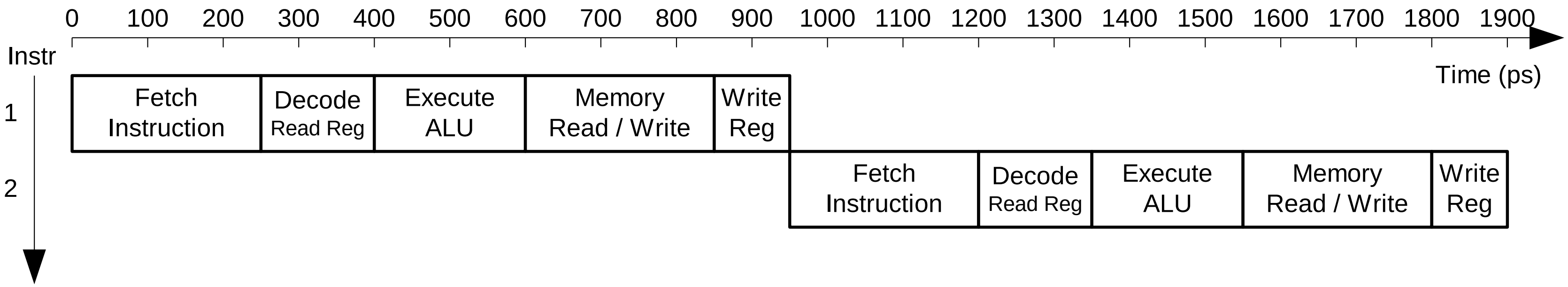
# Pipelined MIPS Processor

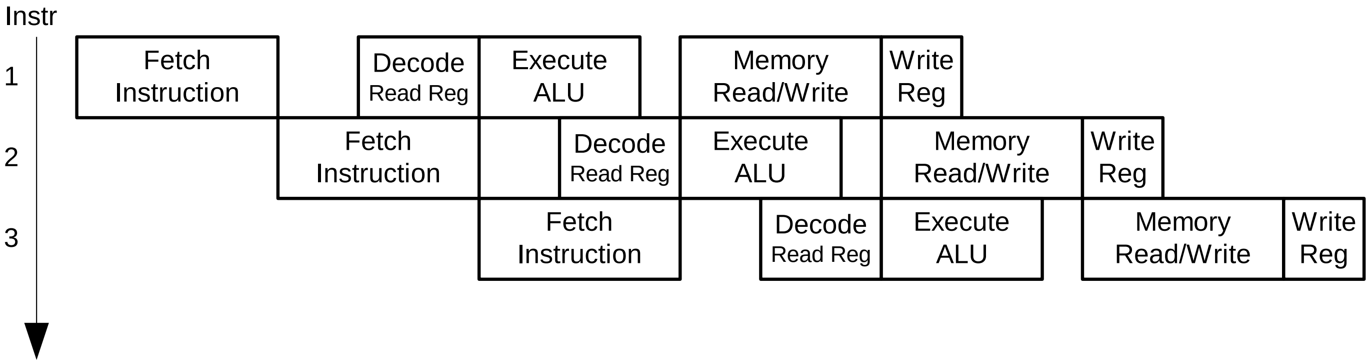
- Temporal parallelism
- Divide single-cycle processor into 5 stages:
  - Fetch
  - Decode
  - Execute
  - Memory
  - Writeback
- Add pipeline registers between stages

# Single-Cycle vs. Pipelined

## Single-Cycle

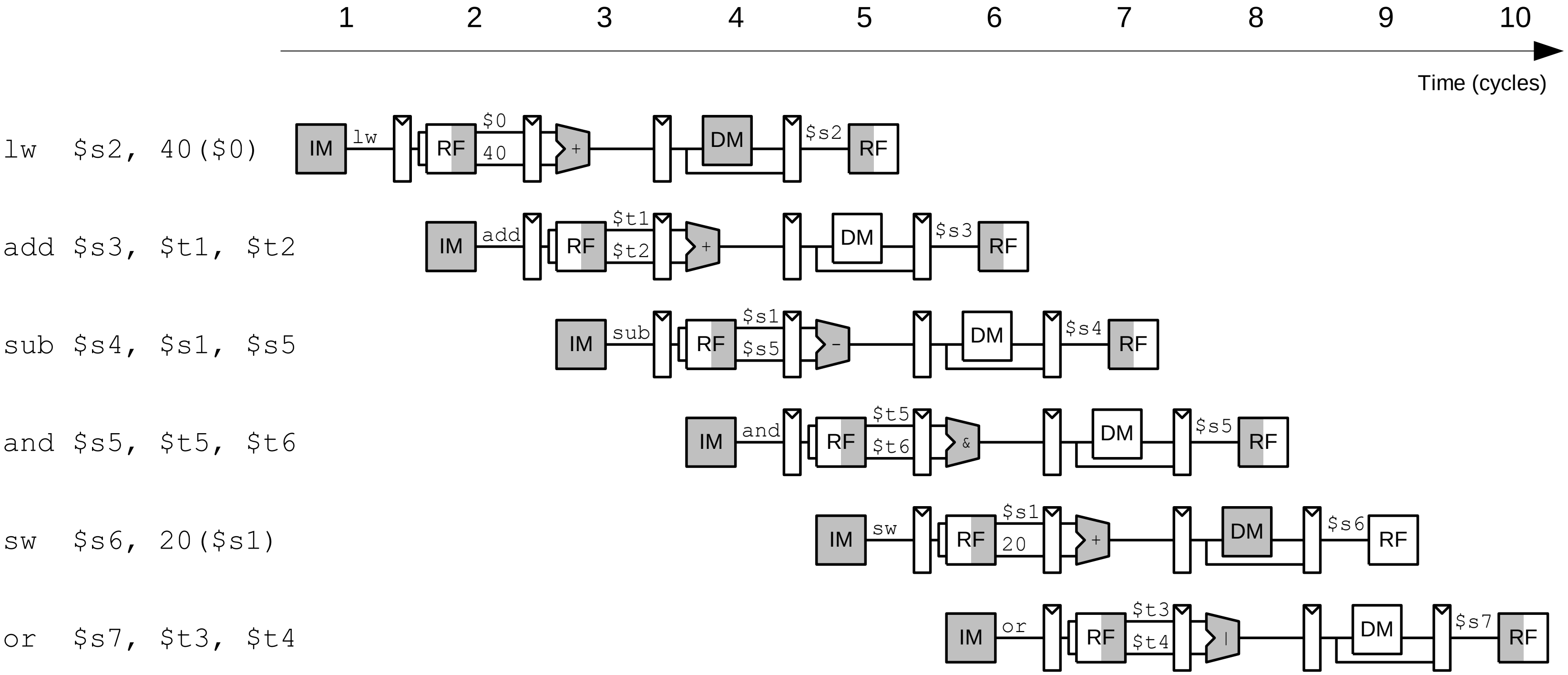


## Pipelined



ELSEVIER

# Pipelined Processor Abstraction



```
lw   $s2, 40($0)
add  $s3, $t1, $t2
sub  $s4, $s1, $s5
and  $s5, $t5, $t6
sw   $s6, 20($s1)
or   $s7, $t3, $t4
```

