中国科学技术大学
University of Science and Technology of China

Ch-7 指令的调度与并行化

王 超

中国科学技术大学计算机学院
高能效智能计算实验室

2026年春

内容提要

- 基本指令集调度
  - ✓指令重排调度
  - ✓循环展开
- 基于多发射的指令集并行调度
  - ✓超标量SuperScalar
  - ✓超长指令字VLIW

通过**基于循环**的例子来说明软件代码在指令集和处理器上的优化方法

指令集调度

- 何为指令集调度
  - ✓通过调整重排指令集的执行(顺序)，提升指令的并行性
  - ✓避免非法或者模糊语义的操作，保证正确性（如相关等）
- 为何要研究指令集调度
  - ✓减少流水线停顿
  - ✓提升指令集并行
- 指令集调度实现
  - ✓静态调度-编译器优化（如分支）
  - ✓动态调度-硬件实现（记分牌、Tomasulo）

指令级调度的若干定义

- **基本块**的定义
  - ✓无分支直线型代码，顺序执行
  - ✓整个程序是由分支语句连接基本块构成
  - ✓MIPS 的分支指令占15%左右，基本块的大小在4~7条指令
  - ✓OS代码中的分支较少
    - 负责资源管理
    - 填写状态寄存器
    - 填写控制寄存器
    - 设置控制变量
- 循环级并行- 循环的特征
  - ✓控制循环的分支指令是有执行偏好的
  - ✓绝大多数是成功的，预测比较容易，需要预测方案
    （分支预测机制、分支延迟机制）

研究指令集调度的原因

- 流水线的平均CPI（影响性能的因素）

  Pipeline CPI = Ideal Pipeline CPI
  + Structure Stalls
  + RAW Stalls
  + WAR Stalls
  + WAW Stalls
  + Control Stalls

- 指令集调度主要研究
  - ✓减少停顿（stalls)数的方法和技术

采用的基本技术

定向路径
分支延迟
记分牌
重命名
分支预测
多发射
推测执行
内存相关预测
循环展开
编译器调度
编译器相关分
软件流水线
编译器推测

Technique	Reduces
Forwarding and bypassing	Potential data hazard stalls
Delayed branches and simple branch scheduling	Control hazard stalls
Basic dynamic scheduling (scoreboarding)	Data hazard stalls from true dependences
Dynamic scheduling with renaming	Data hazard stalls and stalls from antidependences and output dependences
Dynamic branch prediction	Control stalls
Issuing multiple instructions per cycle	Ideal CPI
Speculation	Data hazard and control hazard stalls
Dynamic memory disambiguation	Data hazard stalls with memory
Loop unrolling	Control hazard stalls
Basic compiler pipeline scheduling	Data hazard stalls
Compiler dependence analysis	Ideal CPI, data hazard stalls
Software pipelining, trace scheduling	Ideal CPI, data hazard stalls
Compiler speculation	Ideal CPI, data, control stalls

