《计算机系统结构》课程直播

2020. 4.16

听不到声音请及时调试声音设备，可以下课后补签到

请将ZOOM名称改为“姓名”；

本次内容

- 解决数据冲突的具体实现
- 接下来内容预告
- 作业点评

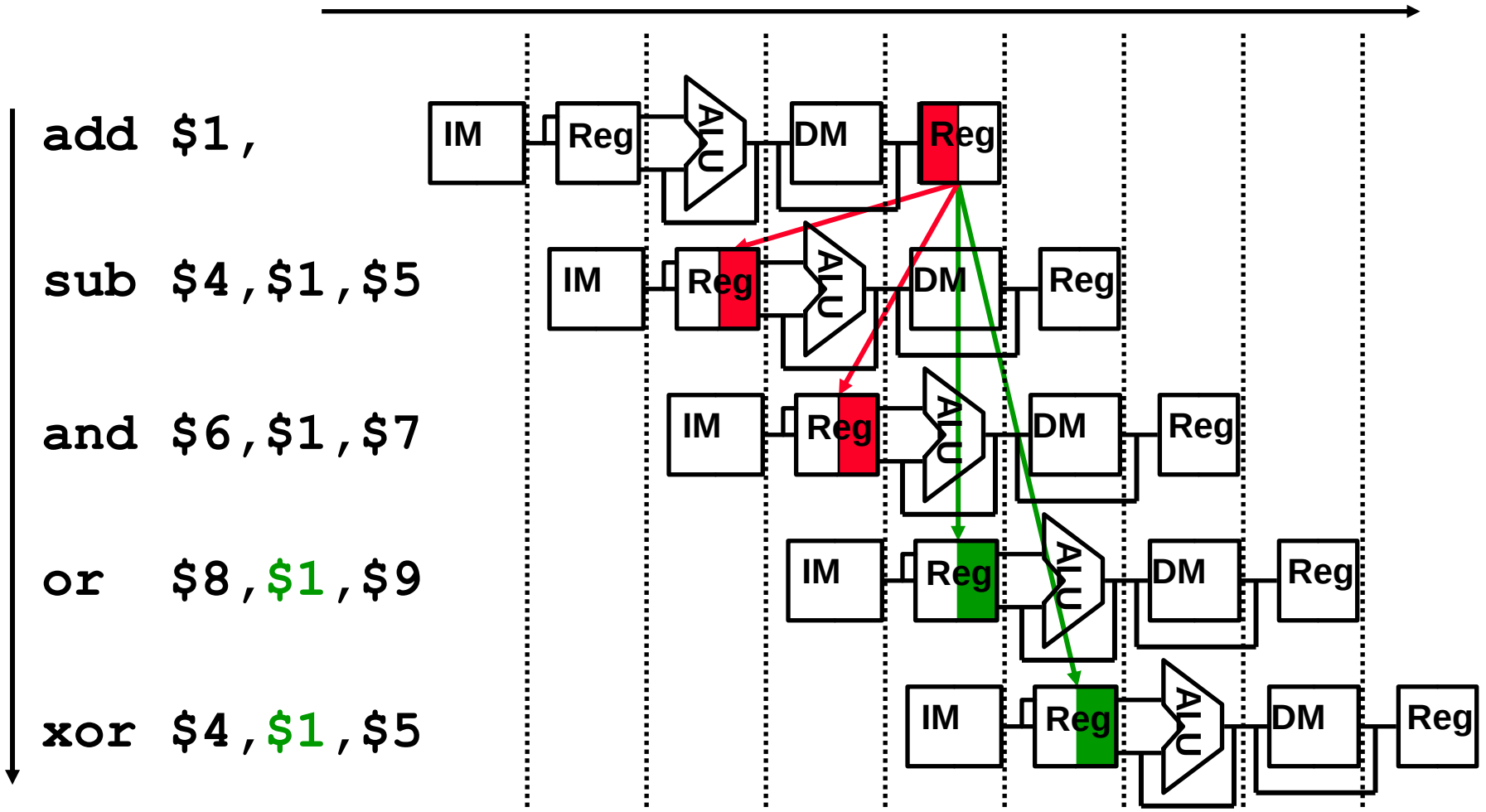
流水线中的相关性和“冒险”

- 流水线中的相关性：相邻或相近的指令之间因存在某种依赖性，或称为相关性，使得指令的执行可能受到影响。
- 这些相关性，可能会影响指令的执行，也可能不影响，因此又称为冒险（hazard）

  - 结构冒险 （ structural hazards ）
    - 资源相关性：所需的硬件部件正在为之前的指令工作
  - 数据冒险（ data hazards ）
    - 数据依赖性需要等待之前的指令完成数据的读写
  - 控制冒险（ control hazards ）
    - 转移指令引起：需要根据指令的结果决定下一步