# Pipelined & RTL

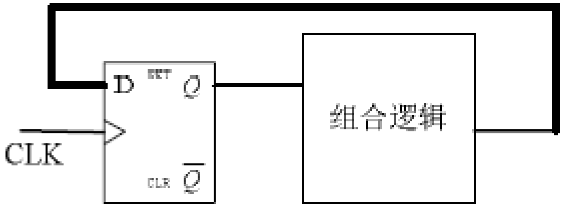


图 4-2 状态机的简单模型

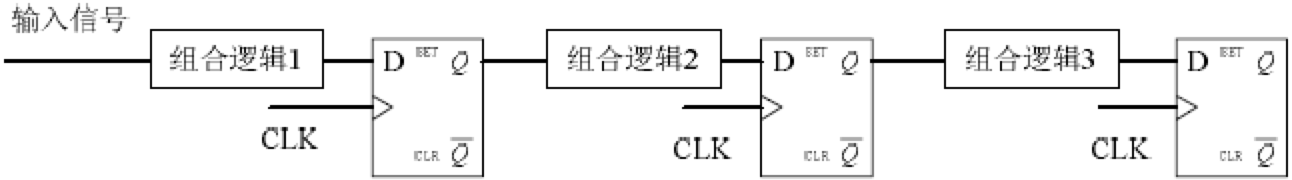


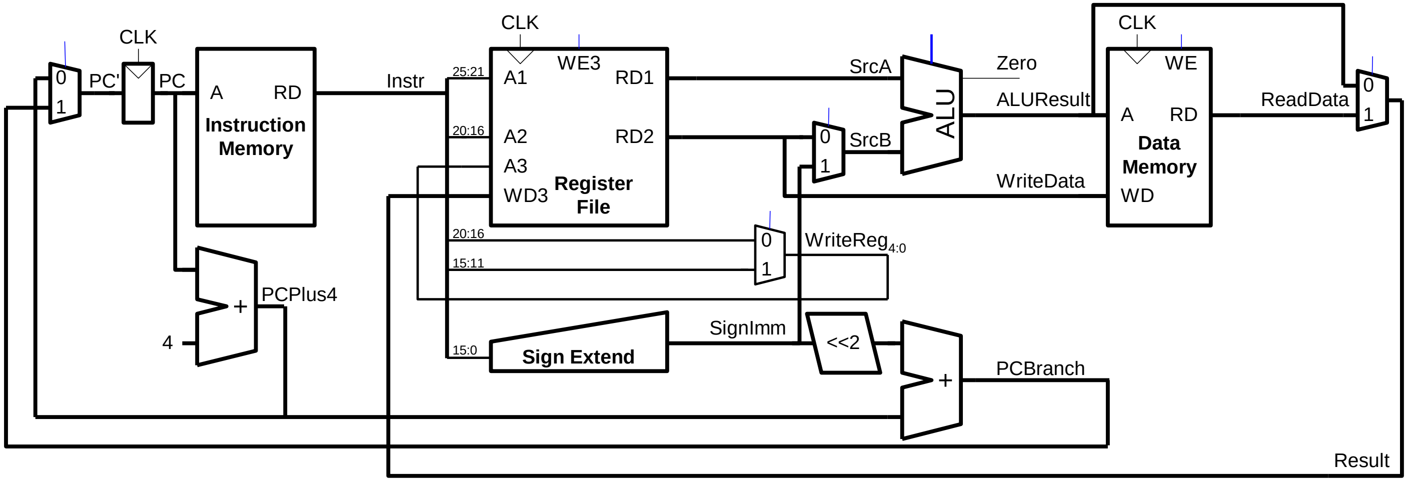
图 4-3 流水线的简单模型

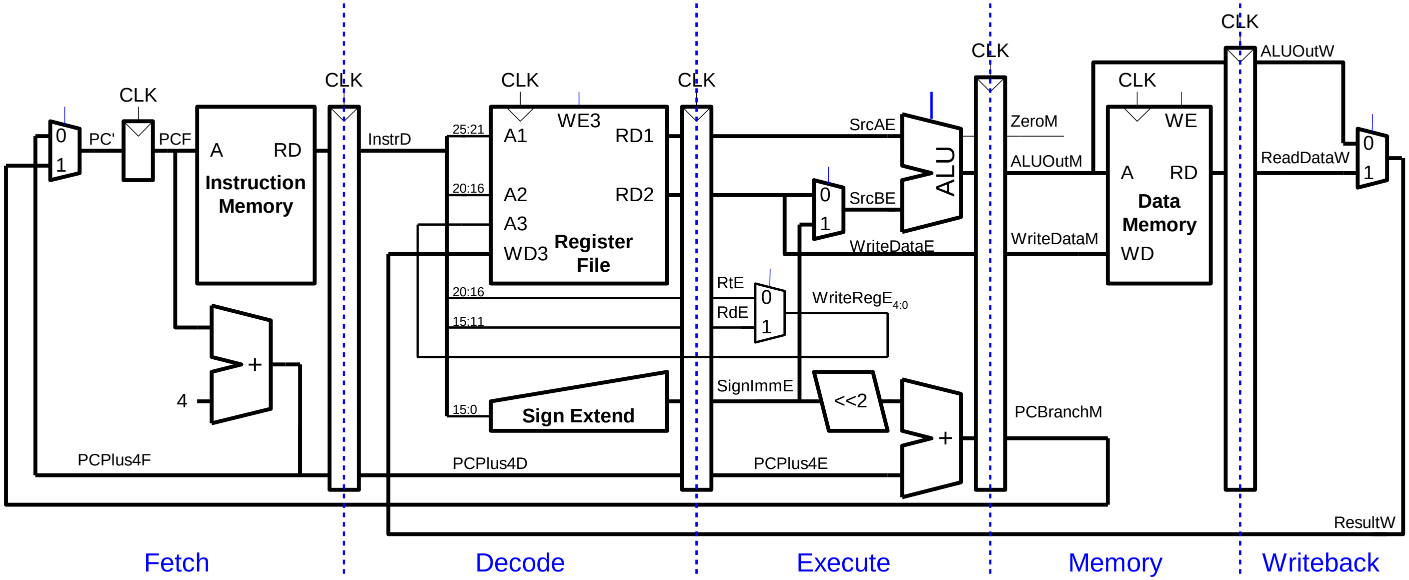
http://blog.csdn.net/leishangwen

RTL: Register Transfer Level

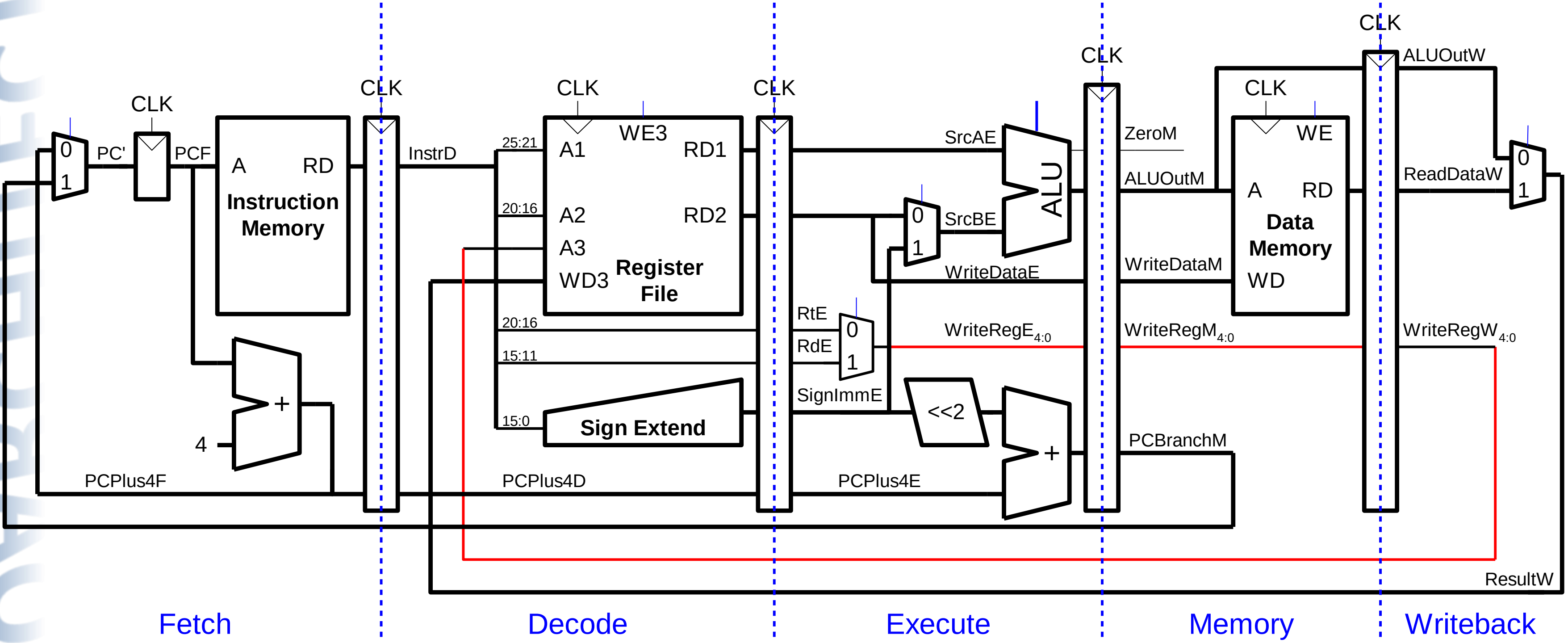
The signal translate between registers, and every stage occurs their change for combined logic

# Single-Cycle & Pipelined Datapath





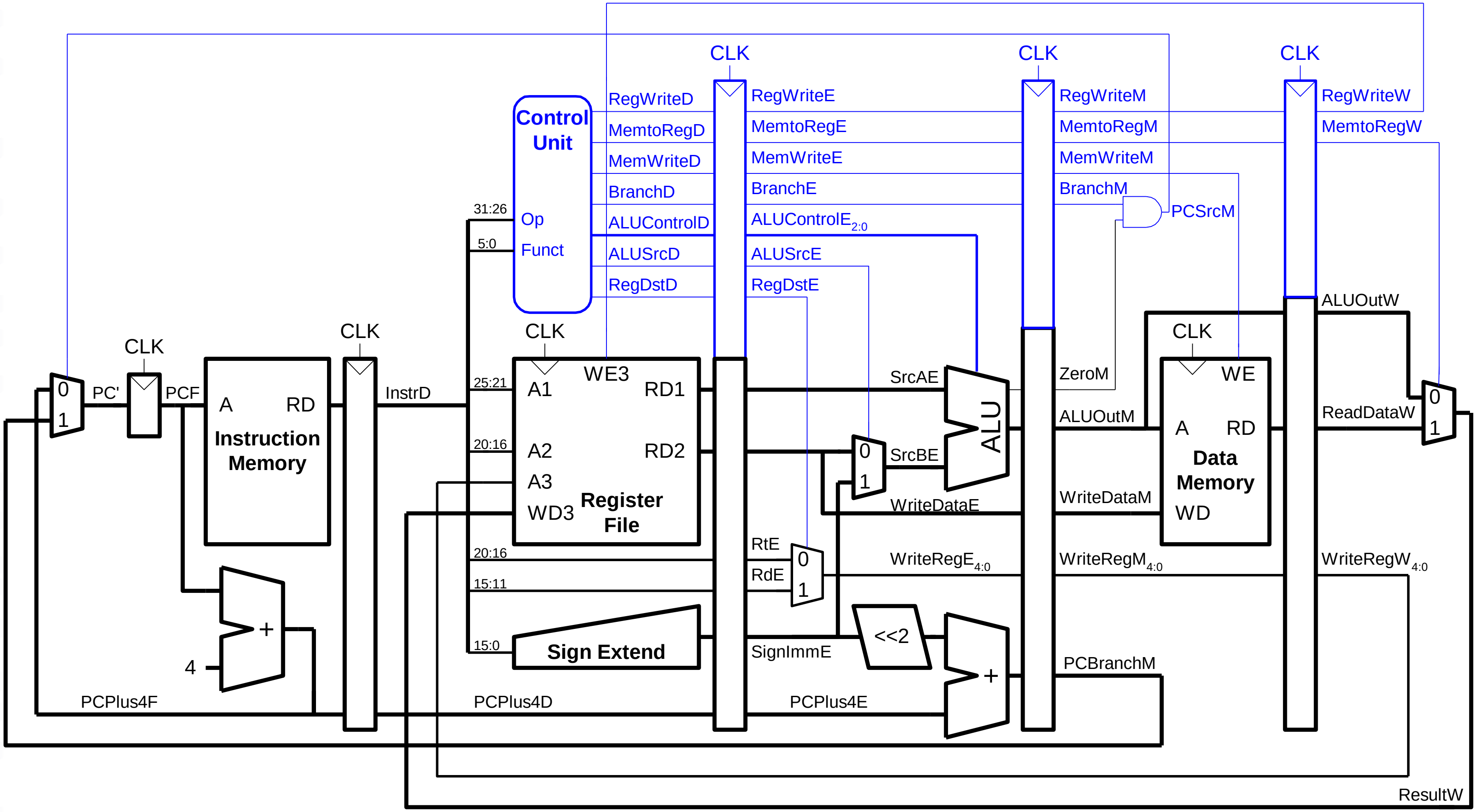
# Corrected Pipelined Datapath



***WriteRegW* must arrive at same time as *ResultW***

ELSEVIER

# Pipelined Processor Control



- Same control unit as single-cycle processor
- Control delayed to proper pipeline stage

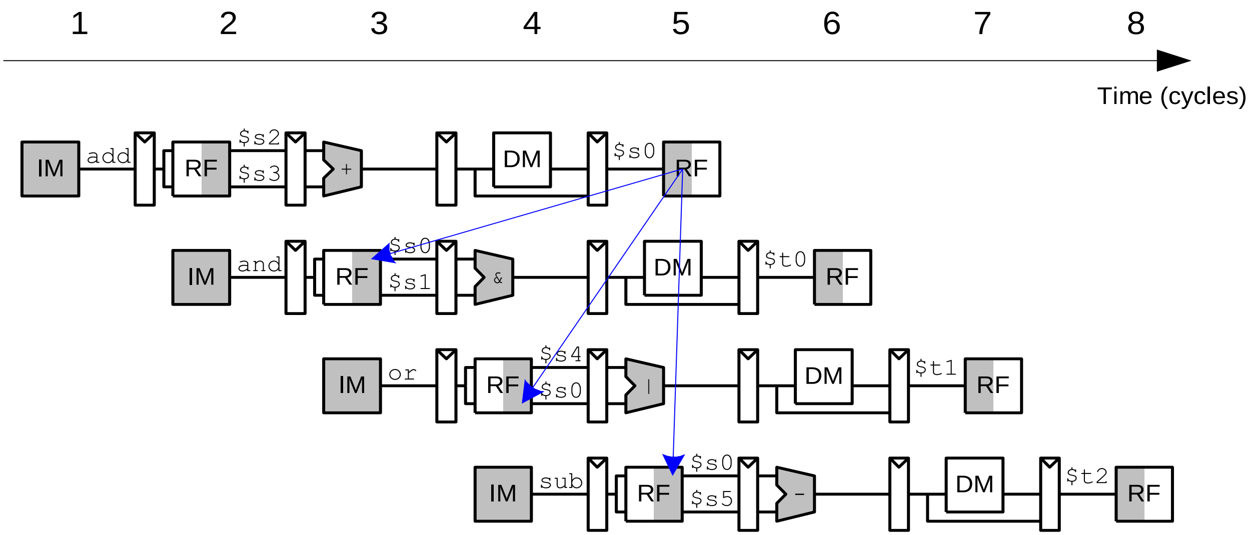
ELSEVIER

# Pipeline Hazards

- When an instruction depends on result from instruction that hasn't completed
- Types:
  - **Structural hazard:** We have already solve it by using Harvard architecture. (depart the instruction memory and data memory into two pieces)
  - **Data hazard:** register value not yet written back to register file
  - **Control hazard:** next instruction not decided yet (caused by branches)

