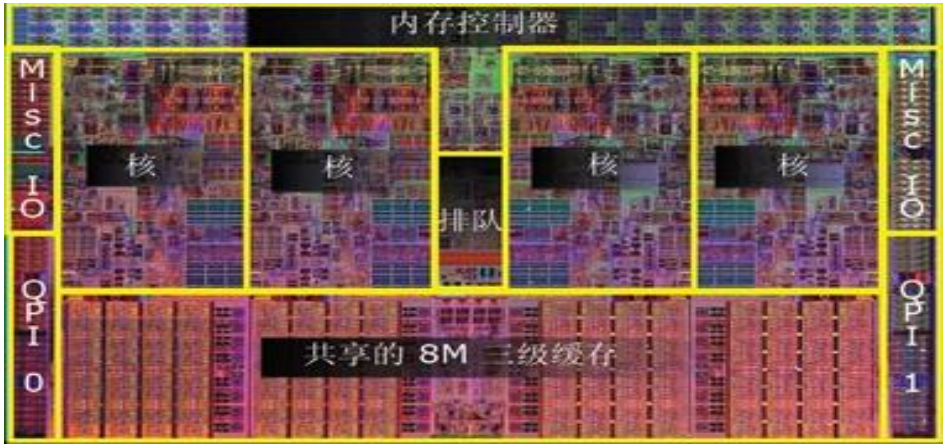
- 2006年推出的Core Duo是全球第一个低耗电的双核处理器（低于25瓦特）。
- Core Duo实现了两个x86超标量处理器，共享二级Cache，Core Duo的每个核有自己的专用L1 Cache：一个32KB的指令Cache和一个32KB的数据Cache。





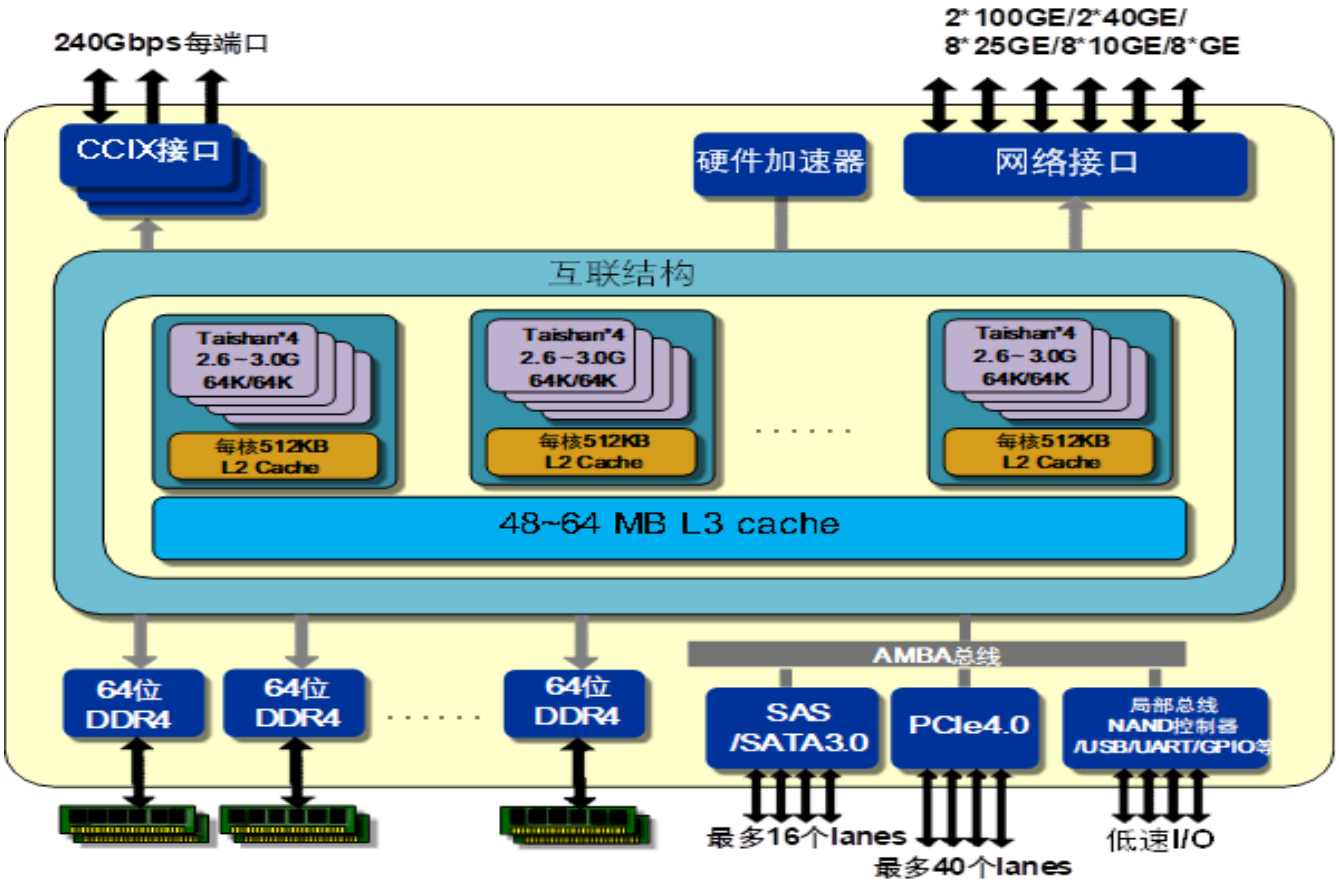
Intel x86多核系统结构 - Intel Core i7

- i7是Intel于2008年11月推出的，实现了4个x86 SMT计算核，每个计算核带一个专用的L2 Cache、一个共享的L3 Cache。
- 在Core i7中，每个核拥有自己的专用L2 Cache，4个核共享一个8MB的L3 Cache。