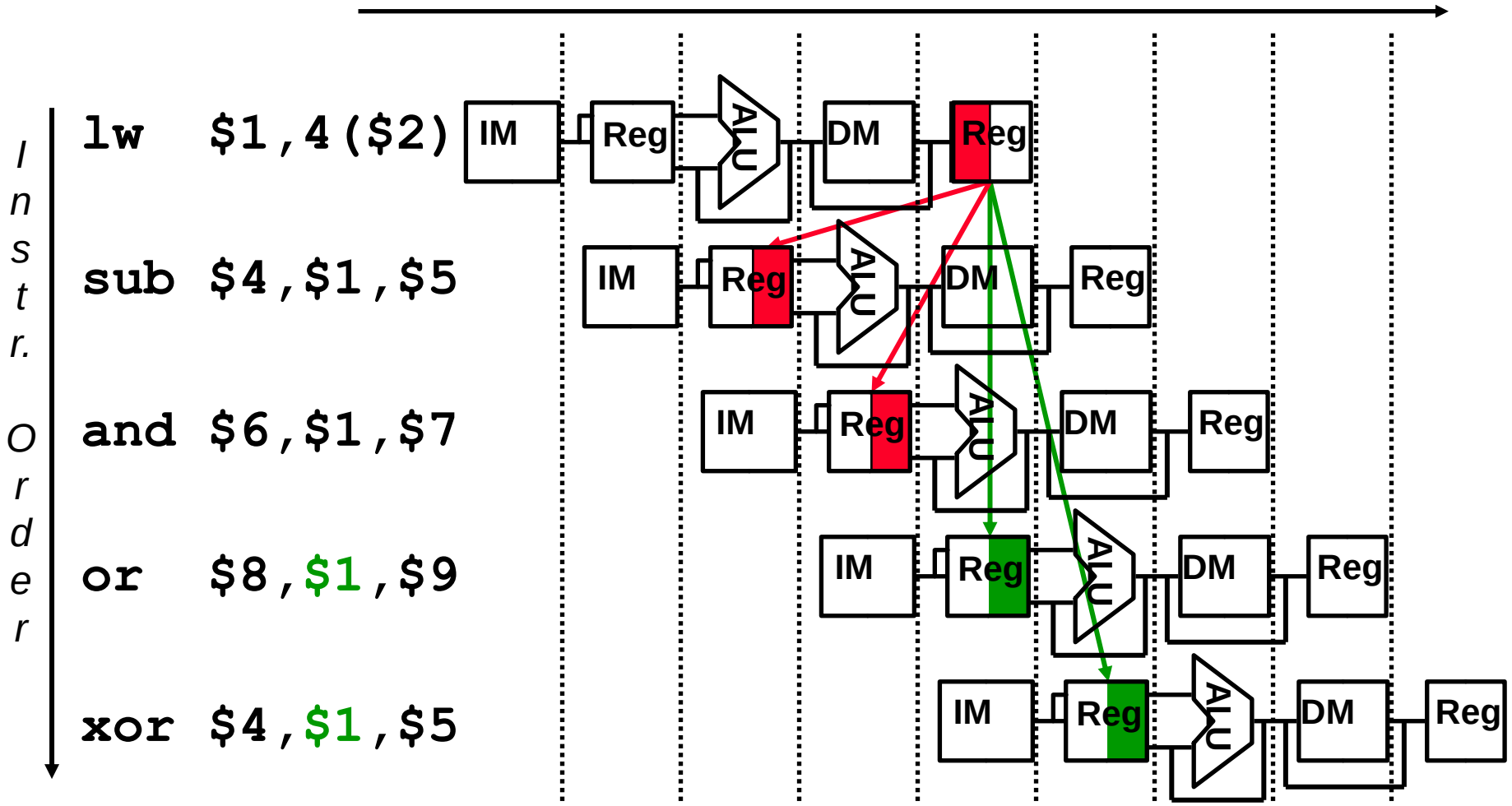
Register Usage Can Cause Data Hazards

- Dependencies backward in time cause hazards



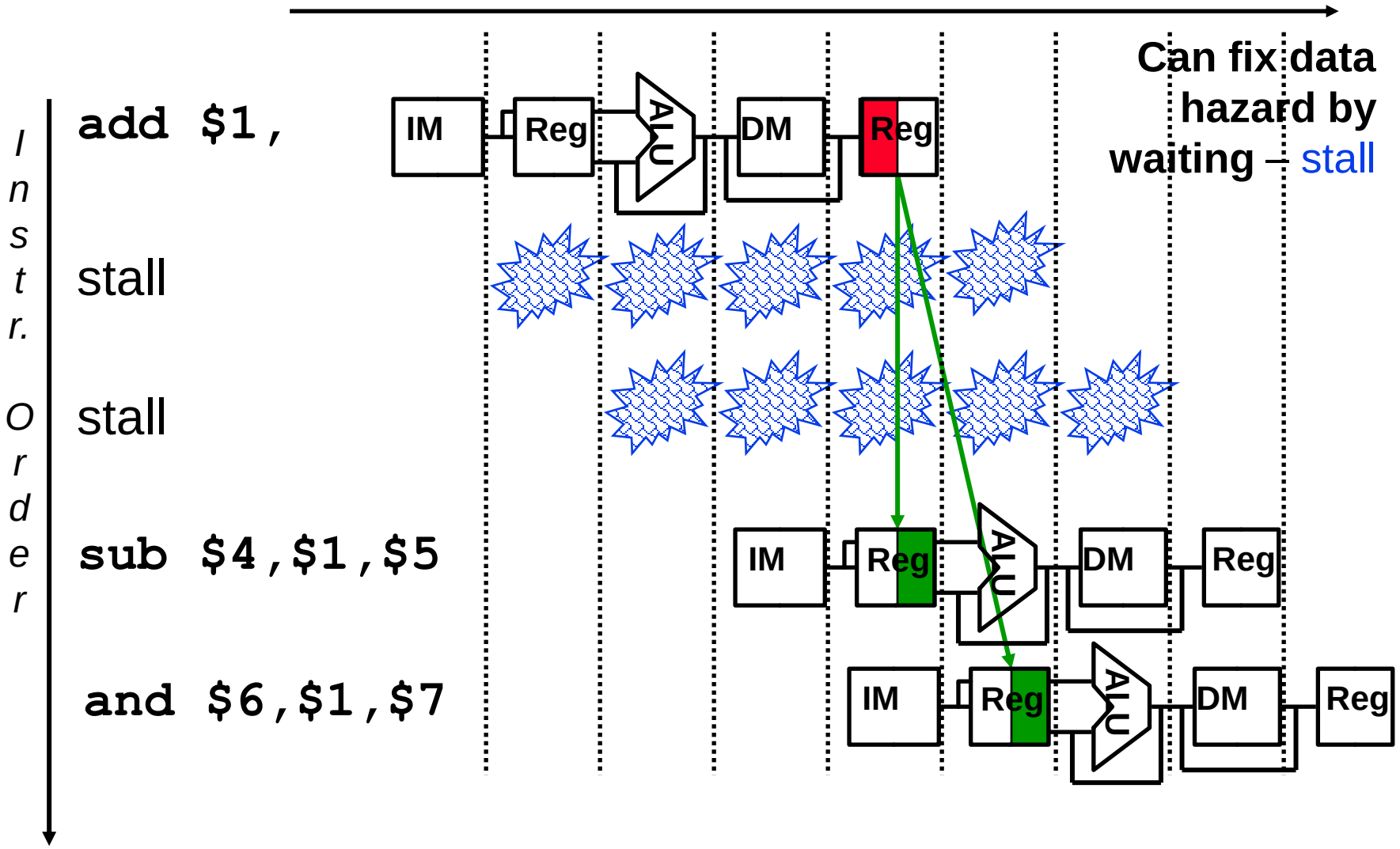
- Read before write data hazard

Loads Can Cause Data Hazards



- Load-use data hazard

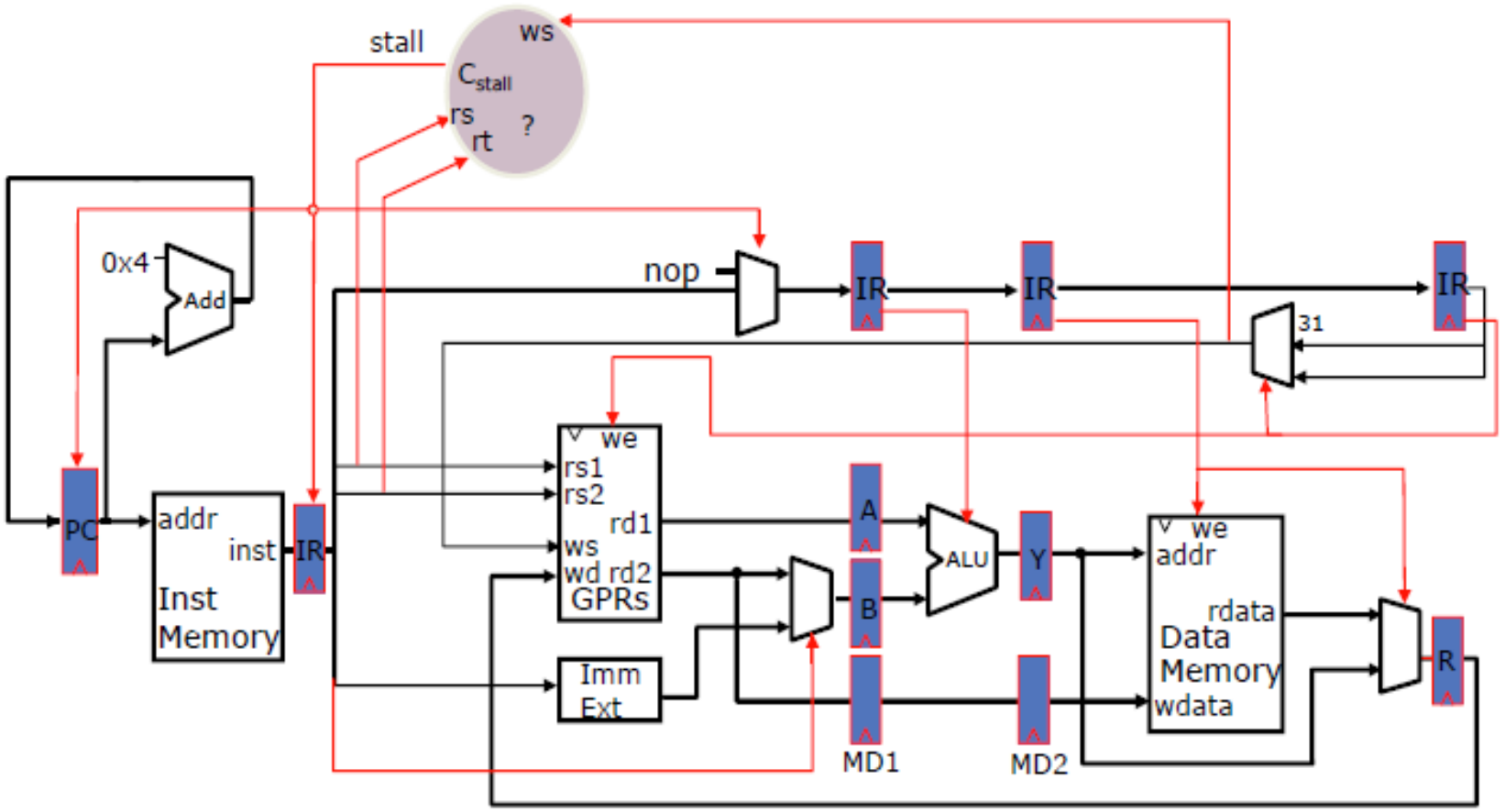
Resolving Data Hazards with Stalls



Stalled Stages and Pipeline Bubbles

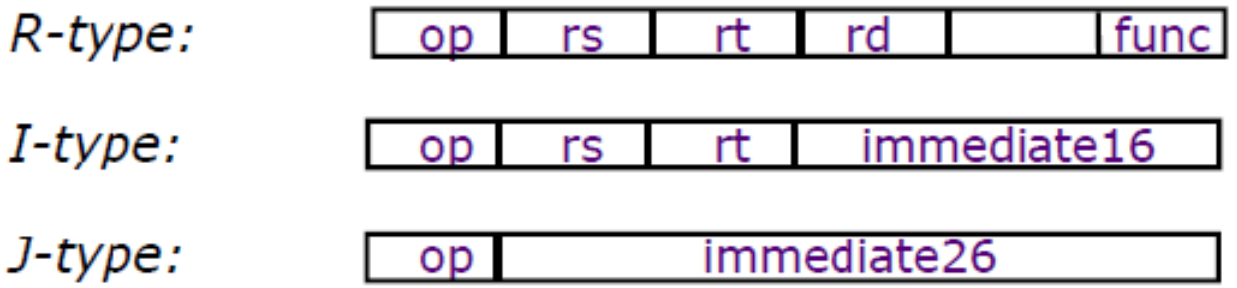
		t0	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10
I1	`add $1,`	IF1	ID1	EX1	MA1	WB1						
I2	`sub $4,$1,$5`		IF2	ID2	ID2	ID2	EX2	MA2	WB2			
I3	`and $6,$1,$7`			IF3	IF3	IF3	ID3	EX3	MA3	WB3		
I4	`or $8,$1,$9`			**stall stage**			IF4	ID4	EX4	MA4	WB4	
I5								IF5	ID5	EX5	MA5	WB5

Stall Control Logic



Compare the source registers of the instruction in the decode stage with the destination register of the uncommitted instructions.