# Data Hazard

```
add $s0, $s2, $s3
and $t0, $s0, $s1
or  $t1, $s4, $s0
sub $t2, $s0, $s5
```

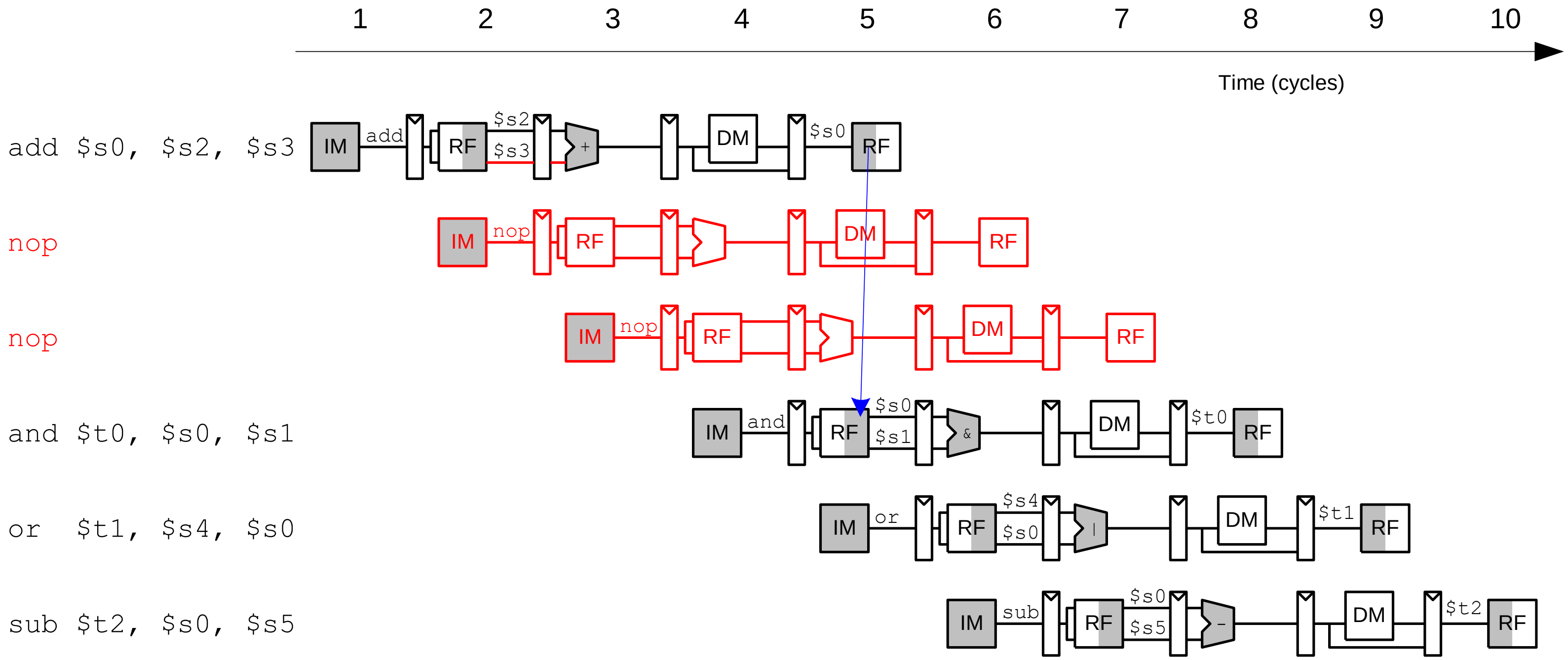


# Handling Data Hazards

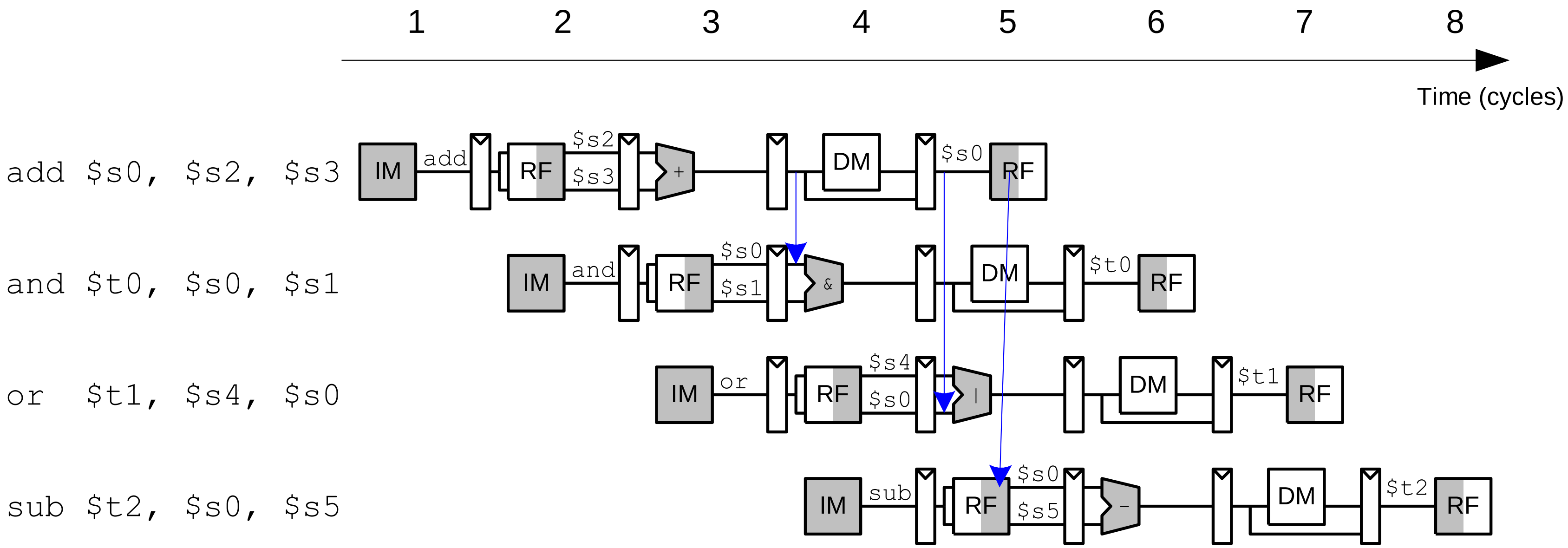
- Insert `nop`s in code at compile time
- Rearrange code at compile time
- Forward data at run time
- Stall the processor at run time

# Compile-Time Hazard Elimination

- Insert enough `nop`s for result to be ready
- Or move independent useful instructions forward

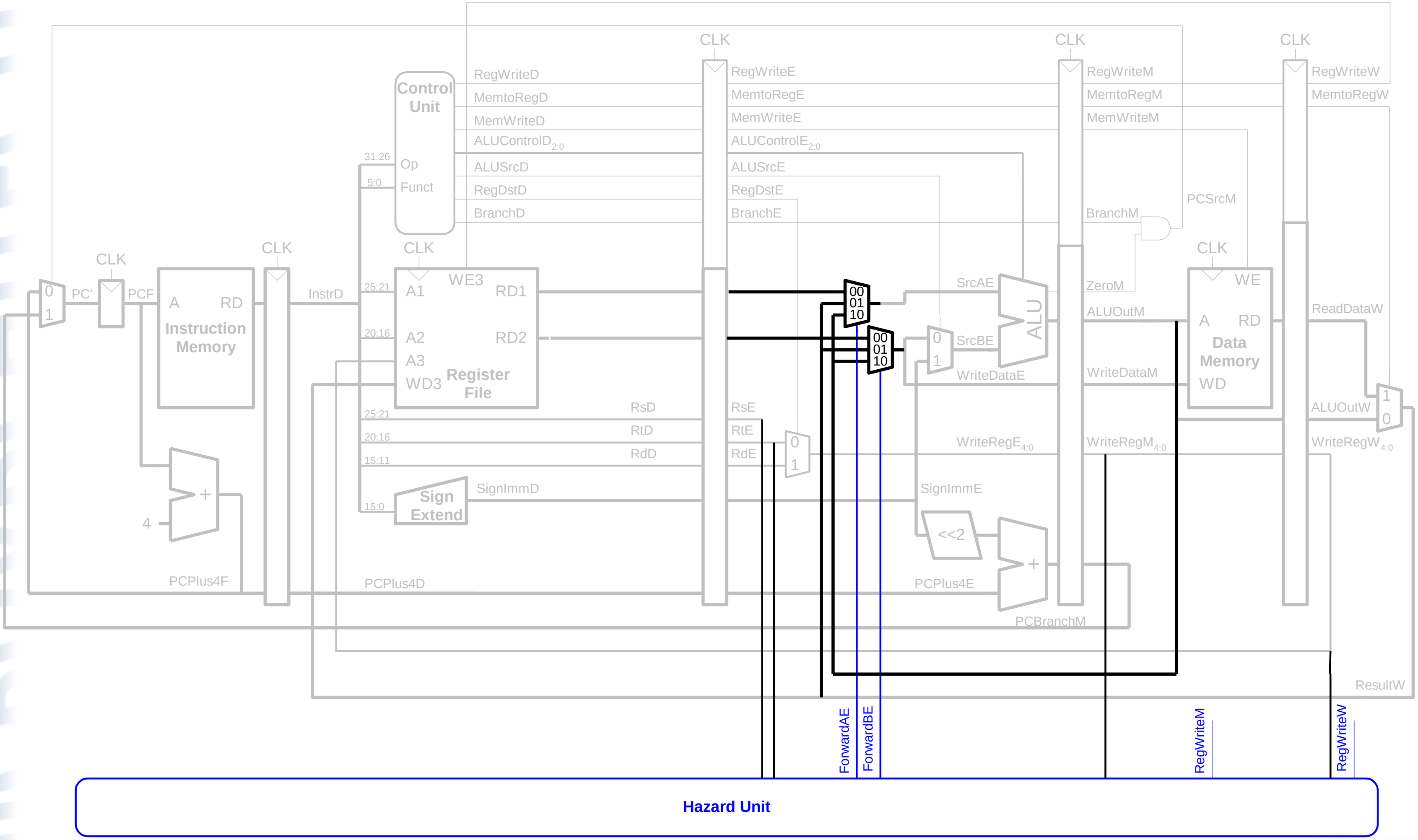


# Data Forwarding



```
add $s0, $s2, $s3
and $t0, $s0, $s1
or  $t1, $s4, $s0
sub $t2, $s0, $s5
```

# Data Forwarding



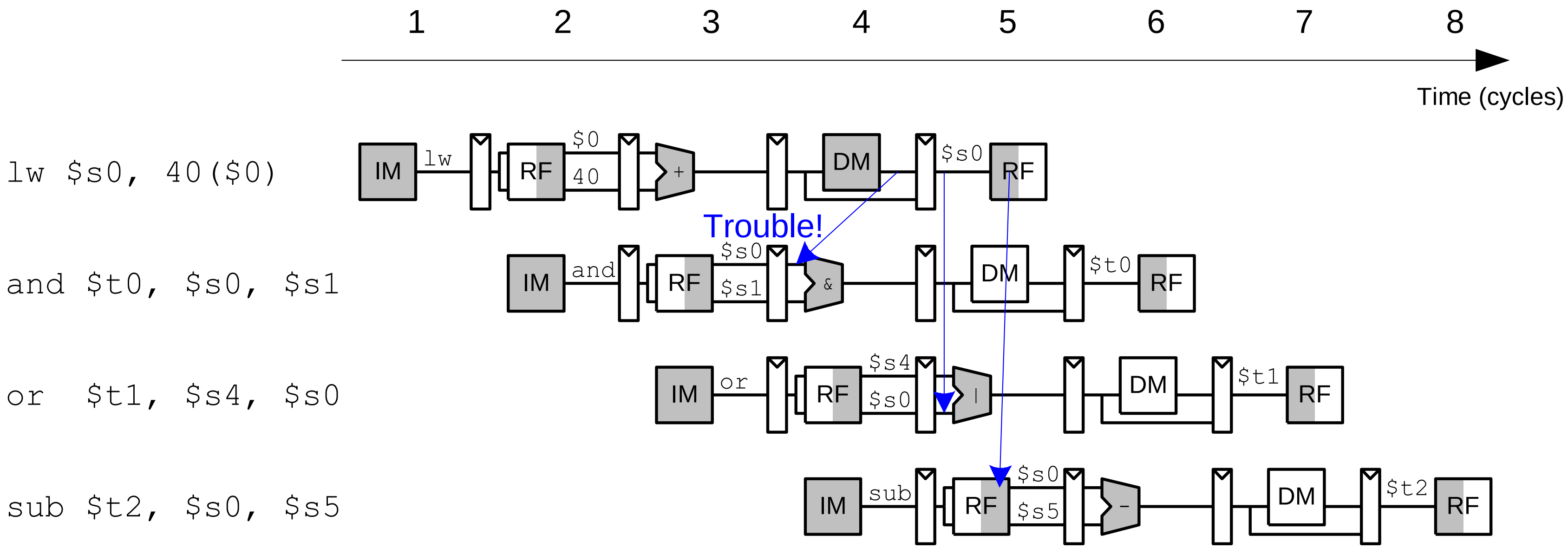
# Data Forwarding

- Forward to Execute stage from either:
  - Memory stage or
  - Writeback stage
- Forwarding logic for *ForwardAE*:

```
if       ((rsE != 0) AND (rsE == WriteRegM) AND RegWriteM)
    then    ForwardAE = 10
else if ((rsE != 0) AND (rsE == WriteRegW) AND RegWriteW)
    then    ForwardAE = 01
else        ForwardAE = 00
```

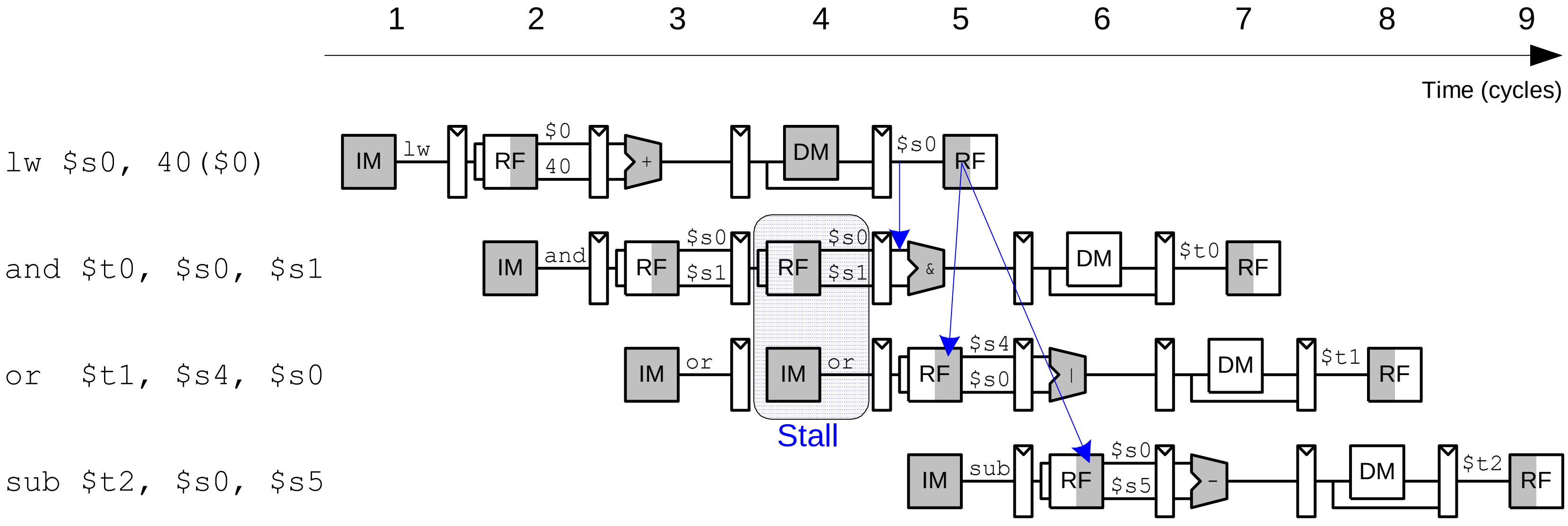
**Forwarding logic for *ForwardBE* same, but replace *rsE* with *rtE***