简单循环及其对应的汇编程序

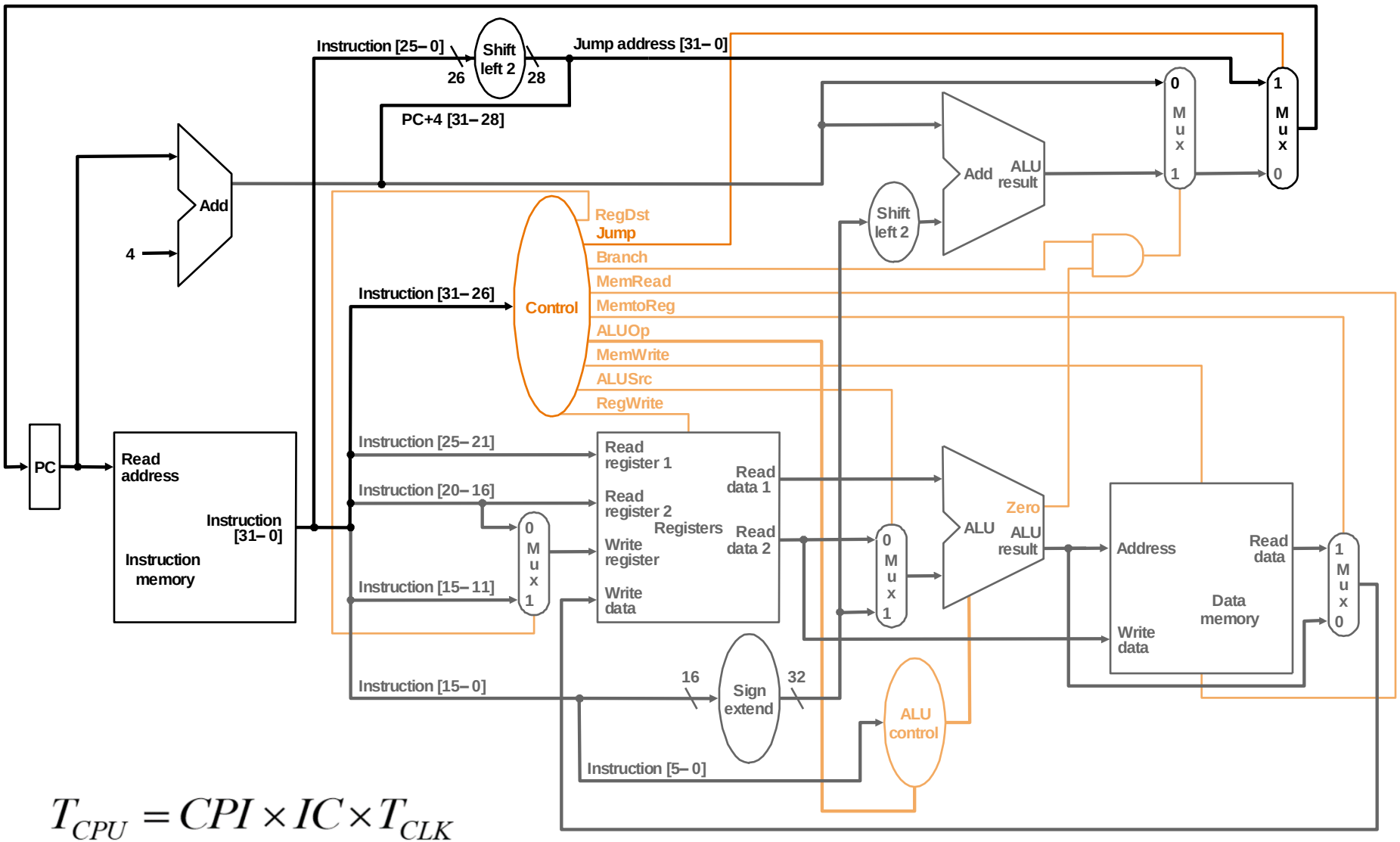

```
for    (i=1; i<=1000; i++)
          x(i) = x(i) + s;
```

```
Loop:  LD      F0,0(R1)        ;F0=vector element
       ADDD    F4,F0,F2        ;add F2
       SD      0(R1),F4        ;store result
       SUBI    R1,R1,8         ;decrement pointer 8B (DW)
       BNEZ    R1,Loop         ;branch R1!=zero
```

1. 如为定点运算访存指令，采用单周期和多周期两种执行方式，需要多少时钟周期完成每轮循环指令流出?

复习

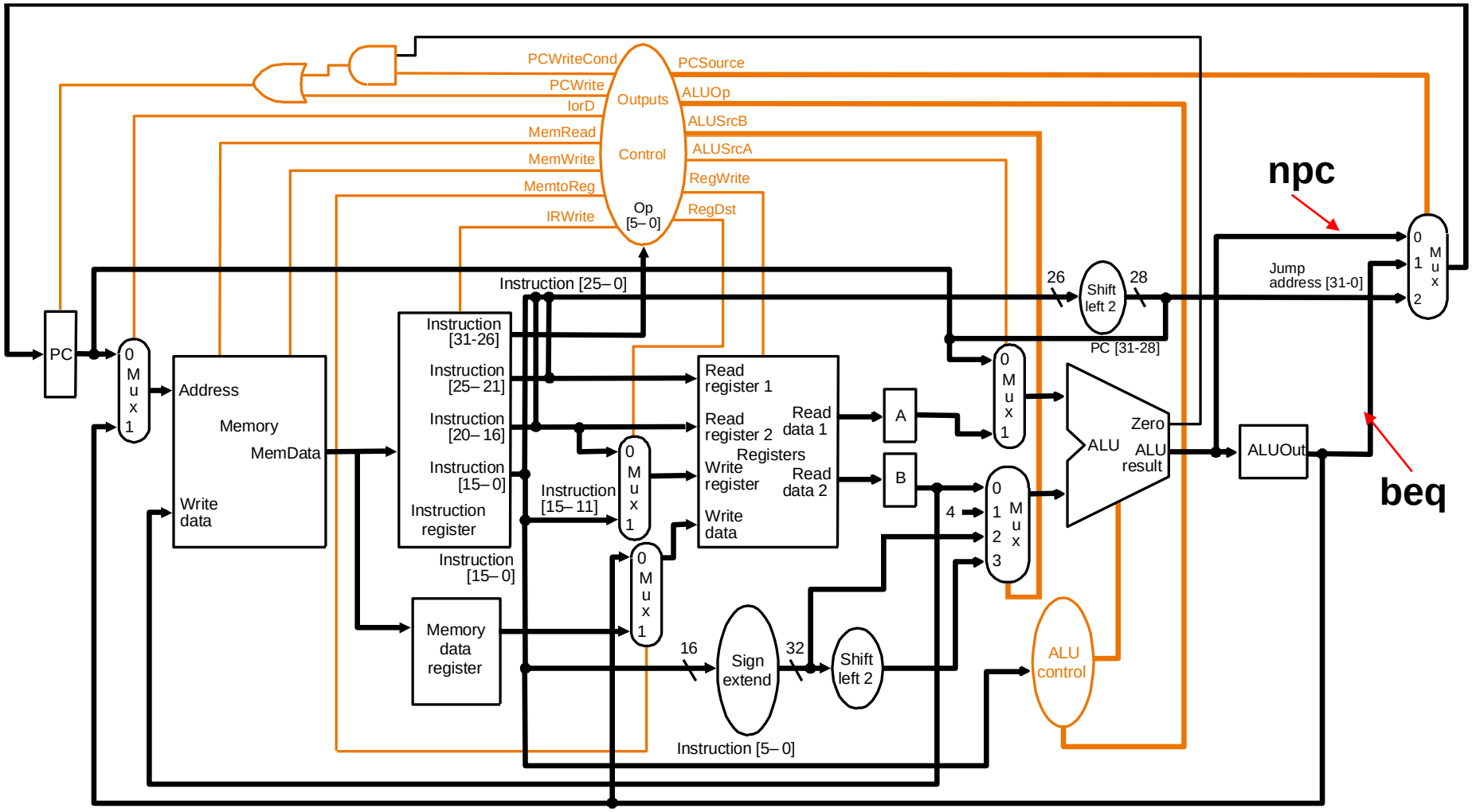
```
Loop:  LD     F0,0(R1)
       ADDD   F4,F0,F2
       SD     0(R1),F4
       SUBI   R1,R1,8
       BNEZ   R1,Loop
```