Source & Destination Registers



		source(s)	*destination*
ALU	rd ← (rs) func (rt)	rs, rt	rd
ALUI	rt ← (rs) op immediate	rs	rt
LW	rt ← M [(rs) + immediate]	rs	rt
SW	M [(rs) + immediate] ← (rt)	rs, rt	
BZ	*cond* (rs)		
	true: PC ← (PC) + immediate	rs	
	false: PC ← (PC) + 4	rs	
J	PC ← (PC) + immediate		
JAL	r31 ← (PC), PC ← (PC) + immediate		31
JR	PC ← (rs)	rs	
JALR	r31 ← (PC), PC ← (rs)	rs	31

Deriving the Stall Signal

C_{dest}

ws = *Case* opcode

ALU	⇒ rd
ALUi, LW	⇒ rt
JAL, JALR	⇒ R31

we = *Case* opcode

ALU, ALUi, LW	⇒ (ws ≠ 0)
JAL, JALR	⇒ on
...	⇒ off

C_{re}

re1 = *Case* opcode

ALU, ALUi, LW, SW, BZ, JR, JALR	⇒ on
J, JAL	⇒ off

re2 = *Case* opcode

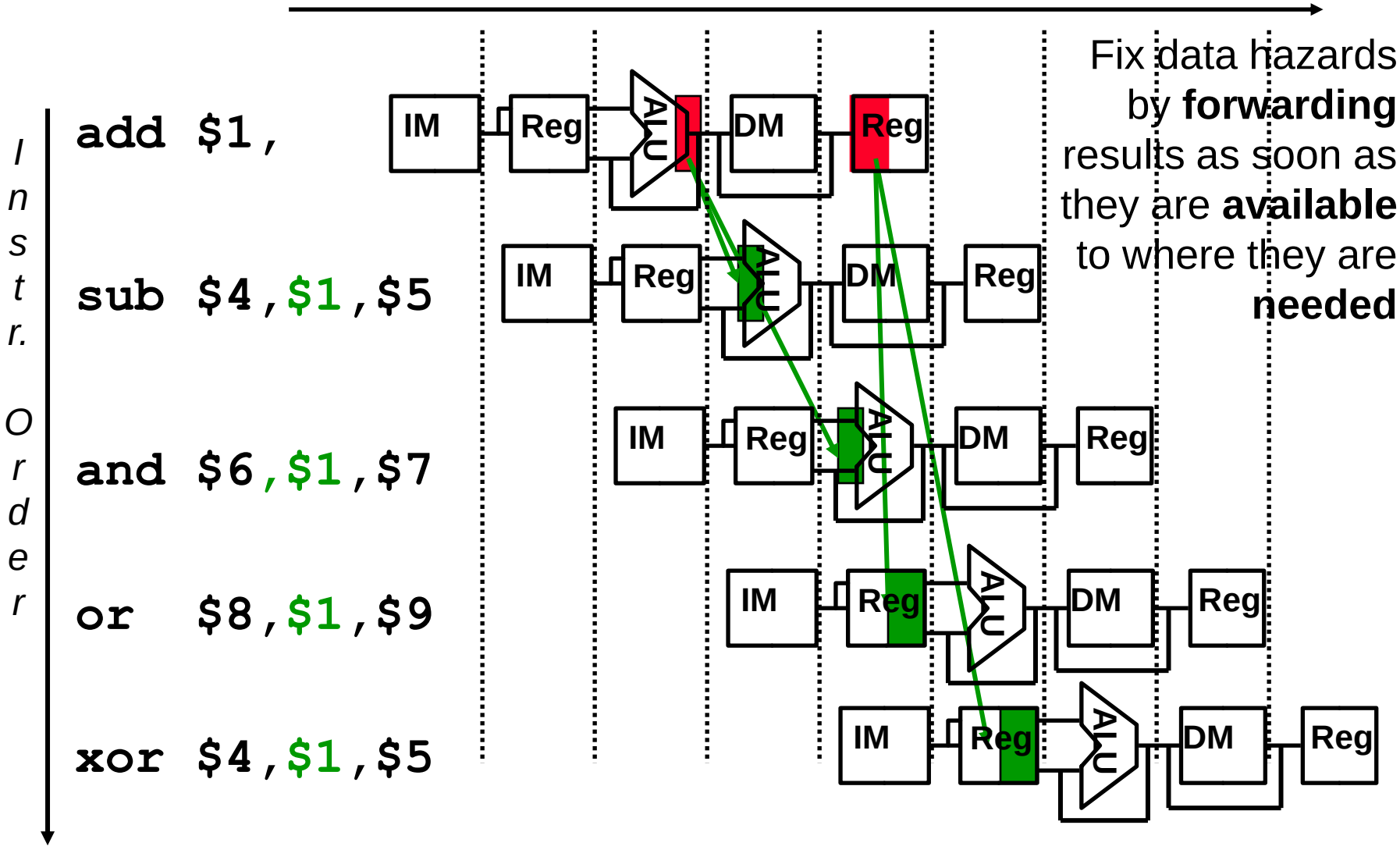
ALU, SW	⇒ on
...	⇒ off

C_{stall}

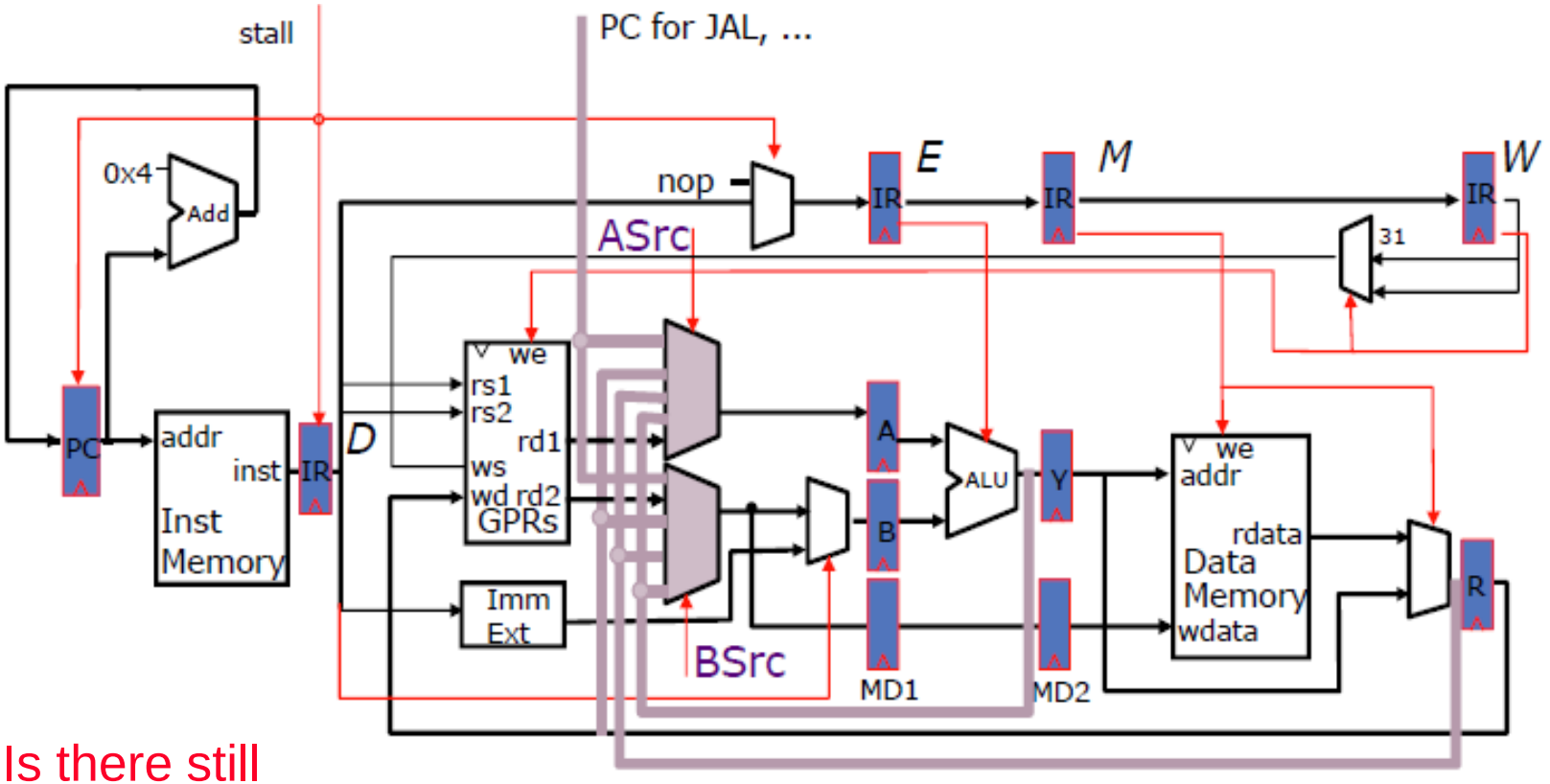
$$stall = ((rs_D = ws_E).we_E + (rs_D = ws_M).we_M + (rs_D = ws_W).we_W) \cdot re1_D + ((rt_D = ws_E).we_E + (rt_D = ws_M).we_M + (rt_D = ws_W).we_W) \cdot re2_D$$

This is not the full story !

Another Way to "Fix" a Data Hazard



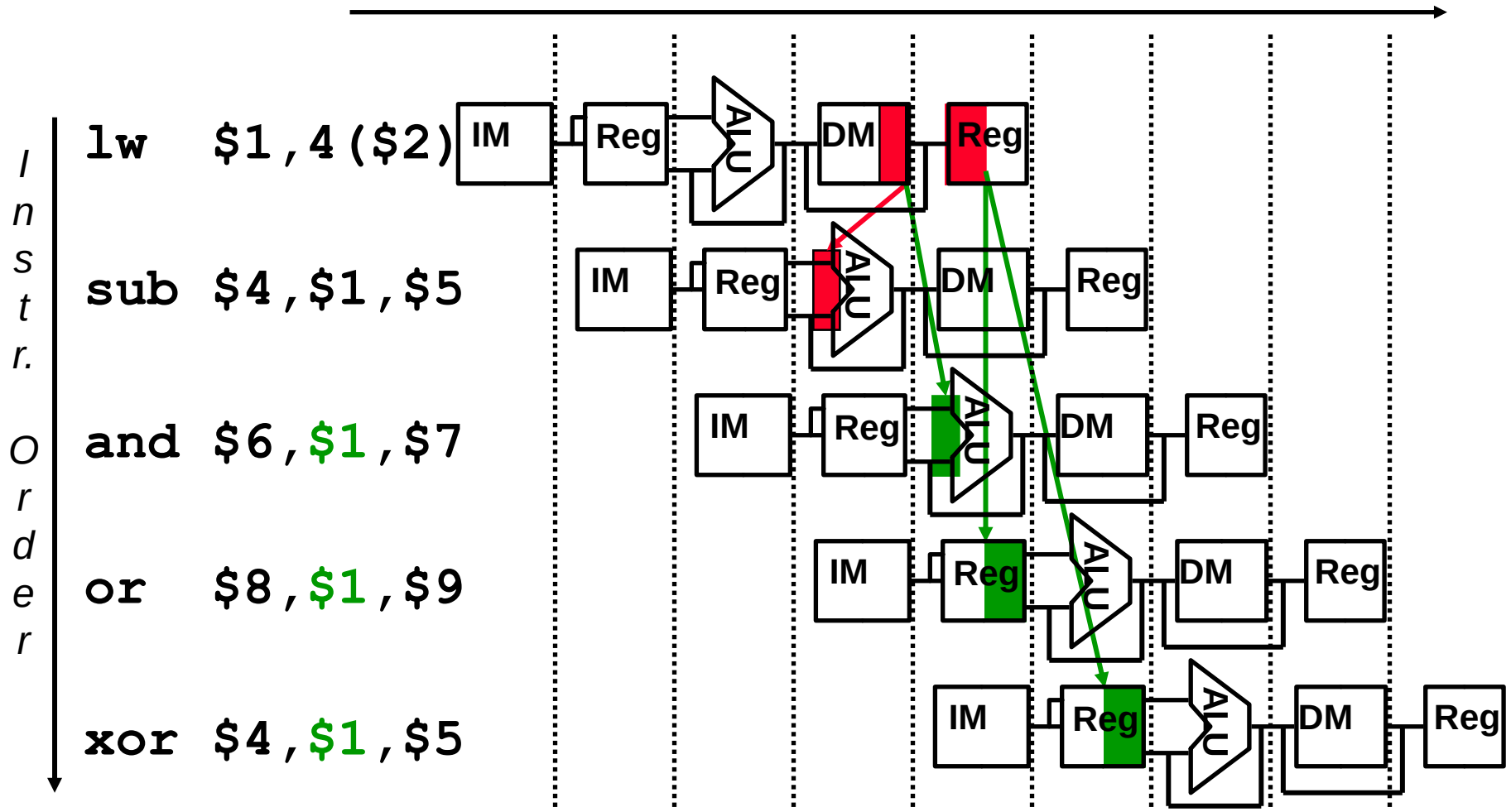
Fully Bypassed Datapath



Is there still a need for the stall signal ?