# Stalling

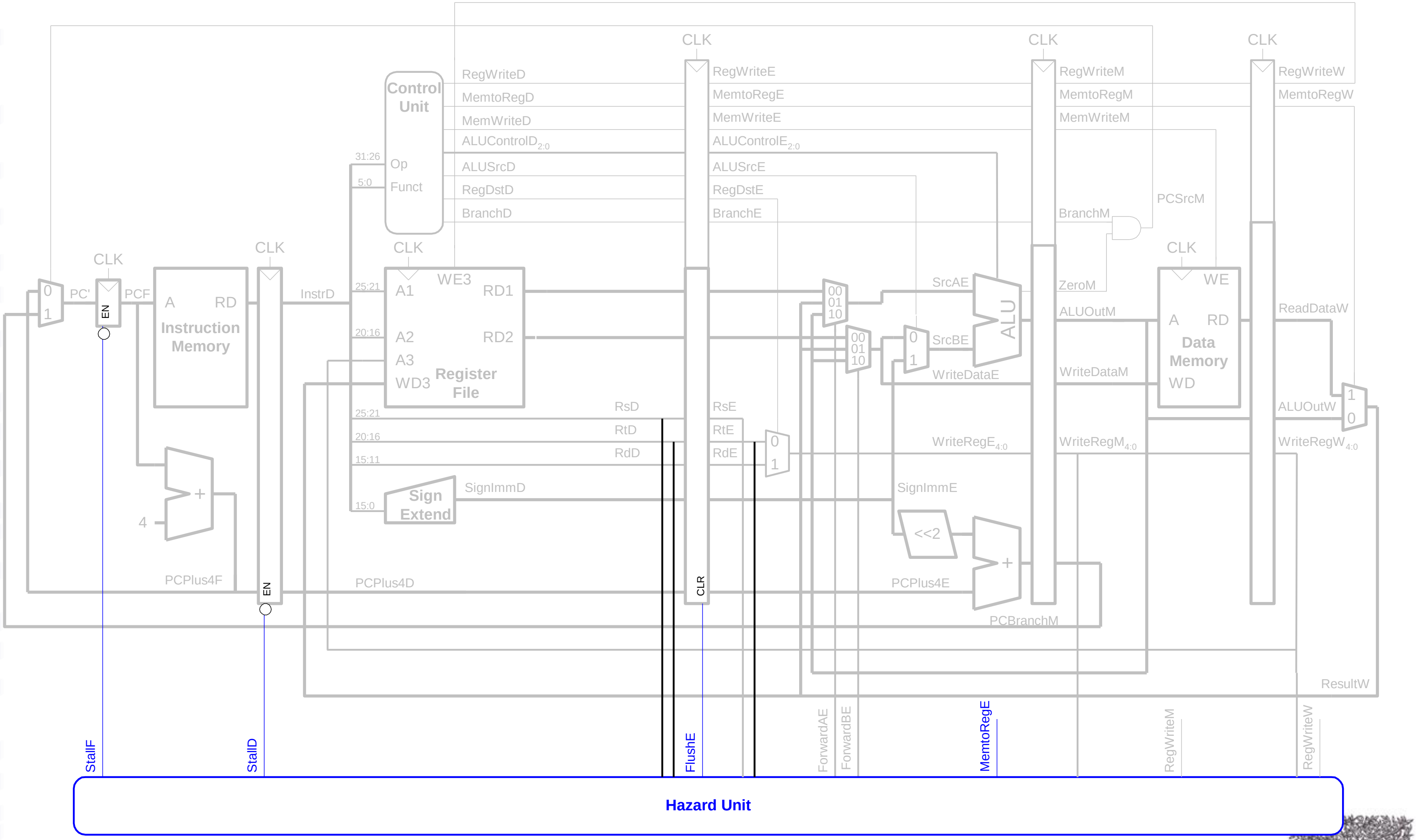


1
2
3
4
5
6
7
8
Time (cycles)
lw $s0, 40($0)
IM
lw
RF
$0
40
+
DM
$s0
RF
Trouble!
and $t0, $s0, $s1
IM
and
RF
$s0
$s1
&
DM
$t0
RF
or $t1, $s4, $s0
IM
or
RF
$s4
$s0
|
DM
$t1
RF
sub $t2, $s0, $s5
IM
sub
RF
$s0
$s5
-
DM
$t2
RF

# Stalling


1
2
3
4
5
6
7
8
9
Time (cycles)
lw $s0, 40($0)
and $t0, $s0, $s1
or $t1, $s4, $s0
sub $t2, $s0, $s5
IM
lw
RF
$0
40
+
DM
$s0
RF
IM
and
RF
$s0
$s1
RF
$s0
$s1
&
DM
$t0
RF
IM
or
IM
or
RF
$s4
$s0
|
DM
$t1
RF
Stall
IM
sub
RF
$s0
$s5
-
DM
$t2
RF

# Stalling Hardware



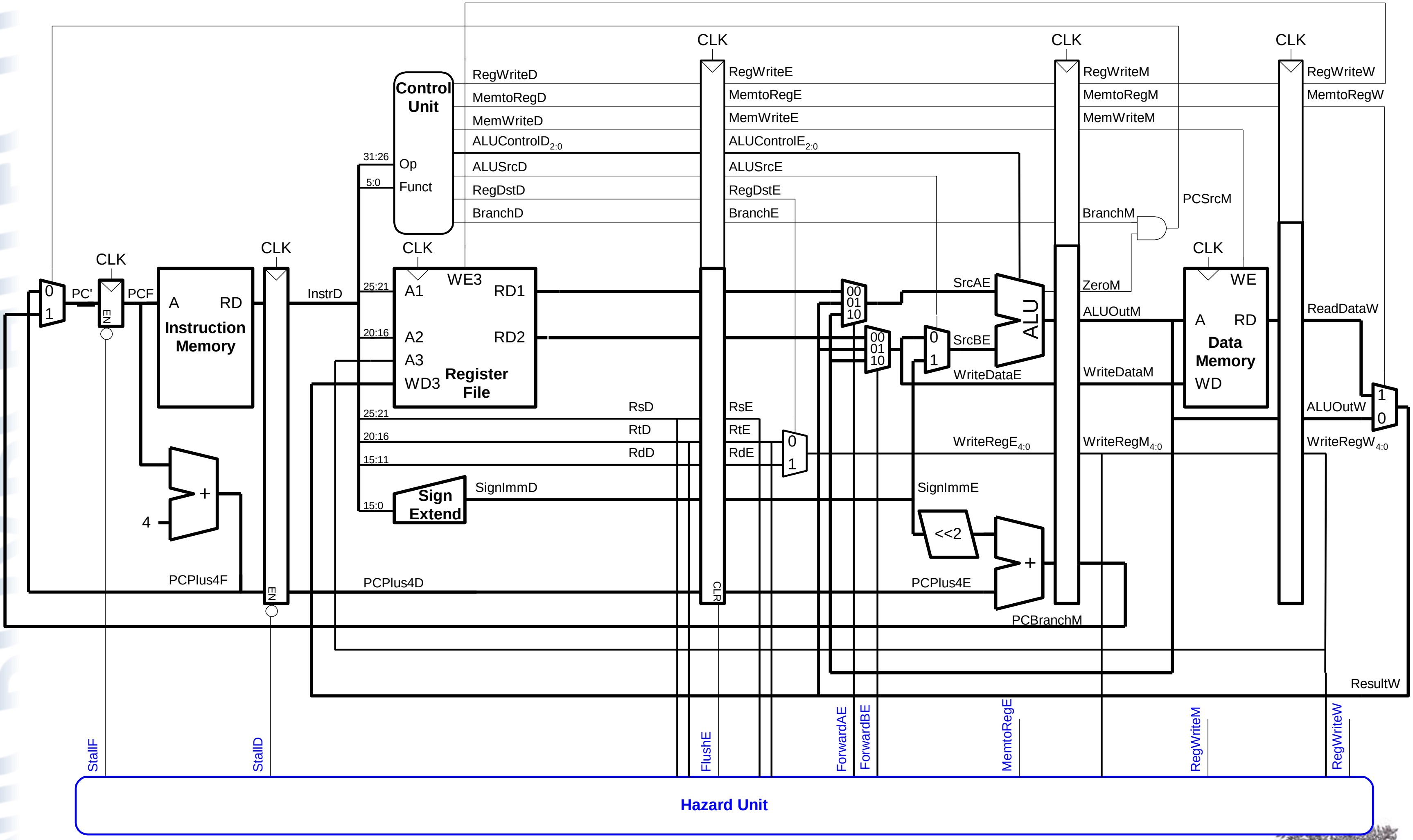
# Stalling Logic

```
lwstall =
   ((rsD==rtE) OR (rtD==rtE)) AND MemtoRegE

StallF = StallD = FlushE = lwstall
```

# Control Hazards

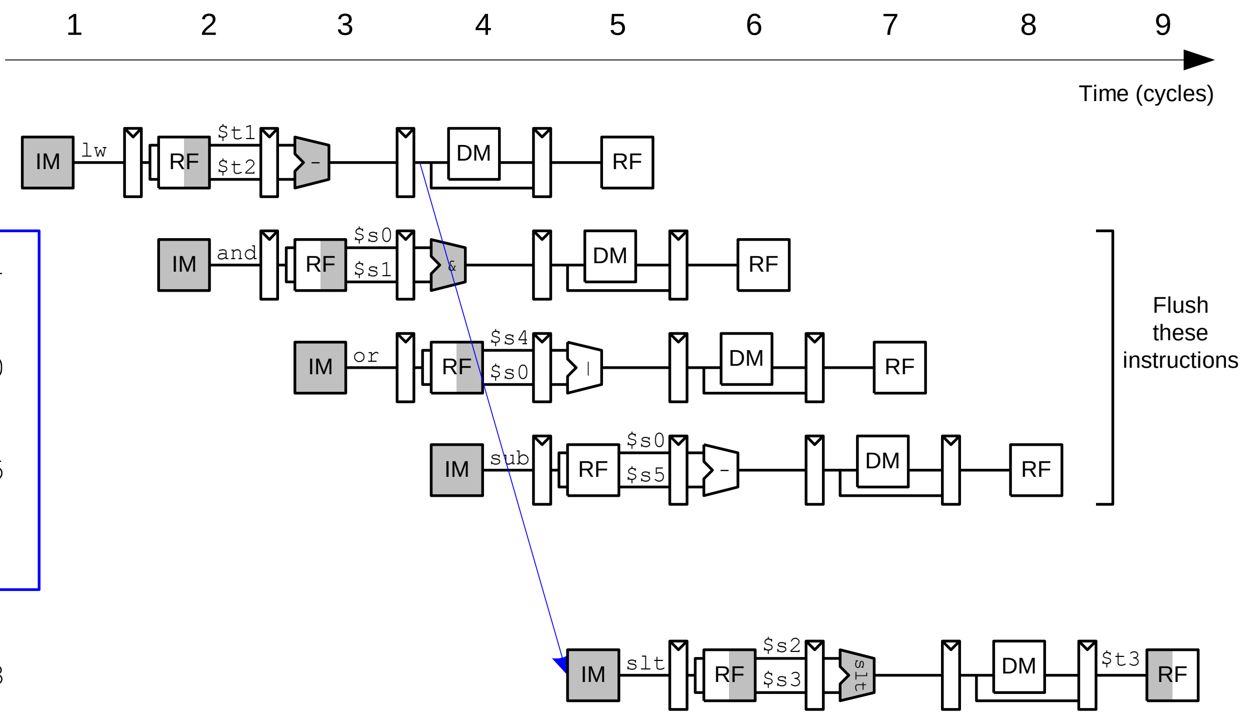
- **`beq`:**
  - branch not determined until 4th stage of pipeline
  - Instructions after branch fetched before branch occurs
  - These instructions must be flushed if branch happens
- **Branch misprediction penalty**
  - number of instruction flushed when branch is taken
  - May be reduced by determining branch earlier