$$T_{CPU} = CPI \times IC \times T_{CLK}$$

复习

```
Loop:  LD     F0,0(R1)
       ADDD   F4,F0,F2
       SD     0(R1),F4
       SUBI   R1,R1,8
       BNEZ   R1,Loop
```



$$T_{CPU} = CPI \times IC \times T_{CLK}$$

流水线执行-分析循环中的相关

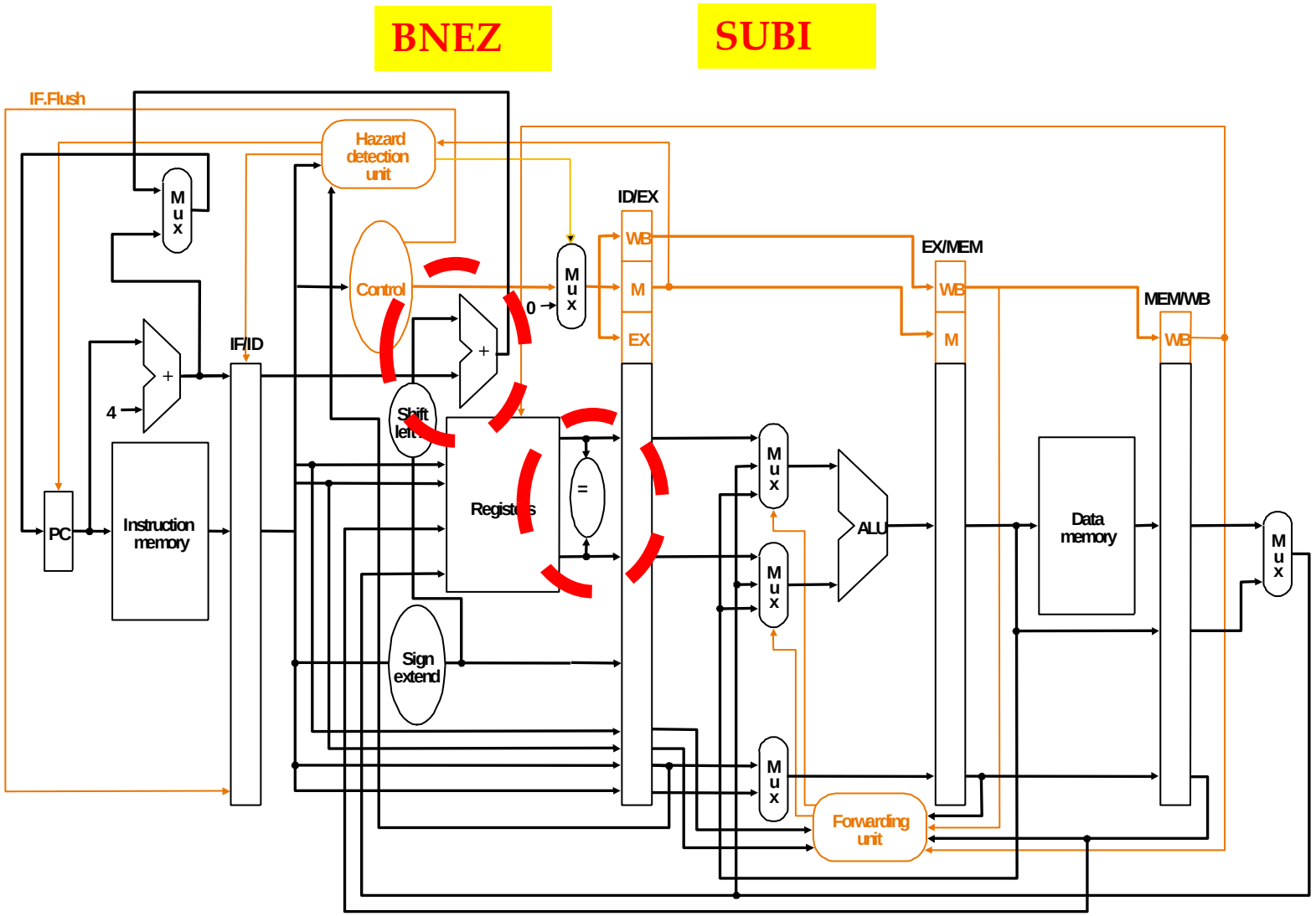

```
Loop:   LD      F0,0(R1)    ;F0=vector element
        ADDD    F4,F0,F2    ;add scalar from F2
        SD      0(R1),F4    ;store result
        SUBI    R1,R1,8     ;decrement pointer 8B (DW)
        BNEZ    R1,Loop     ;branch R1!=zero
        NOP                 ;delayed branch slot
```