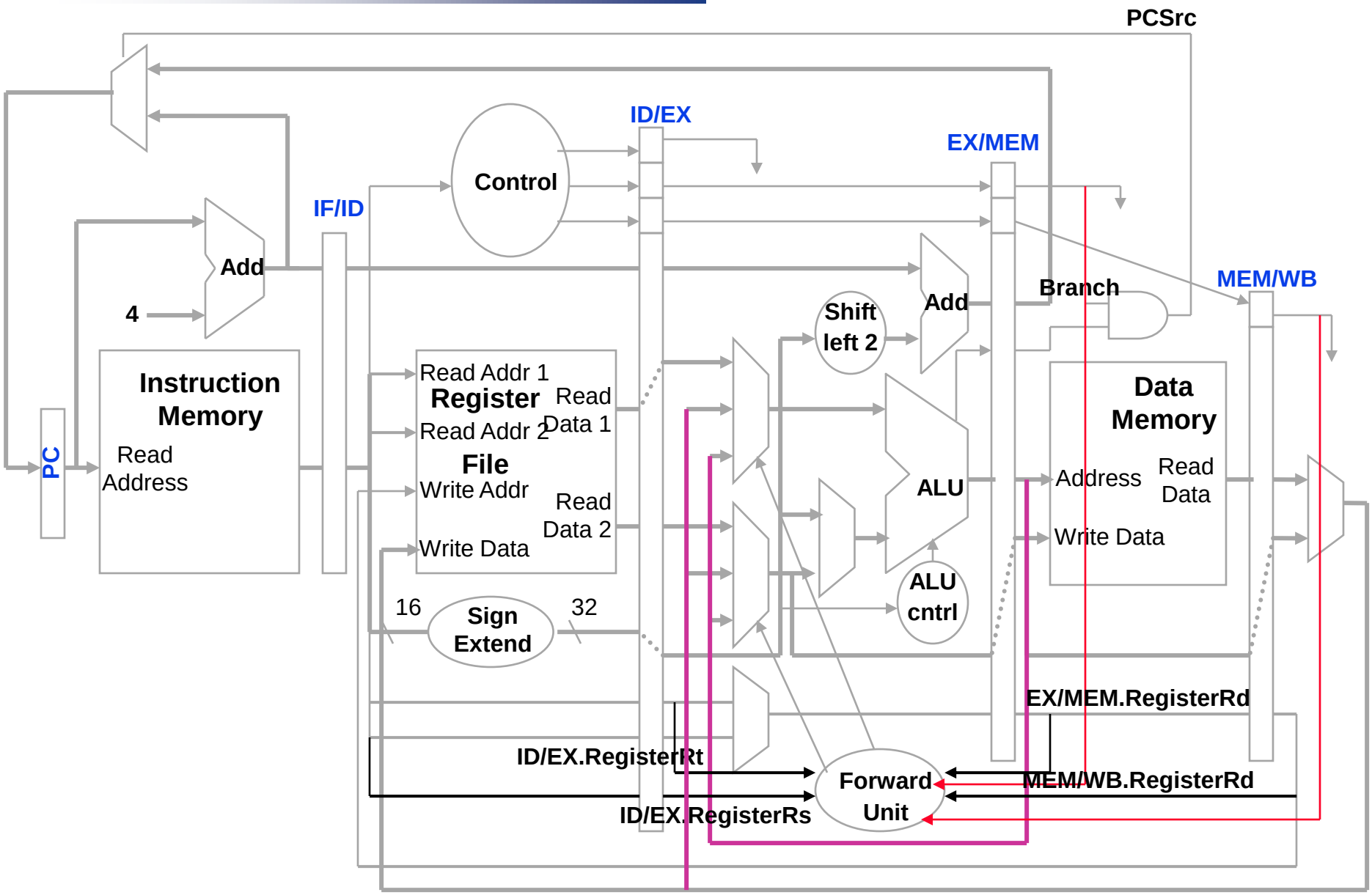
$$\text{stall} = (rs_D = ws_E).(opcode_E = LW_E).(ws_E \neq 0).re1_D + (rt_D = ws_E).(opcode_E = LW_E).(ws_E \neq 0).re2_D$$

Forwarding with Load-use Data Hazards



- Will still need one stall cycle even with forwarding

Datapath with Forwarding Hardware



Corrected Data Forwarding Control Conditions

1. EX Forward Unit:

```
if (EX/MEM.RegWrite
and (EX/MEM.RegisterRd != 0)
and (EX/MEM.RegisterRd = ID/EX.RegisterRs))
      ForwardA = 10
if (EX/MEM.RegWrite
and (EX/MEM.RegisterRd != 0)
and (EX/MEM.RegisterRd = ID/EX.RegisterRt))
      ForwardB = 10
```

Forwards the result from the previous instr. to either input of the ALU

2. MEM Forward Unit:

```
if (MEM/WB.RegWrite
and (MEM/WB.RegisterRd != 0)
and (EX/MEM.RegisterRd != ID/EX.RegisterRs)
and (MEM/WB.RegisterRd = ID/EX.RegisterRs))
      ForwardA = 01

if (MEM/WB.RegWrite
and (MEM/WB.RegisterRd != 0)
and (EX/MEM.RegisterRd != ID/EX.RegisterRt)
and (MEM/WB.RegisterRd = ID/EX.RegisterRt))
      ForwardB = 01
```

Forwards the result from the previous or second previous instr. to either input of the ALU

Pipelining Summary

- All modern day processors use pipelining
- Pipelining doesn't help latency of single task, it helps throughput of entire workload
- Potential speedup: a really fast clock cycle and able to complete one instruction every clock cycle (CPI)
- Pipeline rate limited by slowest pipeline stage
  - Unbalanced pipe stages makes for inefficiencies
  - The time to "fill" pipeline and time to "drain" it can impact speedup for deep pipelines and short code runs
- Must detect and resolve hazards
  - Stalling negatively affects CPI (makes CPI greater than the ideal of 1)