# Control Hazards: Original Pipeline

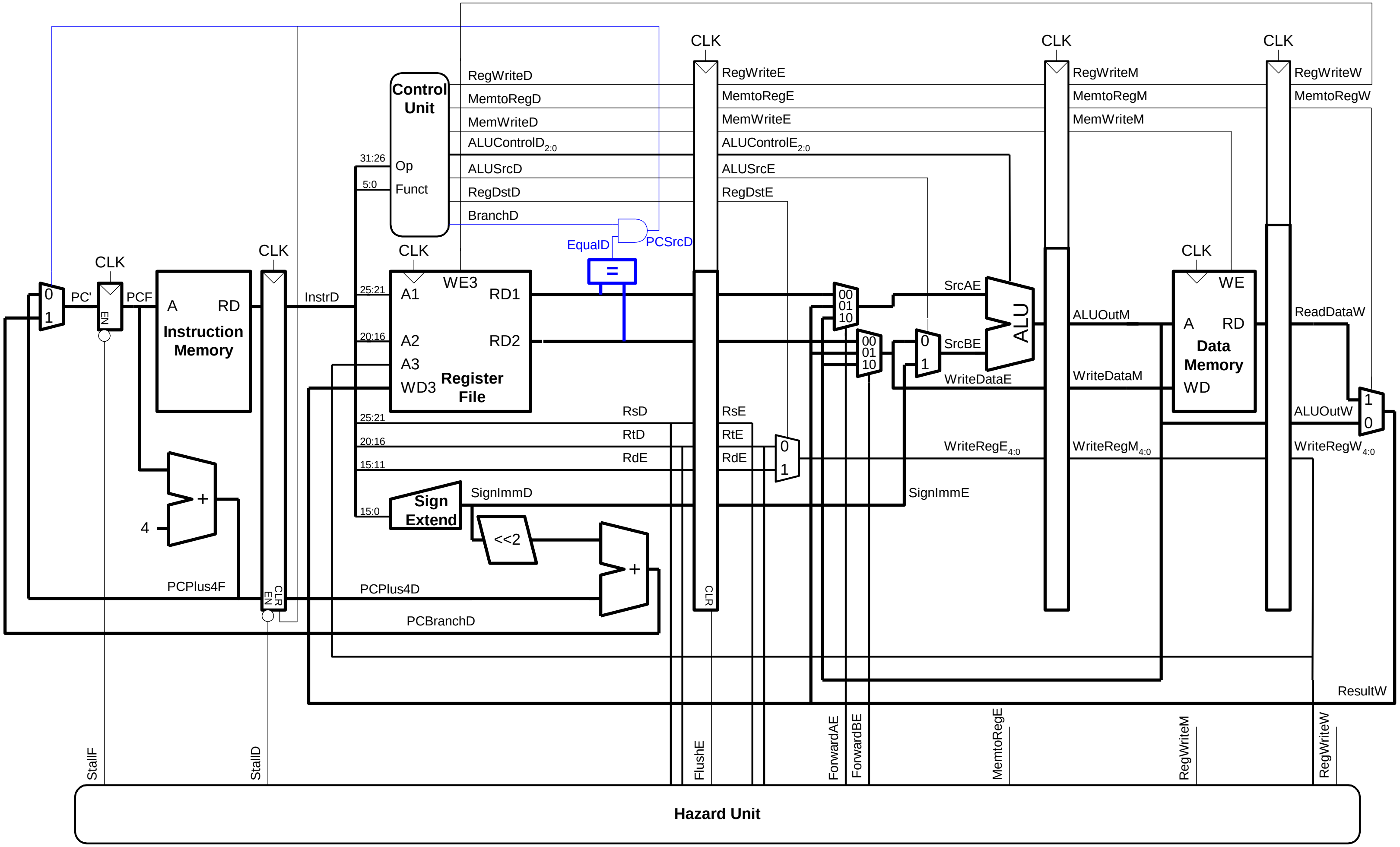


# Control Hazards

```
20   beq $t1, $t2, 40

24   and $t0, $s0, $s1

28   or  $t1, $s4, $s0

2C   sub $t2, $s0, $s5

30   ...
...
64   slt $t3, $s2, $s3
```

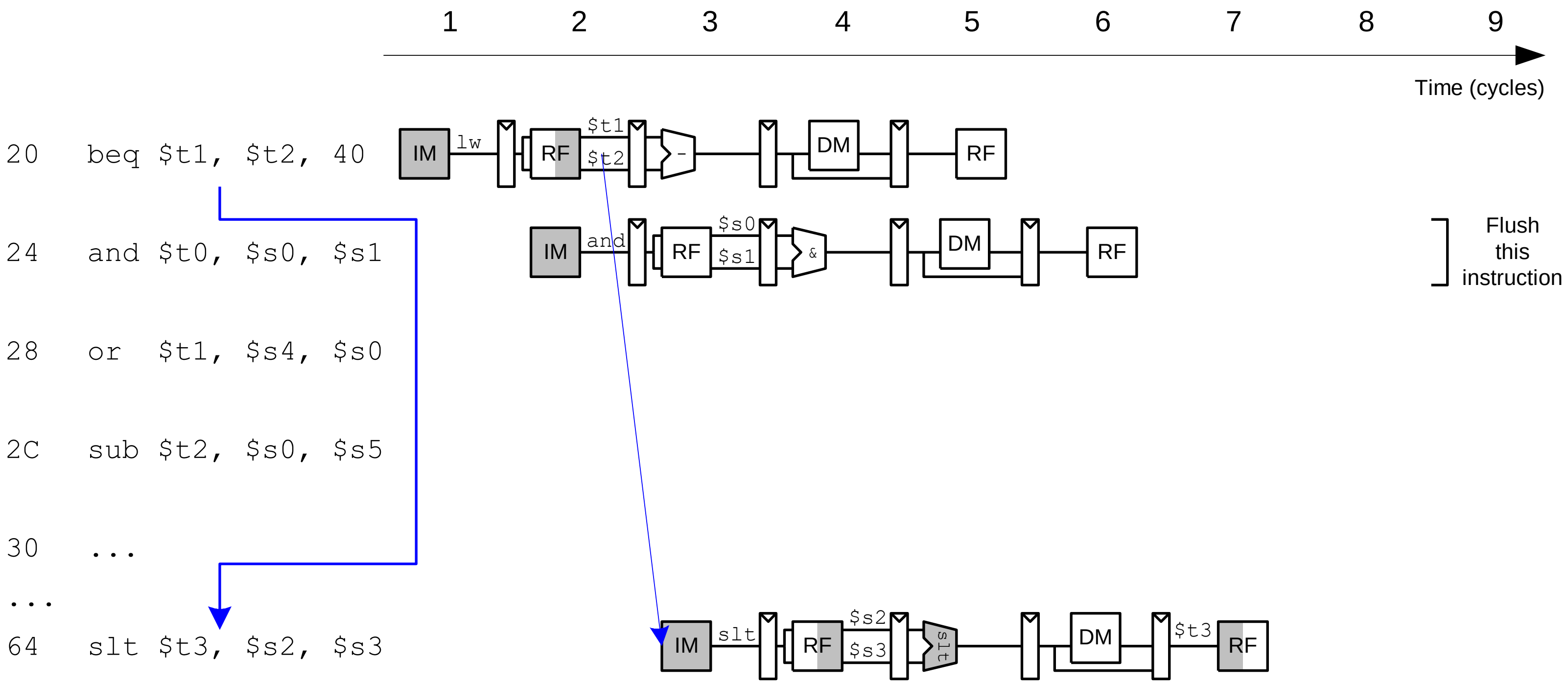


# Early Branch Resolution



**Introduced another data hazard in Decode stage**

# Early Branch Resolution

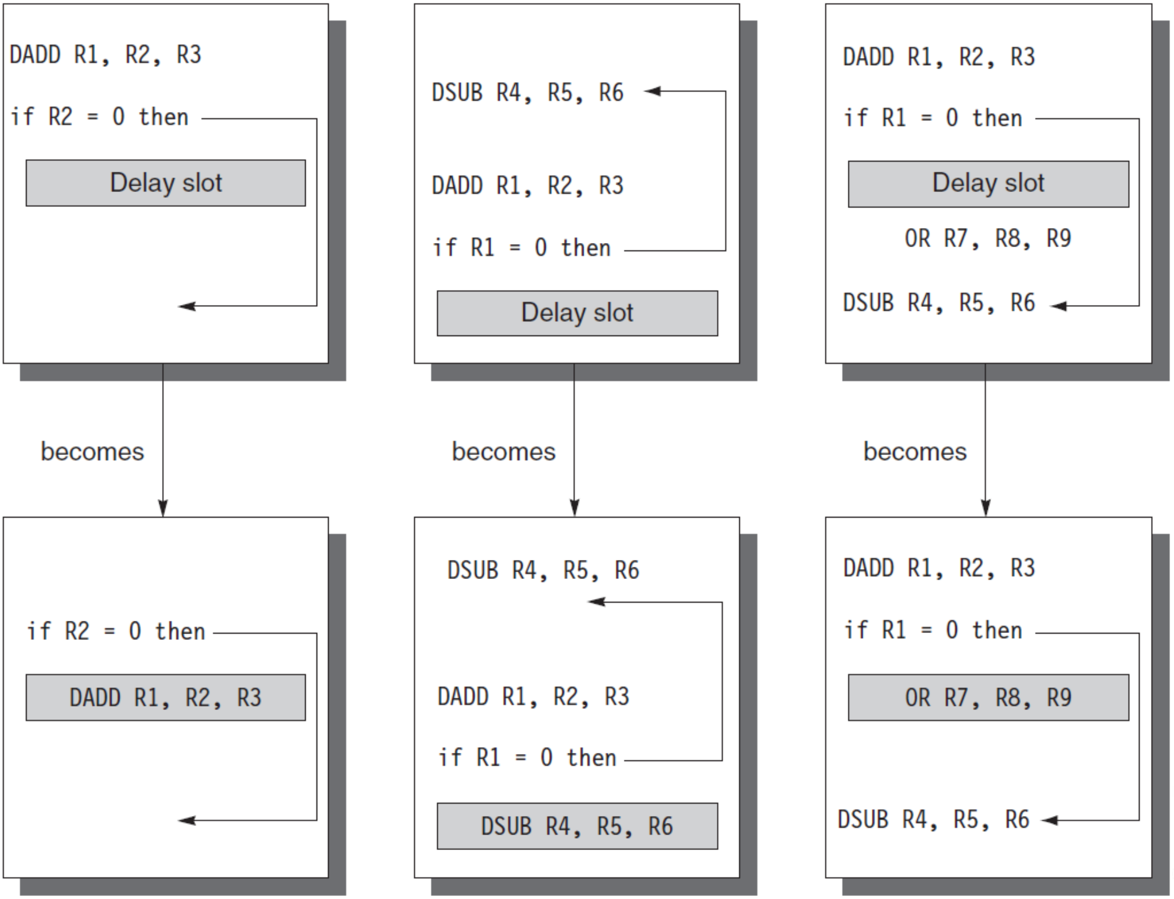


```
20   beq $t1, $t2, 40
24   and $t0, $s0, $s1
28   or  $t1, $s4, $s0
2C   sub $t2, $s0, $s5
30   ...
...
64   slt $t3, $s2, $s3
```

# Reducing Pipeline Branch Penalties

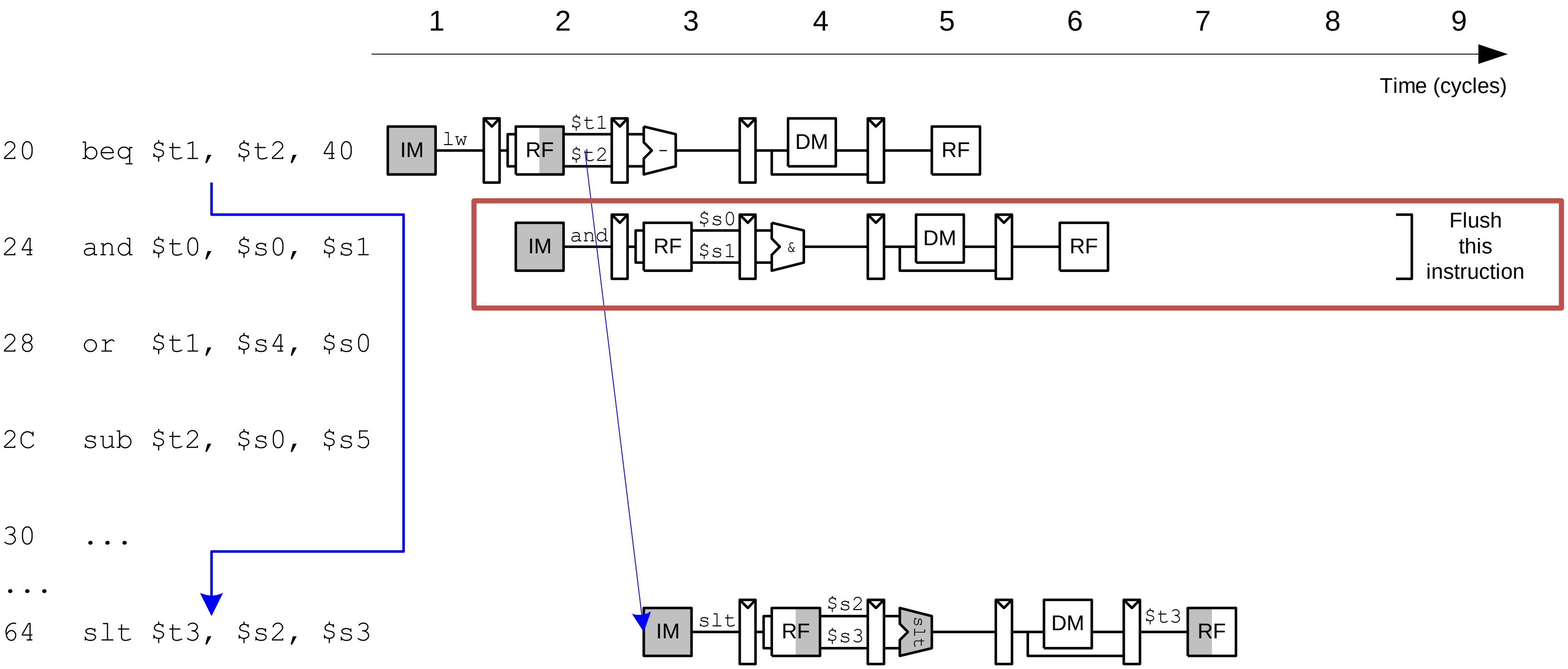
- Use a delayed branch
  - **Compiler** schedules one instruction (or more than one depending on the need) right after the branch. The instruction(s) is always executed, after which we will know the ID outcome of the Branch instruction.