做如下**假设：**

产生结果的指令	**使用结果的指令**	**所需的延时**
FP ALU op	**Another FP ALU op**	**3 （浮点）**
FP ALU op	**Store double**	**2 （浮点）**
Load double	**FP ALU op**	**1 （浮点）**
SUBI	**BNEZ**	**1（定点）**
BNEZ	**XXX**	**1（定点、分支延迟）**

需要在哪里加**stalls?**（假设分支在**ID**段得到地址和条件）

复习-流水线数据通路
分支在ID段结束



指令流出的时间


```
1 Loop:   LD     F0,0(R1)     ;F0=vector element
2         stall
3         ADDD   F4,F0,F2     ;add scalar in F2
4         stall
5         stall
6          SD    0(R1),F4     ;store result
7          SUBI  R1,R1,8      ;decrement pointer 8B (DW)
8         stall                Why Stall here?
9         BNEZ   R1,Loop      ;branch R1!=zero
10        stall                ;delayed branch slot
```

产生结果的指令	使用结果的指令	所需的延时
FP ALU op	Another FP ALU op	3
FP ALU op	Store double	2
Load double	FP ALU op	1
SUBI	BNEZ	1
BNEZ	XXX	1

优化1：指令调度（循环内）


```
1 Loop:   LD     F0,0(R1)
2         SUBI   R1,R1,8
3         ADDD   F4,F0,F2
4         stall
5         BNEZ   R1,Loop      ;delayed branch
6         SD     8(R1),F4     ;altered when move past SUBI
```

10->6 Clocks

Swap BNEZ and SD by changing address of SD

产生结果的指令	使用结果的指令	所需的延时
FP ALU op	Another FP ALU op	3
FP ALU op	Store double	2
Load double	FP ALU op	1
SUBI	BNEZ	1
BNEZ	XXX	1

通过循环展开4次是否可以提高性能?

优化2：循环展开4次(无循环内调度)


```
1 Loop: LD    F0,0(R1)    stall
2       ADDD  F4,F0,F2    stall    stall
3       SD    0(R1),F4              ;drop SUBI & BNEZ
4       LD    F6,-8(R1)   stall
5       ADDD  F8,F6,F2    stall stall
6       SD    -8(R1),F8             ;drop SUBI & BNEZ
7       LD    F10,-16(R1) stall
8       ADDD  F12,F10,F2  stall stall
9       SD    -16(R1),F12           ;drop SUBI & BNEZ
10      LD    F14,-24(R1) stall
11      ADDD  F16,F14,F2  stall  stall
12      SD    -24(R1),F16
13      SUBI  R1,R1,#32   stall   ;alter to 4*8
14      BNEZ  R1,LOOP
15      NOP
```

- 循环展开降低开销
- 重命名解决WAW-WAR相关

15 + 4 x (1+2) + 1 = 28 cycles, or 7 per iteration

Assumes R1 is multiple of 4

优化1+2：指令调度+循环展开

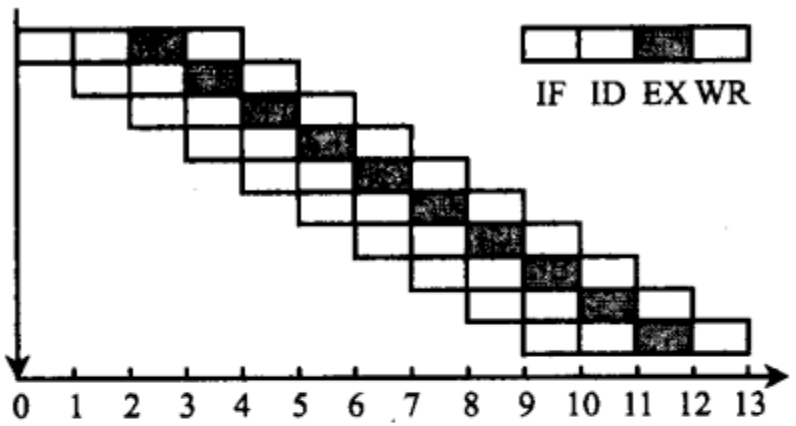

```
1 Loop:  LD     F0,0(R1)
2        LD     F6,-8(R1)
3        LD     F10,-16(R1)
4        LD     F14,-24(R1)
5        ADDD   F4,F0,F2
6        ADDD   F8,F6,F2
7        ADDD   F12,F10,F2
8        ADDD   F16,F14,F2
9        SD     0(R1),F4
10       SD     -8(R1),F8
11       SUBI   R1,R1,#32
12       SD     16(R1),F12
13       BNEZ   R1,LOOP
14       SD     8(R1),F16       ; 8-32 = -24
```