流水线处理器的性能

- CPU执行时间 =指令数目 x 平均每条指令所花的时钟周期（CPI）
  x 一个时钟周期长度

- $CPI_{ideal}=1$
- $CPI = CPI_{ideal} + CPI_{stall}$
- CPI_{stall} 发生的原因
  - 数据冒险
  - 结构冒险
  - 控制冒险
  - 访存延迟

接下来…

现代处理器的微结构

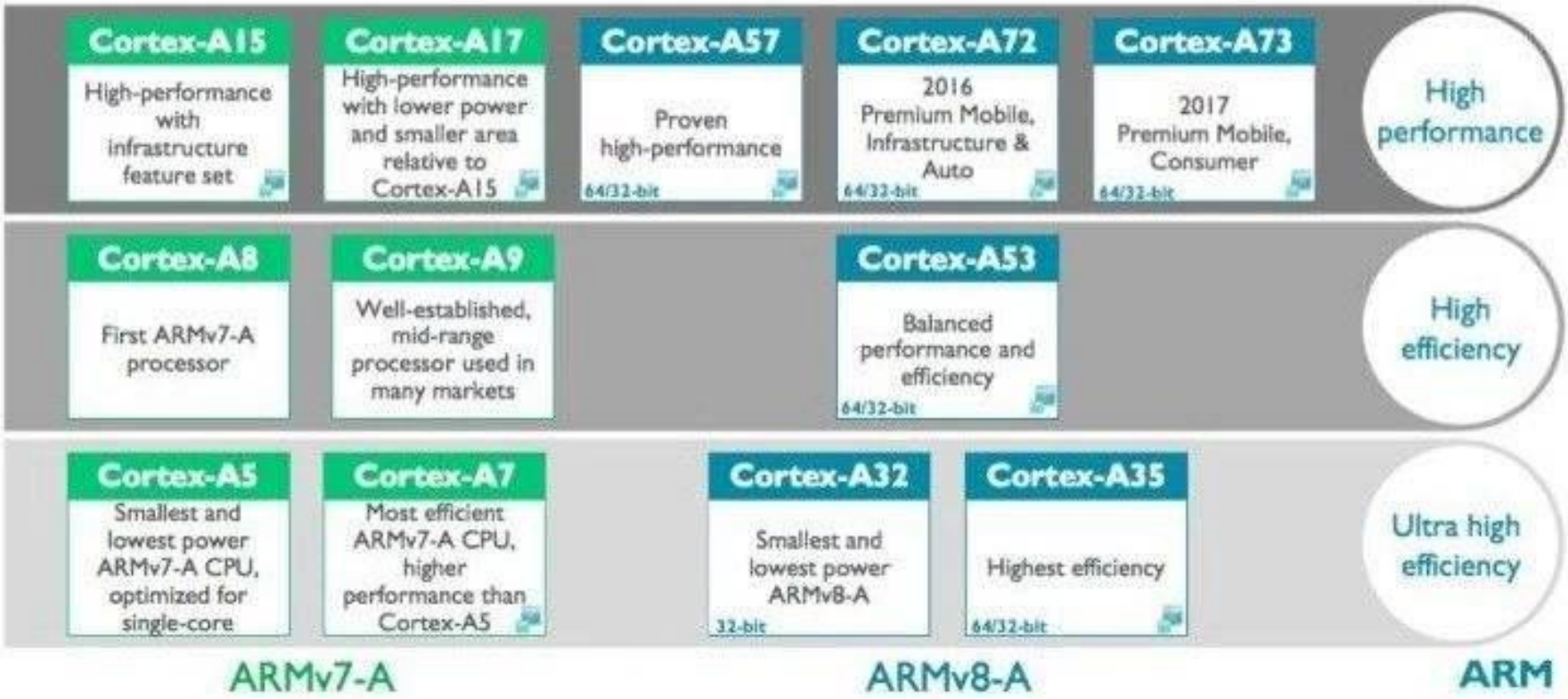
处理器	年份	时钟频率	流水级数	**发射宽度**	乱序执行?	核数	功耗
Intel 486	1989	25 MHz	5	1	No	1	5 W
Intel Pentium	1993	66 MHz	5	2	No	1	10 W
Intel Pentium Pro	1997	200 MHz	10	3	Yes	1	29 W
Intel Pentium 4 Willamette	2001	2000 MHz	22	3	Yes	1	75 W
Intel Pentium 4 Prescott	2004	3600 MHz	31	3	Yes	1	103 W
Intel Core	2006	2930 MHz	14	4	Yes	2	75 W
Sun USPARC III	2003	1950 MHz	14	4	No	1	90 W
Sun T1 (Niagara)	2005	1200 MHz	6	1	No	8	70 W

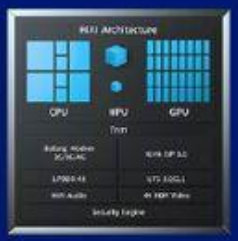
接下来…

多发射处理器的实现和主要特点

常用名	发射结构	冲突检测	调度方式	特点	处理器举例
静态超标量 superscalar (static)	动态	硬件	静态	按序执行	大部分嵌入式处理器，例如ARM cortex-A8
动态超标量 superscalar (dynamic)	动态	硬件	动态	乱序执行	目前无
推测执行超标量 superscalar (speculative)	动态	硬件	带推测的动态	乱序、推测执行	大部分通用处理器，如Intel Core i3,i5,i7、arm A76
超长指令字 (VLIW)	静态	主要由软件完成	静态	编译器（隐式）完成冲突检测、指令调度	某些特定领域,如信号处理器 TI C6x
显式并发指令运算(EPIC)	主要为静态	主要由软件完成	主要为静态	编译器（显式）完成冲突检测、指令调度	**Intel** 安腾 **Itanium**处理器

ARM Cortex-A系列



	麒麟990 5G	麒麟990
		
工艺	7nm+EUV	7nm
CPU	2X Cortex-A76 Based@2.86GHz 2X Cortex-A76 Based@2.36GHz 4X Cortex-A55@1.95GHz	2X Cortex-A76 Based@2.86GHz 2X Cortex-A76 Based@2.09GHz 4X Cortex-A55@1.86GHz
GPU	16 Core Mali-G76	16 Core Mali-G76
NPU	2 Big Core +1 Tiny Core	1 Big Core +1 Tiny Core
存储	UFS 3.0, UFS2.1	UFS 3.0, UFS2.1
Modem	2G/3G/4G/5G	2G/3G/4G