(a) is the best.
(b) and (c) are used only when (a) is impossible.
(b) is preferred if Branch has higher probability to be taken.
(c) is preferred if otherwise.
Both (b) and (c) need to undone their changes (registers for example) if their assumptions turn out to be false!



(a) From before (b) From target (c) From fall-through

# Branch delay slot

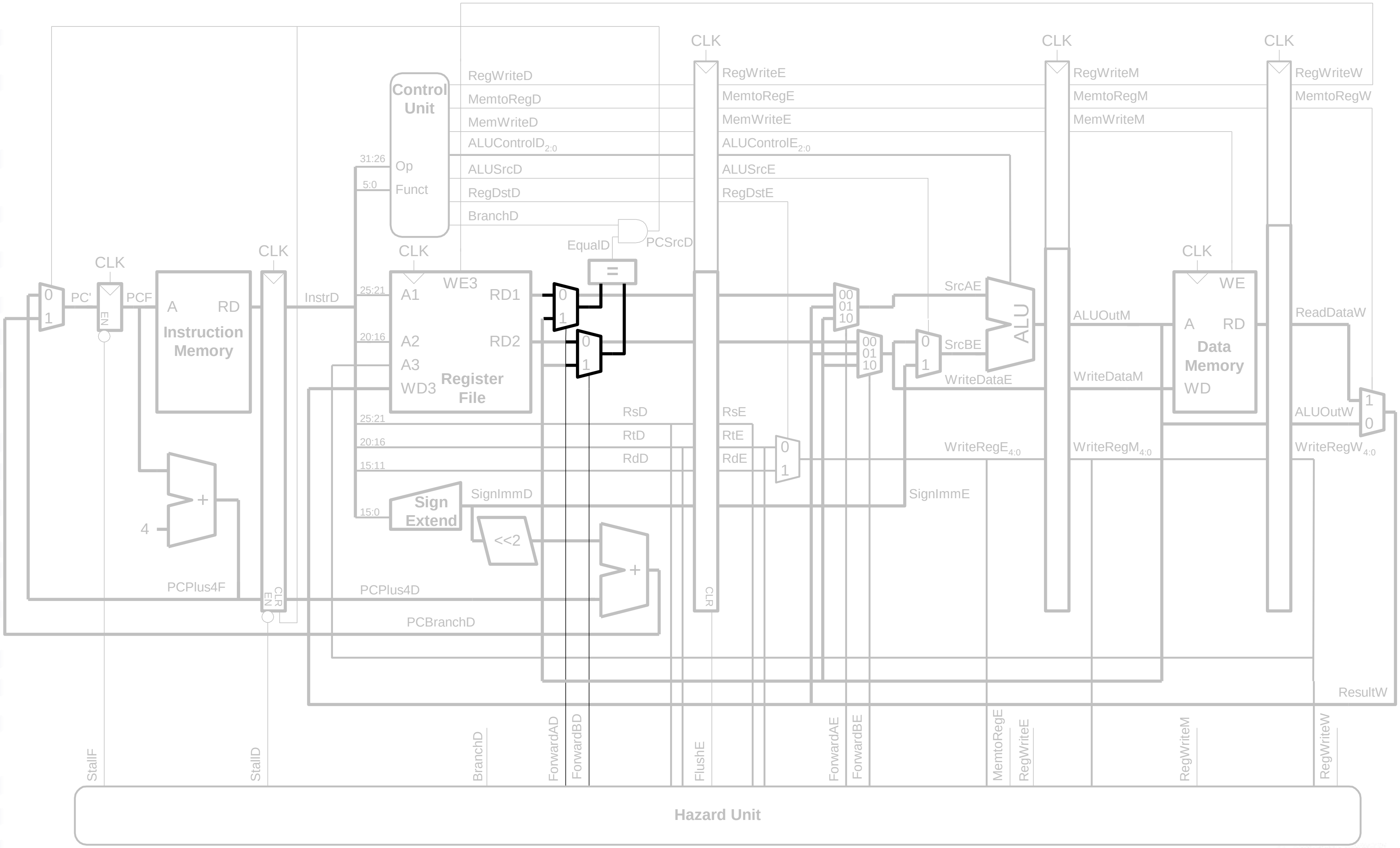


```
20   beq $t1, $t2, 40
24   and $t0, $s0, $s1
28   or  $t1, $s4, $s0
2C   sub $t2, $s0, $s5
30   ...
...
64   slt $t3, $s2, $s3
```

Branch delay slot： The instruction after branch always execute.

In this way, the only left instruction could use effectively, instead of wasting this stage

# Handling Data & Control Hazards



© *Digital Design and Computer Architecture*, 2nd Edition, 2012

# Control Forwarding & Stalling Logic

- **Forwarding logic:**

```
ForwardAD = (rsD !=0) AND (rsD == WriteRegM) AND RegWriteM
ForwardBD = (rtD !=0) AND (rtD == WriteRegM) AND RegWriteM
```

- **Stalling logic:**

```
branchstall = BranchD AND RegWriteE AND
                 (WriteRegE == rsD OR WriteRegE == rtD)
              OR
              BranchD AND MemtoRegM AND
                 (WriteRegM == rsD OR WriteRegM == rtD)

StallF = StallD = FlushE = lwstall OR branchstall
```

# Branch Prediction

- Guess whether branch will be taken
  - Backward branches are usually taken (loops)
  - Consider history to improve guess
- Good prediction reduces fraction of branches requiring a flush

# Pipelined Performance Example

- SPECINT2000 benchmark:
  - 25% loads
  - 10% stores
  - 11% branches
  - 2% jumps
  - 52% R-type
- Suppose:
  - 40% of loads used by next instruction
  - 25% of branches mispredicted
  - All jumps flush next instruction
- **What is the average CPI?**

# Pipelined Performance Example

- SPECINT2000 benchmark:
  - 25% loads
  - 10% stores
  - 11% branches
  - 2% jumps
  - 52% R-type
- Suppose:
  - 40% of loads used by next instruction
  - 25% of branches mispredicted
  - All jumps flush next instruction
- **What is the average CPI?**
  - Load/Branch CPI = 1 when no stalling, 2 when stalling
  - $\text{CPI}_{\text{lw}} = 1(0.6) + 2(0.4) = 1.4$
  - $\text{CPI}_{\text{beq}} = 1(0.75) + 2(0.25) = 1.25$

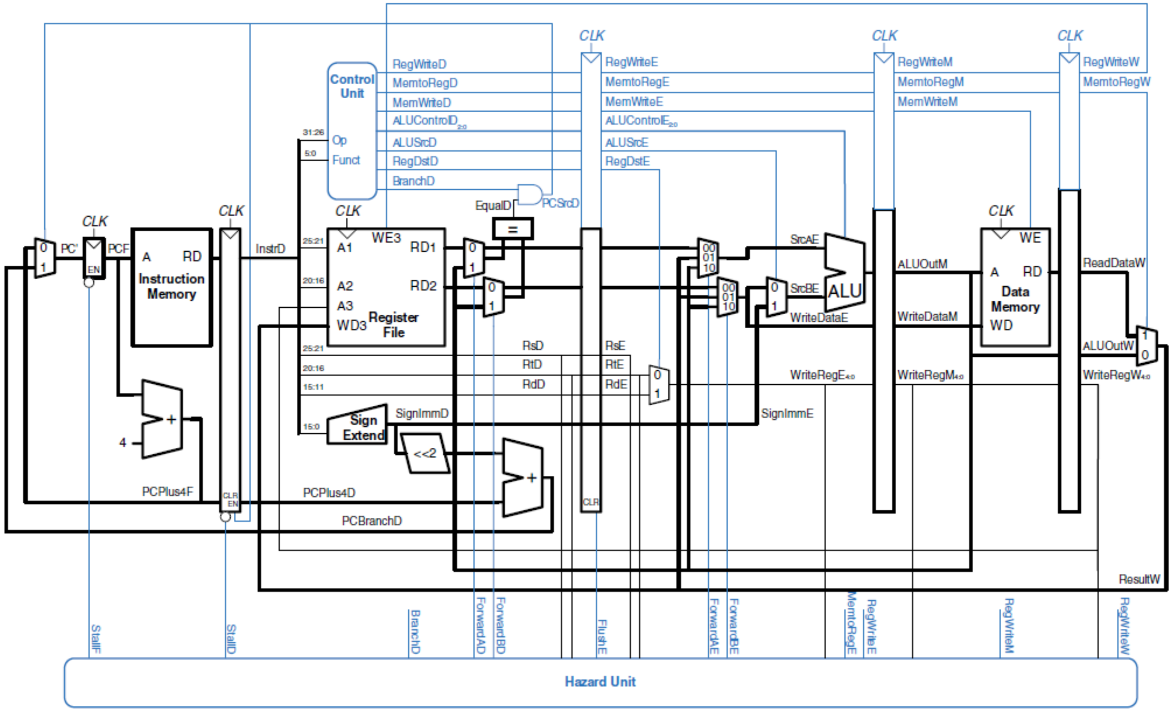
**Average CPI =** $(0.25)(1.4) + (0.1)(1) + (0.11)(1.25) + (0.02)(2) + (0.52)(1)$

**= 1.15**

# Pipelined Performance

- Pipelined processor critical path:

$$
\begin{aligned}
T_c = \max\{ \\
&t_{pcq} + t_{\text{mem}} + t_{\text{setup}}, \\
&2(t_{RFread} + t_{\text{mux}} + t_{\text{eq}} + t_{\text{AND}} + t_{\text{mux}} + t_{\text{setup}}), \\
&t_{pcq} + t_{\text{mux}} + t_{\text{mux}} + t_{\text{ALU}} + t_{\text{setup}}, \\
&t_{pcq} + t_{\text{memwrite}} + t_{\text{setup}}, \\
&2(t_{pcq} + t_{\text{mux}} + t_{\text{RFwrite}}) \}
\end{aligned}
$$