- 代码移动后
  - ✓SD移动到SUBI后，注意偏移量的修改

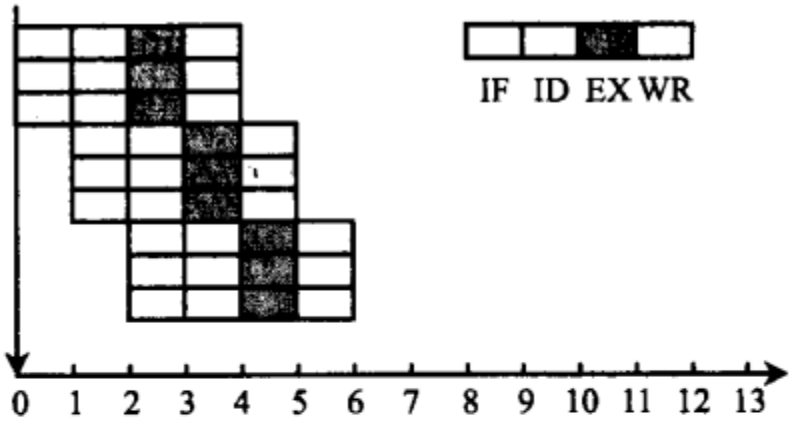
14 clock cycles, or 3.5 per iteration

是否还有进一步的优化手段?

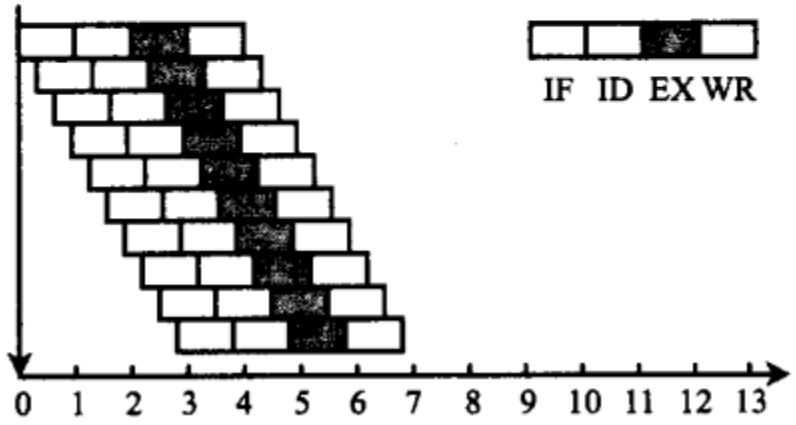
复习-四种流水技术



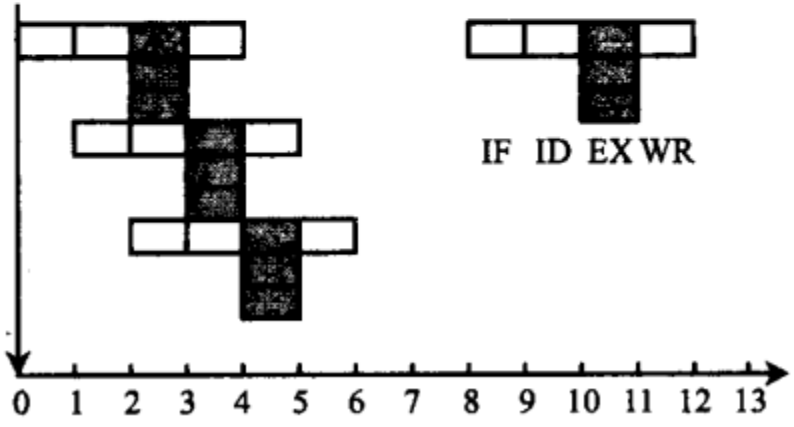
(a) 普通流水



(b) 超标量流水



(c) 超流水线



(d) 超长指令字

Superscalar

- Superscalar: 每个时钟周期发射2条指令，1条FP指令和1条其他指令
  - 每个时钟周期取64位; 左边为Int , 右边为FP
  - 只有第一条指令发射了，才能发射第二条
  - 需要更多的寄存器端口，因为如果两条指令中第一条指令是对FP的load操作（通过整数部件完成），另一条指令为浮点操作指令，则都会有对浮点寄存器文件的操作