华为MATE30手机中的处理器

CPU：

2大核+2中核+4小核能效架构

2×A76(2.86Ghz)、

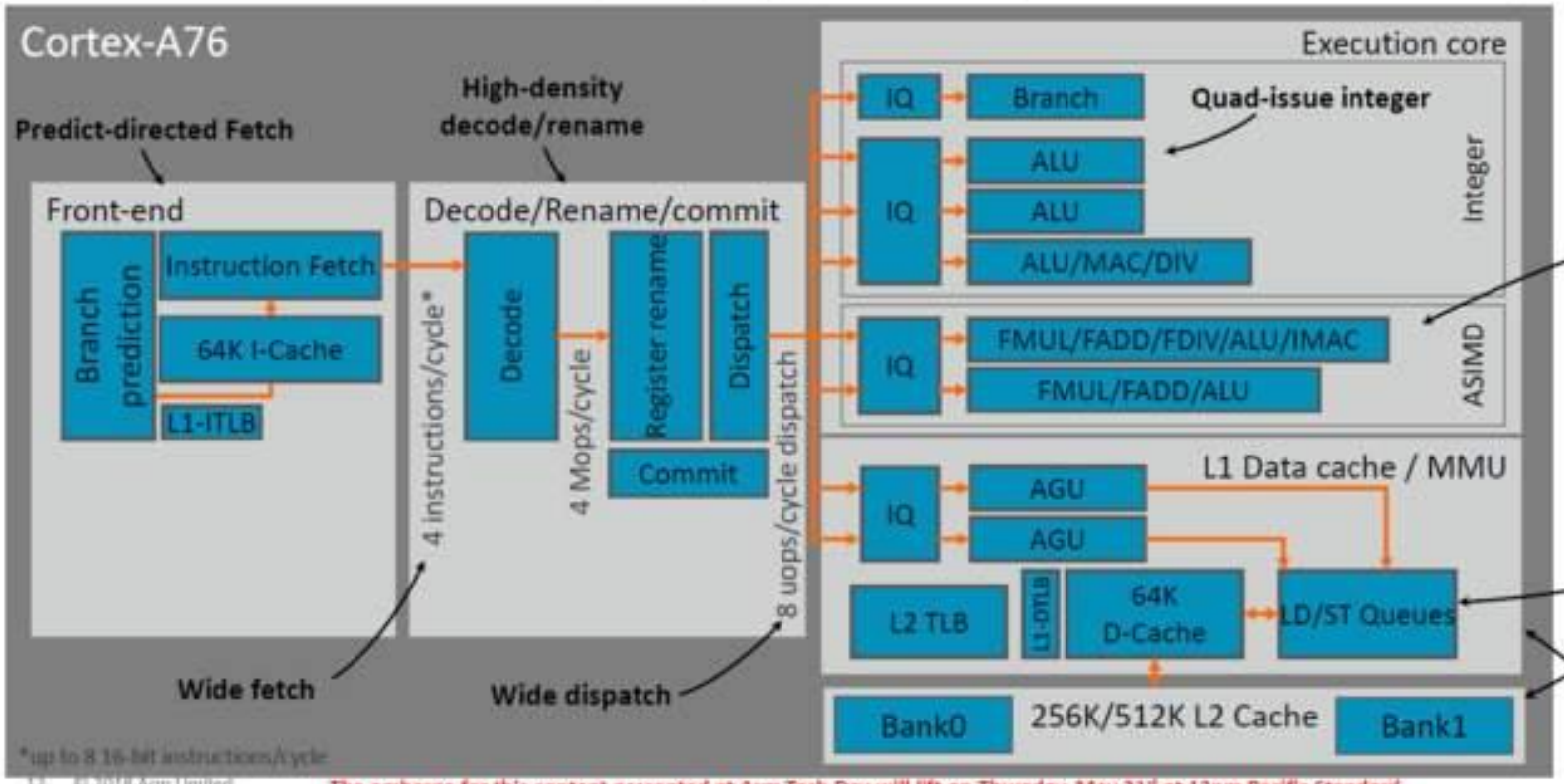
2×A76(2.36Ghz)、

4×A55(1.95Ghz）

麒麟990对标骁龙855+的性能

Cortex-A76: Microarchitecture overview

The foundation of a new family of high-performance products



arm

作业讲解

习题：

考虑以下结构体：

```
typedef struct
 {      char a[3];
        short b[3];
        double c;
        long double d;
        int* e;
        int f; } foo;
```

- x86-64 Linux 系统如何为foo分配内存？用变量的名字标记分配给该变量的字节， 用X 标记用于内存对齐的空白字节。例如: aXXXbbbb, 表示分配给char a一个字节，分配给int b四个字节，a和b之间有三个空白字节是用于内存对齐。注意: long double 长度为16 bytes.

习题

Cache Friendly Code: 下面考虑访问数组A时，不同高速缓存的性能问题。假设其他变量都放置在寄存器中，运行代码之前cache为空. 考虑以下代码:

```
#define N 128
int myst(int[ ] A) {
        int i, result;
        for (i = 0; i < N; i++) result += A[i]*A[N-i-1];
        return result;
}
```

假设有一个大小为64-byte, 一共有4个组（sets）的直接映射高速缓存。当代码执行到返回result的结果时，cache中的内容是什么？补充填写图中的内容。图中一个方块格子代表4个字节。

```
#define N 128
int myst(int[ ] A) {
        int i, result;
        for (i = 0; i < N; i++) result += A[i]*A[N-i-1];
        return result;
}
```

假设有一个大小为64-byte, 一共有4个组（sets）的直接映射高速缓存。当代码执行到返回result的结果时，cache中的内容是什么？补充填写图中的内容。图中一个方块格子代表4个字节。

```
#define N 128
int myst(int[ ] A) {
        int i, result;
        for (i = 0; i < N; i++) result += A[i]*A[N-i-1];
        return result;
}
```

- 假设有一个大小为64-byte, 一共有4个组（sets）的2路组相联映射高速缓存（two-way set associative cache with 4 sets当代码执行到返回result的结果时，cache中的内容是什么？补充填写图中的内容。图中一个方块格子代表4个字节。

再见