# Pipelined Performance Example

| Element | Parameter | Delay (ps) |
|---|---|---|
| Register clock-to-Q | $t_{pcq\_\mathrm{PC}}$ | 30 |
| Register setup | $t_{\mathrm{setup}}$ | 20 |
| Multiplexer | $t_{\mathrm{mux}}$ | 25 |
| ALU | $t_{\mathrm{ALU}}$ | 200 |
| Memory read | $t_{\mathrm{mem}}$ | 250 |
| Register file read | $t_{RF\mathrm{read}}$ | 150 |
| Register file setup | $t_{RF\mathrm{setup}}$ | 20 |
| Equality comparator | $t_{eq}$ | 40 |
| AND gate | $t_{\mathrm{AND}}$ | 15 |
| Memory write | $T_{\mathrm{memwrite}}$ | 220 |
| Register file write | $t_{RF\mathrm{write}}$ | 100 ps |

$$T_c = 2(t_{RFread} + t_{\mathrm{mux}} + t_{\mathrm{eq}} + t_{\mathrm{AND}} + t_{\mathrm{mux}} + t_{\mathrm{setup}})$$

$$= 2[150 + 25 + 40 + 15 + 25 + 20] \text{ ps} = \mathbf{550\ ps}$$

ELSEVIER

# Pipelined Performance Example

Program with 100 billion instructions

**Execution Time** = (# instructions) × CPI × $T_c$

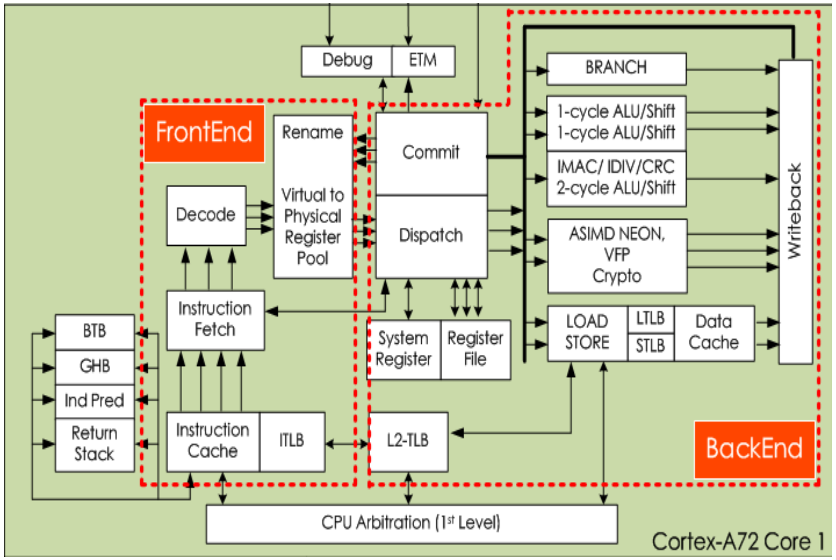
= $(100 \times 10^{9})(1.15)(550 \times 10^{-12})$

**= 63 seconds**

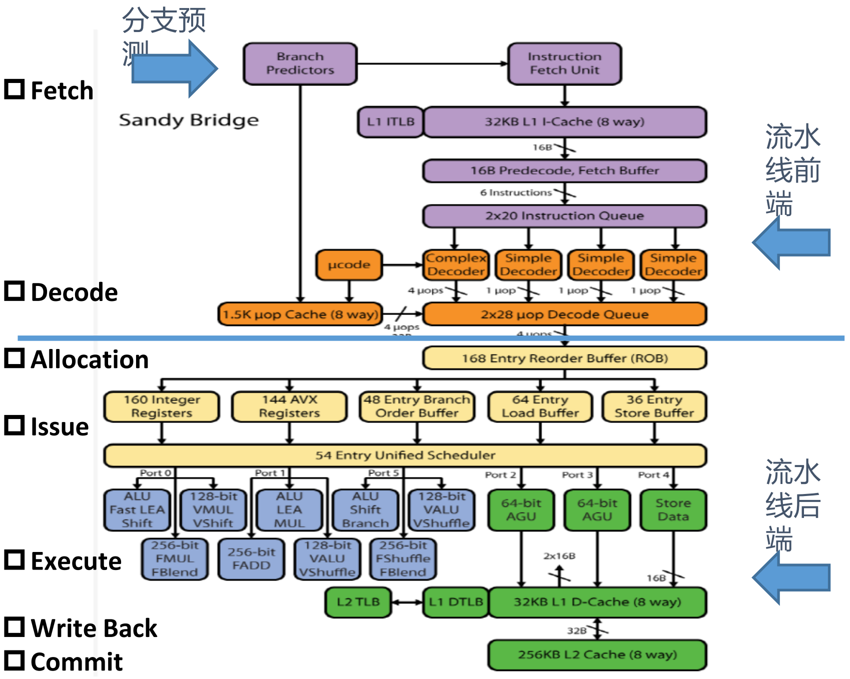
# Processor Performance Comparison

| Processor | Execution Time (seconds) | Speedup (single-cycle as baseline) |
|---|---|---|
| **Single-cycle** | 92.5 | 1 |
| **Multicycle** | 133 | 0.70 |
| **Pipelined** | 63 | 1.47 |

# CPU流水线：ARM vs X86



ARM64

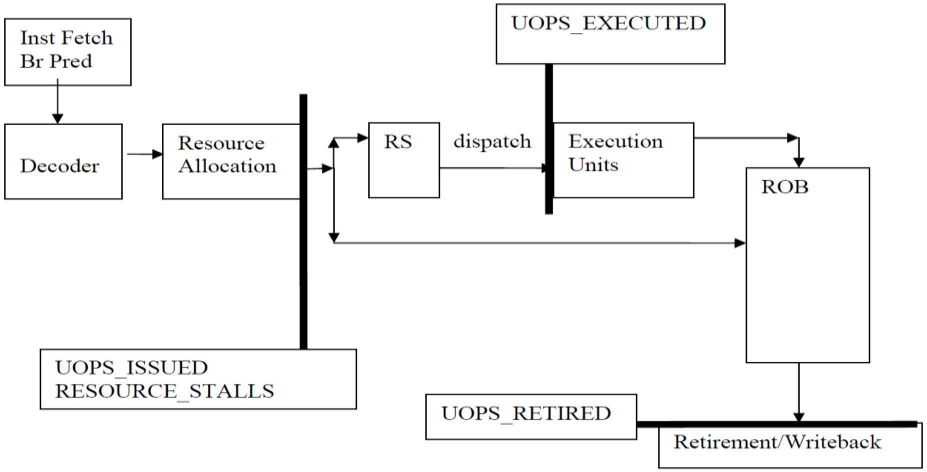


X86 Xeon

ARM流水线与Xeon基本一致，按流水线前后端分解

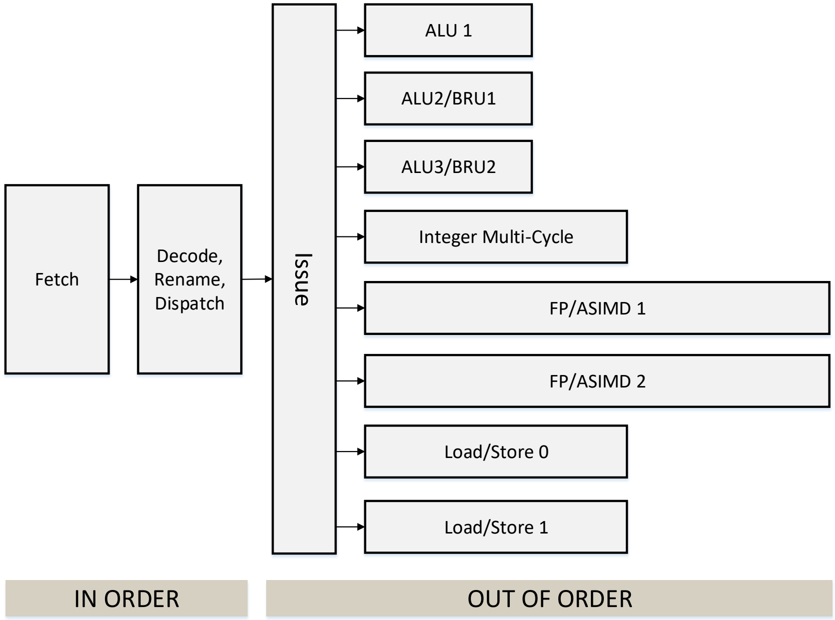
# 鲲鹏流水线技术亮点：CPU流水线主要阶段

- **Fetch**：提取指令并计算下一次Fetch的地址。包括指令缓存、Branch Prediction、Branch Target Buffer、Return Address Stack。
- **Decode**
  - 分解指令流到独立指令；
  - Translate X86指令到RISC-like Uops;
  - 理解指令语义，包括指令类型（Control、Memory、Arithmetic，等等），运算操作类型、需要什么资源(读和写需要的寄存器，等等）。
- **Allocation:**RegisterRenaming+Resources Reservation.
- **Issue**：分发指令到相应执行单元，从这儿开始进入错序执行阶段。
- **Execute**：指令执行阶段
- **Write Back**：将执行结果写入Register File、Reorder Buffer、等等
- **Commit**：重整执行结果次序、决定Speculative Execution正确性，最终输出结果。



| | | | | | | |
|---|---|---|---|---|---|---|
| Fetch | Decode | Allocation | Issue | Execute | WriteBack | Commit |

# 鲲鹏流水线技术：弱保序



CPU弱保序，即乱序执行：

- 处理器不按程序规定的顺序执行指令，它根据内部功能部件的空闲状态，动态分发执行指令，但是指令结束的顺序还是按照原有程序规定的顺序。
- 处理器内部功能部件并行运转，避免了不必要的阻塞，有效提高了处理器执行指令的性能。
- 处理器分析影响执行结果的指令，避免出现有显式的数据依赖和控制依赖的乱序，但是，特殊情况下的读写乱序可能影响程序执行结果，需要软件甄别。

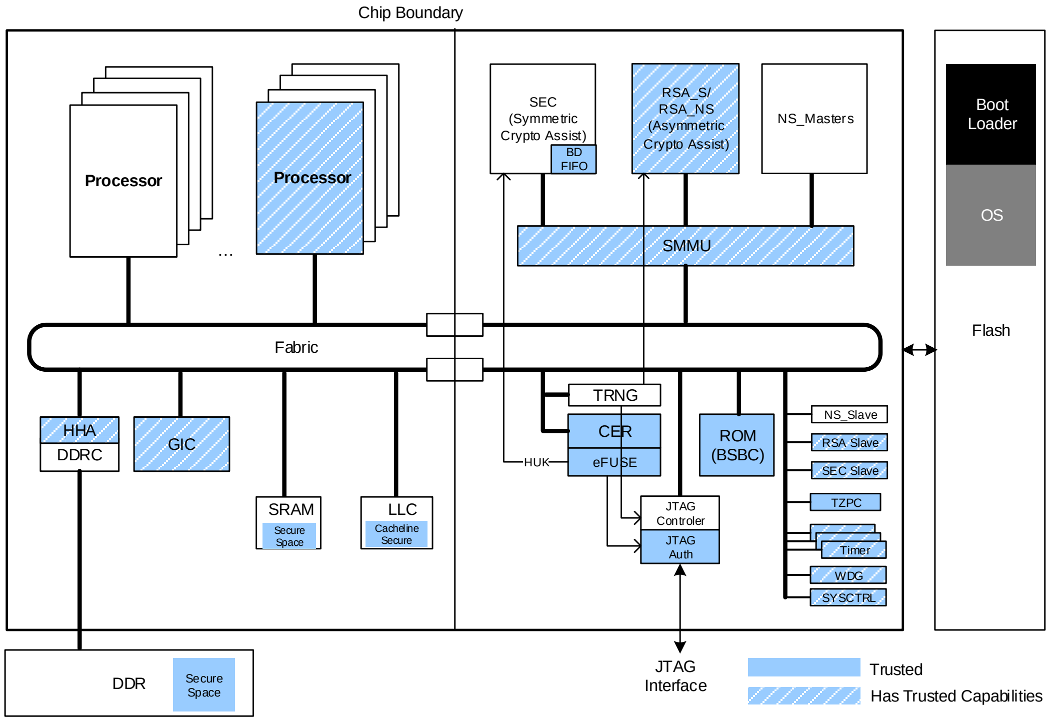
制约CPU效率因素

•CPU流水线前端限制：执行单元空闲，但前端不能输送充分多操作指令

•CPU流水线后端限制：执行单元繁忙或执行时等待数据，指令执行出现等待

•分支预测错误：执行错误分支浪费时间 + 处理错误分支执行消耗时间。

# 芯片架构– 系统安全&IMU



系统安全：支持安全启动，以及保证系统在可信环境内运行的一套软硬件方案。该方案由Secure Boot技术和ARM架构中的Trust Zone技术结合而成。

IMU（Intelligent Management Unit）是Hi162x芯片内部的智能管理单元，完善ARM节点在数据中心的管理和控制，未来数据中心设备管理要求统一、智能和协同，遵循管理系统集中决策+节点执行监控，按照设备节点模型统一管理。

IMU作为数据中心的管理末端，协同BMC，完成数据中心的节点执行监控。

IMU可以覆盖的功能：RAS故障预处理以及错误记录上报、安全信任根、能效管理、芯片内部管理。

# 鲲鹏920系列芯片——加速器引擎功能 (1)

| 组件 | 算法 | 规格描述 |
|---|---|---|
| OpenSSL引擎库 | SM4 | 支持SM4-CBC/SM4-CTR/SM4-XTS，符合GM/T0002-2012规范。<br>单处理器(Kunpeng 920)最大带宽30Gbps。支持同步、异步模式。 |
| | SM3 | 支持SM3，符合GM/T 0004-2012规范。<br>单处理器(Kunpeng 920)最大带宽60Gbps。支持同步、异步模式。 |
| | RSA | 支持RSA1024/RSA2048/RSA3072/RSA4096，符合NIST FIPS-197标准规范。<br>单处理器(Kunpeng 920)RSA2048 sign最大带宽54K ops。支持同步、异步模式。 |
| | DH | 支持DH 768/1024/1536/2048/3072/4096，符合NIST FIPS-197标准规范<br>单处理器(Kunpeng 920)DH768/1024/1536/2048/3072/4096 最大带宽为94.4/56/25.6/14/4.8/2.24kops。<br>支持同步、异步模式。 |
| | AES | 支持AES-ECB/AES-CBC/AES-CTR/AES-XTS，符合NIST FIPS-197标准规范。<br>单处理器(Kunpeng 920)最大带宽60Gbps。支持同步、异步模式。 |