Type	*Pipe Stages*								
Int. instruction	IF	ID	EX	MEM	WB				
FP instruction	IF	ID	EX	MEM	WB				
Int. instruction		IF	ID	EX	MEM	WB			
FP instruction		IF	ID	EX	MEM	WB			
Int. instruction			IF	ID	EX	MEM	WB		
FP instruction			IF	ID	EX	MEM	WB		

优化4：采用Superscalar的循环展开

	Integer instruction	*FP instruction*	*Clock cycle*
Loop:	LD F0,0(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

- 循环展开5次以消除延时
- 12 clocks, or 2.4 clocks per iteration

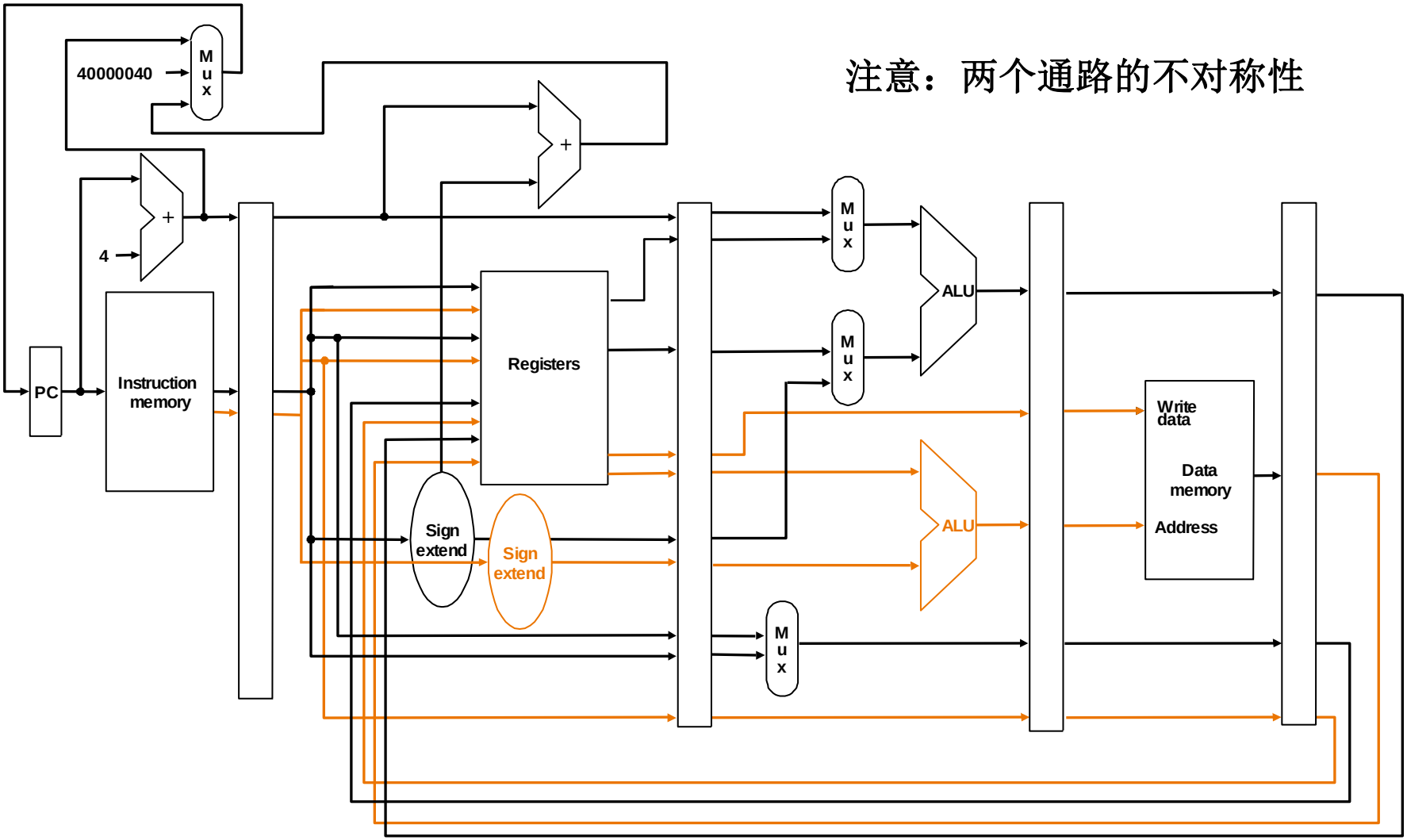
超长指令字VLIW

- 超长指令字VLIW
  - ✓指令字较长可以容纳较多的操作
  - ✓根据定义,VLIW中的所有操作是由**编译时刻**组合的，并且是相互无关的，可以并行执行
  - ✓例如 2 个整数操作，2个浮点操作，2个存储器引用，1个分支指令