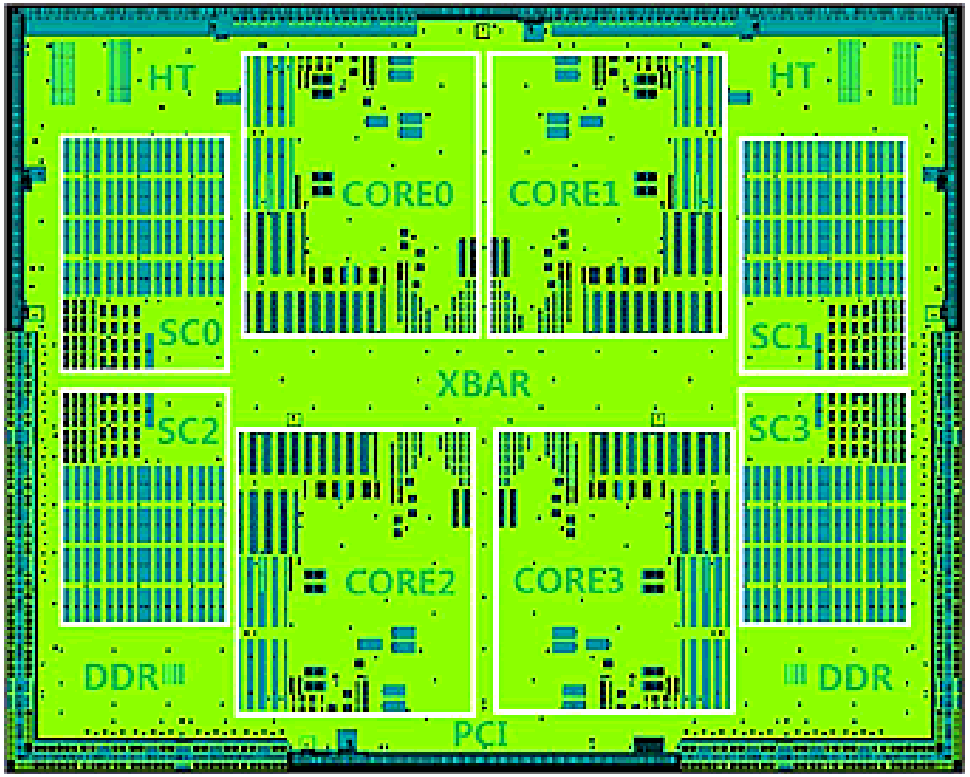
华为鲲鹏920系列多核系统架构

- 集成最多64×自研核，指令集兼容ARMv8.2, 最高主频达3.0GHz，每核集成64KB L1 I/D 缓存，每核独享 512KB L2 缓存，单芯片共享48-64MB L3缓存
- 支持CCIX接口，支持FPGA/ASIC加速器的缓存一致性



Loongson多核系统结构

- 龙芯3A2000处理器核心采用自主设计高性能GS464E微结构，支持自主龙芯指令系统LoongISA，四发射乱序执行。
- 私有L1指令Cahche和数据Cache 64KB，私有L2级Cache 256KB，共享L3级Cache 4MB，流水线深度12级。



指令级并行总结

1. 指令级并行的概念
   - 循环展开调度的基本方法
   - 指令的相关性
2. 指令的动态调度
   - 记分牌
   - **Tomasulo**算法
3. 控制相关的动态解决技术
   - 分支预测缓冲
   - 分支目标缓冲
   - 前瞻技术
4. 多指令流出技术
   - 超标量技术
   - 超长指令字技术

第7章作业

- 王志英教材：**P159　T3、T4、T6**
  - ✓ 注：第六题，**DONE是X的最后一个元素存放在存储器的地址**
  - ✓ **11周周三下午，和第六章作业一起交**

- 华为微认证：鲲鹏处理器揭秘 **https://edu.huaweicloud.com/certifications/9ae486f67b224abbbbf58049c3005efe**
  - ✓ 相当于一次作业，大概只需要**1-2**个小时，
  - ✓ 完成在线实验之后将截图发给助教，**15**周周三前完成
  - ✓ 学生微认证代金卷获取链接：**https://edu.huaweicloud.com/activity/colleges-students.html**

- 报 告 ： **Recent technologies for high performance computing**
  - ✓ 近5年文章中出现的高性能计算方法和技术，可以从流水线技术、分支预测、指令预取、Cache与存储器、存算一体、编译技术、异构加速等方面进行选题。
  - ✓ 内容包括：如何实现性能的提升，还存在什么问题及可能解决的方法
  - ✓ 十五周周三之前提交给助教

- 企业讲座：《华为鲲鹏CPU架构基础》、《鲲鹏BoostKit加速库》
  - ✓ 安排在十三周左右，及具体时间QQ群通知，
  - ✓ 会有一次Quiz。

谢谢各位同学！