# 鲲鹏920系列芯片——加速器引擎功能 (2)

| 组件 | 算法 | 规格描述 |
|---|---|---|
| Zlib库 | ZLIB | 支持ZLIB数据格式，符合RFC1950标准规范。<br>单处理器(Kunpeng 920)最大压缩带宽7GB/s，静态Huffman解压最大带宽8GB/s。压缩率50%。支持同步模式。 |
|  | GZIP | 支持GZIP数据格式，符合RFC1952标准规范。<br>单处理器(Kunpeng 920)最大压缩带宽7GB/s，静态Huffman解压最大带宽8GB/s。压缩率50%。支持同步模式。 |
| 内核Crypto | SM4 | 支持SM4-XTS，符合GM/T 0002-2012规范。<br>单处理器(Kunpeng 920)最大带宽30Gbps。支持异步模式。 |

# 鲲鹏系列处理器 vs 业界主流产品的性能和效能对比

鲲鹏920 64核型号

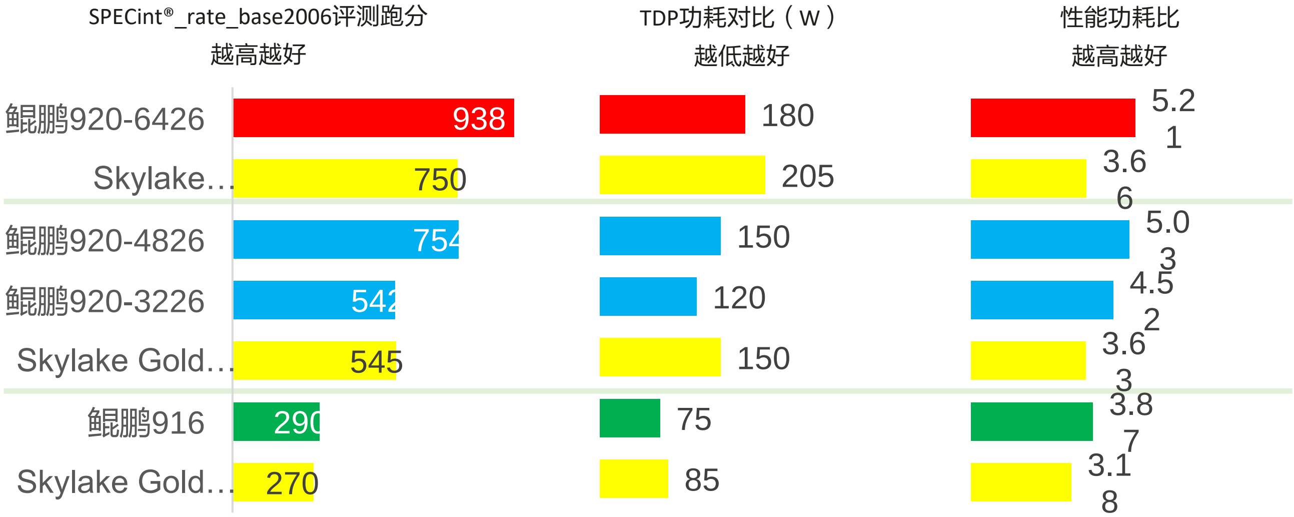
极致性能 25%+

鲲鹏920 32/48核型号

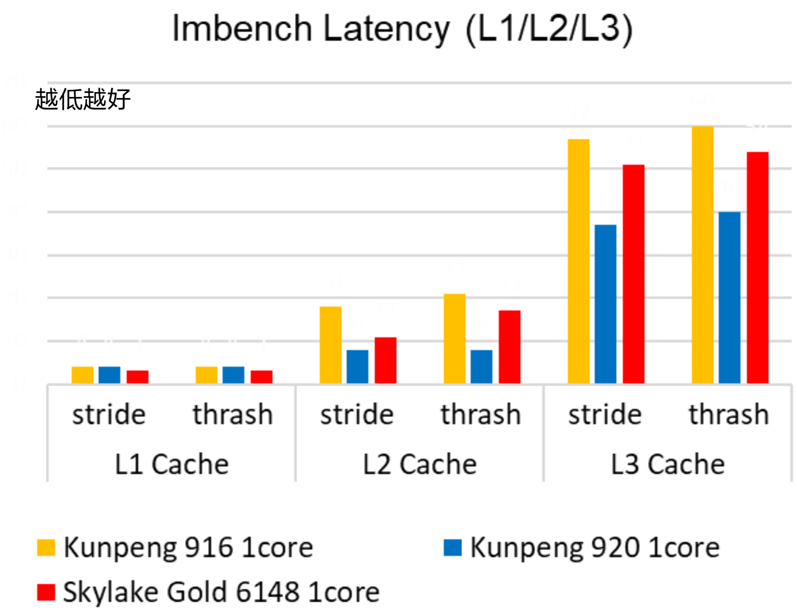
极致效能 30%+

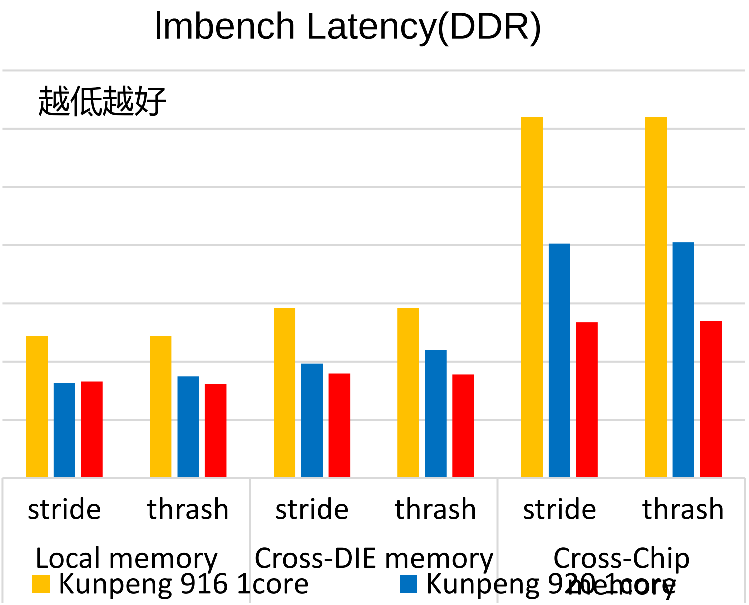
鲲鹏916 32核型号

低功耗 11%



# 鲲鹏整体性能对比-内存带宽和时延性能





| 平台 | TaiShan V2 with 2-socket Kunpeng 920-4826/DDR4-2666) | X86 server with 2-socket Skylake 6148 (20 cores, 2.0 GHz/DDR4-2666) |
|---|---|---|
| STREAM<br>测试结果 | 287 GB/S with 84.28% efficiency | 192 GB/S with 75.08% efficiency |

# 鲲鹏整体性能对比- IO性能

| 特性 | 模式 | 规格类别 | 目标 | 实测数据（本地关SMMU） | 实测数据（本地开SMMU） |
|---|---|---|---|---|---|
| PCIe | IB：Mlx CX5 | PPS | read>95Mpps@4B<br>write:>64Mpps@4B<br>send:>63Mpps@4B | read:24并发时达到86.4Mpps@4B<br>write:24并发时达到86.4Mpps@4B<br>send：24并发时达到90Mpps@4B | read:24并发时达到86.4Mpps@4B<br>write:24并发时达到86.4Mpps@4B<br>send：24并发时达到90Mpps@4B |
| | | 时延 | Intel对接CX4数据：<br>read：1.95us@2B<br>write：0.93us@2B<br>read：4.09us@4KB<br>write：3.22us@4KB | read：1.66us@2B<br>write：0.99us@2B<br>send：1.06@2B<br>read：2.21us@4KB<br>write：1.98us@4KB<br>send：2.05@4KB | read：1.65us@2B<br>write：1.01us@2B<br>send：1.06@2B<br>read：2.18us@4KB<br>write：1.95us@4KB<br>send：2.04@4KB |
| | | 带宽 | read：线速@2KB<br>write:线速@2KB | read：线速90%@2KB，线速@8KB；<br>write：线速87%@2KB，线速@8KB；<br>（与x86+CX5持平） | read：线速90%@2KB，线速@8KB；<br>write：线速86%@2KB，线速@8KB；<br>（与x86+CX5持平） |
| | 网络：Mlx CX5/1822 | PPS | >10Mpps@64B | CX5：24队列 25.3Mpps@64B;<br>1822：16队列 13.7Mpps@64B; | CX5：24队列 14.5Mpps@64B;<br>1822：16队列 12Mpps@64B; |
| | | 时延 | 13.5us@64B | CX5：1队列1Mpps 7.8us<br>1822：1队列1Mpps 9.4us | CX5：1队列0.7Mpps 11.5us<br>1822：1队列0.7Mpps 13us |
| | | 带宽 | 线速@1518KB | CX5：16队列 线速@1518KB<br>1822：16队列 98Gb@1518KB | CX5：16队列 90Gb@1518KB<br>1822：16队列 68Gb@1518KB |
| | NVME：ES3000 V3/V5 | 最大读带宽(MB/s)@256KB | 3100 | 3025.7 | 3012.4 |
| | | 持续随机读KIOPS@4KB | 760 | 775.2 | 753.6 |
| | | 4K读延时(us) | 88 | avg：44.72<br>99%：88 | avg：46.24<br>99%：89 |
| | | 最大写带宽(MB/s)@256KB | 1950 | 2005.4 | 1957.7 |
| | | 持续随机写KIOPS@4KB | 175 | 508.6 | 503.6 |
| | | 4K写延时(us) | 18 | avg：11.75<br>99%：10 | avg：12.57<br>99%：11 |

HUAWEI

# 鲲鹏整体性能对比- IO性能 (2)

详细测试数据 Microsoft Excel 工作表

| 特性 | 模式 | 类别 | 规格 | 实测数据（本地关SMMU） | 实测数据（本地开SMMU） |
|---|---|---|---|---|---|
| NIC | 100G | PPS | >10Mpps@64B | 24核：<br>25Mpps@64B | 24核：<br>14.7Mpps@64B |
| | | 时延 | 100GE：<13.5us@64B<br>10GE：<15us@64B | 100G：12us@64B<br>10G：12us@64B | 10G：12.1us@64B<br>100G：12.6us@64B |
| | | 带宽 | 100GE：>94Gbps@双向带宽<br>25GE：>23.5Gbps@双向带宽<br>10GE：9.35Gbps@双向带宽 | 100G：94.1Gbps@1518B<br>25G：23.5@1518B<br>10GE：9.41@1518B | 100G：94.1Gbps@1518B<br>25G：23.5@1518B<br>10GE：9.41@1518B |
| RoCE | 100G | Mpps | read：>30Mpps@2B<br>write:>30Mpps@2B<br>send:>30Mpps@2B | read: 31.5Mpps@2B<br>write: 33.2Mpps@2B<br>send: 33.12Mpps@2B | read : 31.02Mpps@2B<br>write: 33.35Mpps@2B<br>send : 32.67Mpps@2B |
| | | 时延 | read：2.0us@2B<br>write：0.85us@2B<br>send：1.5@2B<br>read：3us@4KB<br>write：2.5us@4KB<br>send：2.5@4KB | read：1.51us@2B<br>write：0.83us@2B<br>send：1.14@2B<br>read：2.41us@4KB<br>write：1.70us@4KB<br>send：1.87@4KB | read：1.58us@2B<br>write：0.98us@2B<br>send：1.79@2B<br>read：1.83us@4KB<br>write：1.56us@4KB<br>send：2.29@4KB |
| | | 带宽 | read：线速@1KB<br>write:线速@1KB<br>send:线速@1KB | read: >99.5Gbps@1024B<br>write: >99.5Gbps@1024B<br>send : 99.2Gbps@1024B | read: >99.5Gbps@1024B<br>write: >99.5Gbps@1024B<br>send :97Gbps@1024B |
| SAS | X8*12G | IOPS | 读：800K/S@4KB<br>写：800K/S@4KB | 读：1451.6K/s@4K<br>写：982.37K/s@4K | 读：1031.2K/s@4K<br>写：941.4K/s@4K |
| | | 时延 | 4KB读：800us<br>4KB写：800us | 4KB读：345.5us<br>4KB写：514.94us | 4KB读：489.79us<br>4KB写：536.96us |
| | | 带宽 | 读：8000MB/S@256KB<br>写：8000MB/S@256KB | 读：8539.6MB/s@256KB<br>写：8282.7MB/s@256KB | 读：8539.6MB/s@256KB<br>写：8282.7MB/s@256KB |