    每一个操作用16到24位表示

    => 共7*16 = 112 bits到7*24 = 168 bits wide
  - ✓需要用**编译技术**调度来解决分支问题

VLIW Datapath（ALU+MEM）



注意：两个通路的不对称性

优化5：基于VLIW的循环展开

Memory reference 1	*Memory reference 2*	*FP operation 1*	*FP op. 2*	*Int. op/ branch*	*Clock*
LD F0,0(R1)	LD F6,-8(R1)				1
LD **F10**,-16(R1)	LD F14,-24(R1)				2
LD F18,-32(R1)	LD F22,-40(R1)	ADDD F4,F0,F2	ADDD F8,F6,F2		3
LD F26,-48(R1)		ADDD F12,**F10,**F2	ADDD F16,F14,F2		4
		ADDD **F20**,F18,F2	ADDD F24,F22,F2		5
SD 0(R1),F4	SD -8(R1),F8	ADDD F28,F26,F2			6
SD -16(R1),F12	SD -24(R1),F16			SUBI R1,R1,#48	7
SD 16(R1),**F20**	SD 8(R1),F24				8
SD -0(R1),F28				BNEZ R1,LOOP	9

Unrolled 7 times to avoid delays

7 iterations in 9 clocks, or 1.3 clocks per iteration

23/9≈2.5 ops per clock, 50% efficiency

注: 在VLIW中，一条超长指令有更多的读写寄存器操作

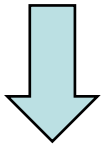
循环间相关


```
for (i=1; i<=100; i=i+1)
 {
        A[i] = A[i] + B[i];     /* S1 */
        B[i+1] = C[i] + D[i]; /* S2 */
 }
```

- S1使用由S2在前一次循环中计算的值。
- **循环间相关-循环内相关**
  - ✓ 这种存在于循环间的相关，我们称为 “loop-carried dependence”、“inter-loop dependence”。 这表示循环间存在相关，它与前面的例子中循环内部相关、循环间无关“intra-loop dependence”是有区别的

循环间转化为循环内


```
for (i=1; i<=100; i=i+1) {
        A[i] = A[i] + B[i];    /* S1 */
        B[i+1] = C[i] + D[i];} /* S2 */
```



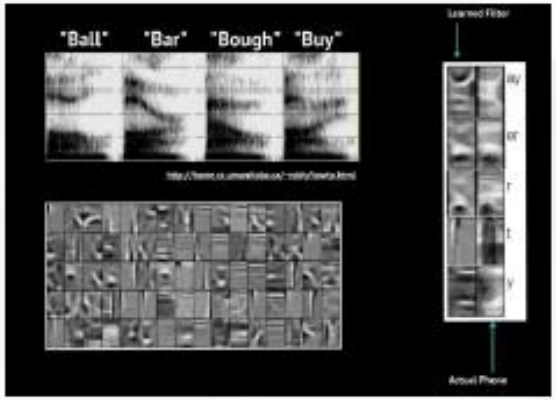
```
A[1] = A[1] + B[1];
for (i=1; i<=99; i=i+1) {
    B[i+1] = C[i] + D[i];
    A[i+1] = A[i+1] + B[i+1];
}
B[101] = C[100] + D[100];
```

采用循环展开、寄存器重命名等技术

实际的例子?

循环展开的例子-深度学习

人脸识别准确度
超过人类



讯飞语音速记战胜
人类专业速记员

Source: Cambricon

AlphaGo战胜
柯洁、李世乭

LeNet-5
5层CNN模型
用于手写字符识别
1998

AlexNet
ILSVRC 2012冠军
8层CNN
2012

ZFNet
ILSVRC 2013冠军
7层CNN
2013

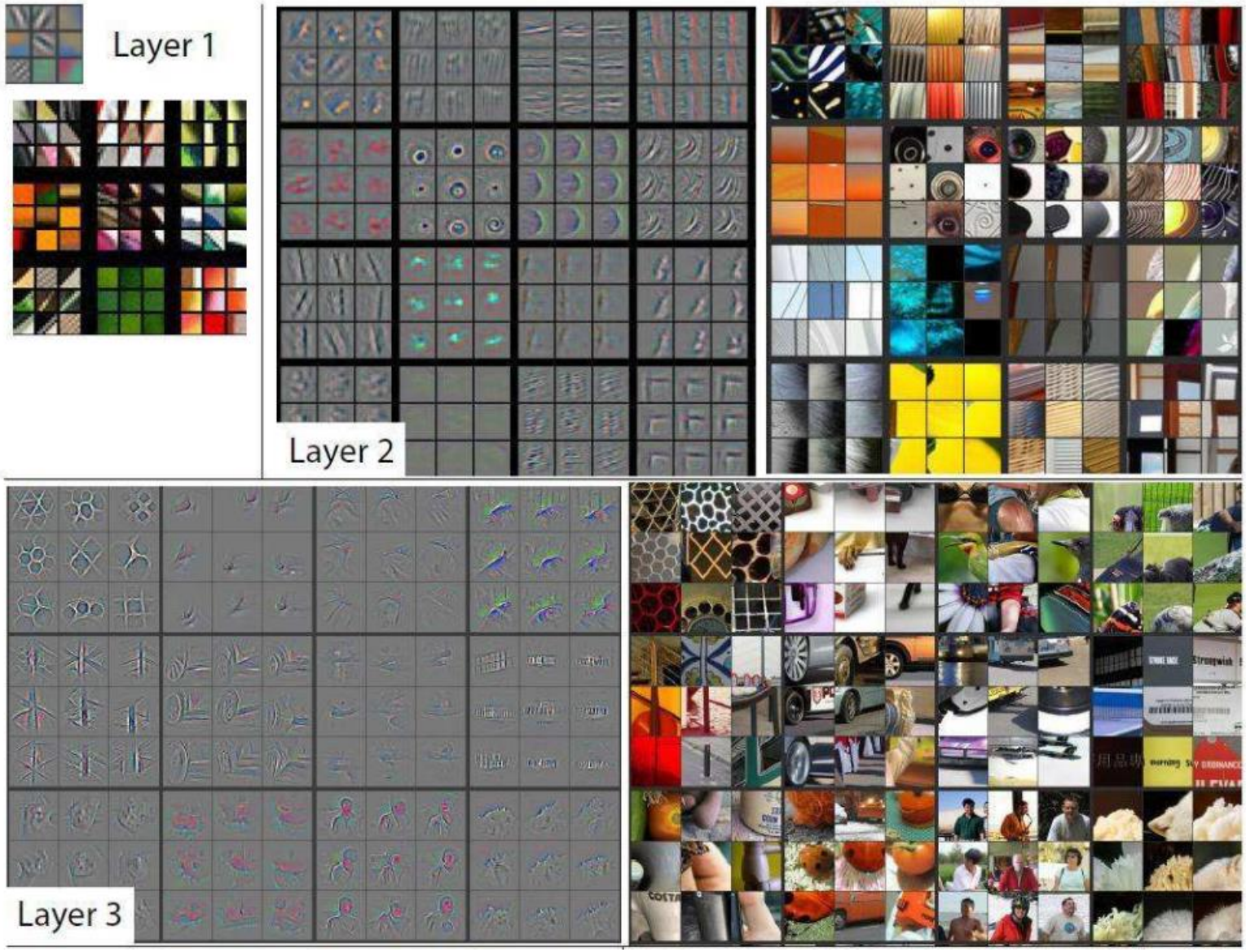
GoogleNet
ILSVRC 2014冠军
22层CNN

VGGNet
ILSVRC 2014
19层CNN
2014

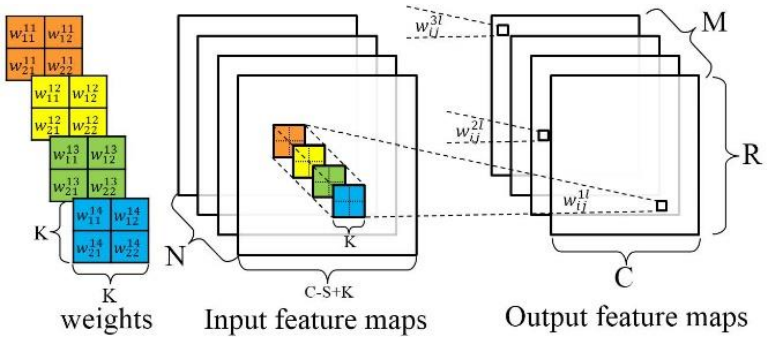
ResNet
ILSVRC 2015冠军
152层CNN
2015

SqueezeNet
AlexNet简化模型
权值<0.5MB
2017

深度学习是如何工作的?



循环展开的例子-卷积




```
for(row=0; row<R; row++) {
 for(col=0; col<C; col++) {
  for(to=0; to<M; to++) {
   for(ti=0; ti<N; ti++) {
    for(i=0; i<K; i++) {
     for(j=0; j<K; j++) {
   L:  output_fm[to][row][col] +=
          weights[to][ti][i][j]*
          input_fm[ti][S*row+i][S*col+j];
} } } } } }
```

参数解释：

- R：输出特征矩阵行数
- C：输出特征矩阵列数
- S：卷积的滑动步长
- M：输出特征矩阵通道数 & 卷积核个数
- N：输入特征矩阵通道数 & 各卷积核通道数
- K：卷积核行列数

相应的：

- 输入数据量=N*(C-S+K)(R-S+K)
- 输出数据量=M*R*C
- 权值数据量=M*N*K*K
- 总计算量=2*M*N*R*C*K*K

课后思考：对不同层的循环展开分别代表什么含义，应该如何优化？

小结

- 循环级并行的基本策略
  - ✓循环展开（手动or编译器）
  - ✓寄存器重命名（编译器）
  - ✓指令调度（编译器）
  - ✓超标量SuperScalar
  - ✓超长指令字VLIW
- 循环间并行（部分可以转化）
- 调研思考：结合Arch2030报告,对神经网络六层循环不同层的循环展开分别代表的含义，采用上述技术应该如何优化（软件+硬件）？

"study the past if you would define the future."

by Confucius