4.2.4. 简单单周期的最终实现

将数据通路部件和控制单元结合起来，就完成了本章 MIPS 核心子集的单周期实现，如 Figure 4-24 所示：

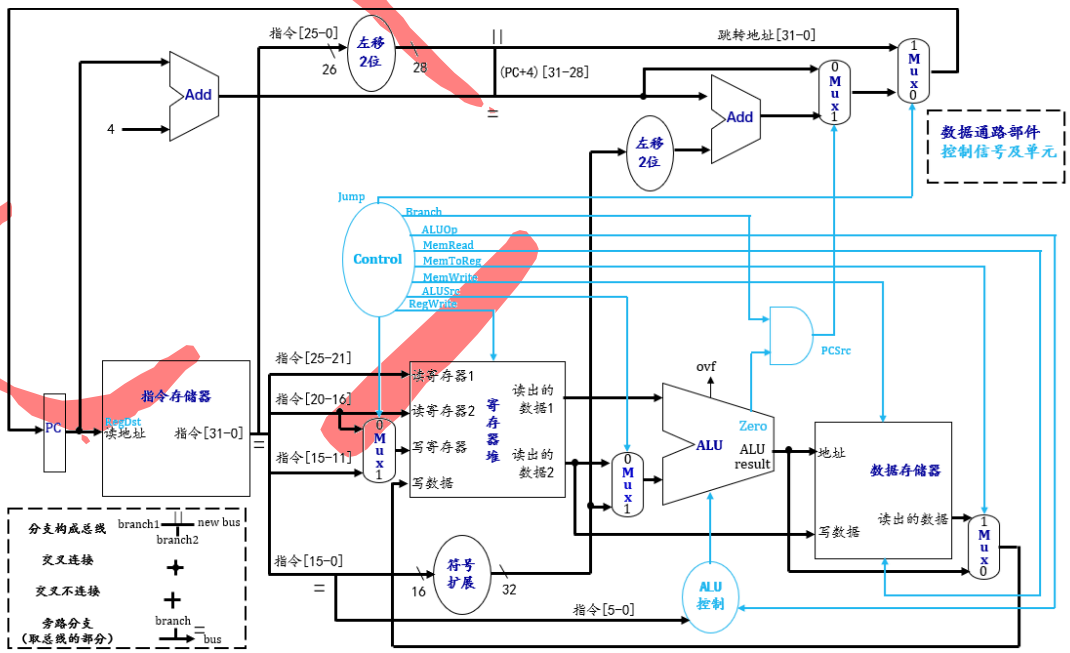

Figure 4-24 The Simple Datapath with the Control Unit

Example 4.1

假设处理器取入的指令字为 0XAC62 0014，试考察该指令在单周期数据通路中如何执行：

(1) 在 Figure 4-24 基础上画出该指令执行时的数据通路；

(2) 该指令执行后，新的 PC 地址是什么；

(3) 给出指令执行时每个多路选择器的输入输出值；

(4) 两个加法器及 ALU 的输入是什么；

(5) 控制单元 ALU 的输入输出是什么；

(6) 寄存器堆输入（包括控制信号）是什么。

取指后译码 0XAE72 0014=0B101011 10011 10010 00000 00000 010100

查表可知，该指令是 sw $s2/Reg[0x12], 20($s3/Reg[0x13])

(1)

图中高亮部分即为 sw 指令的数据通路

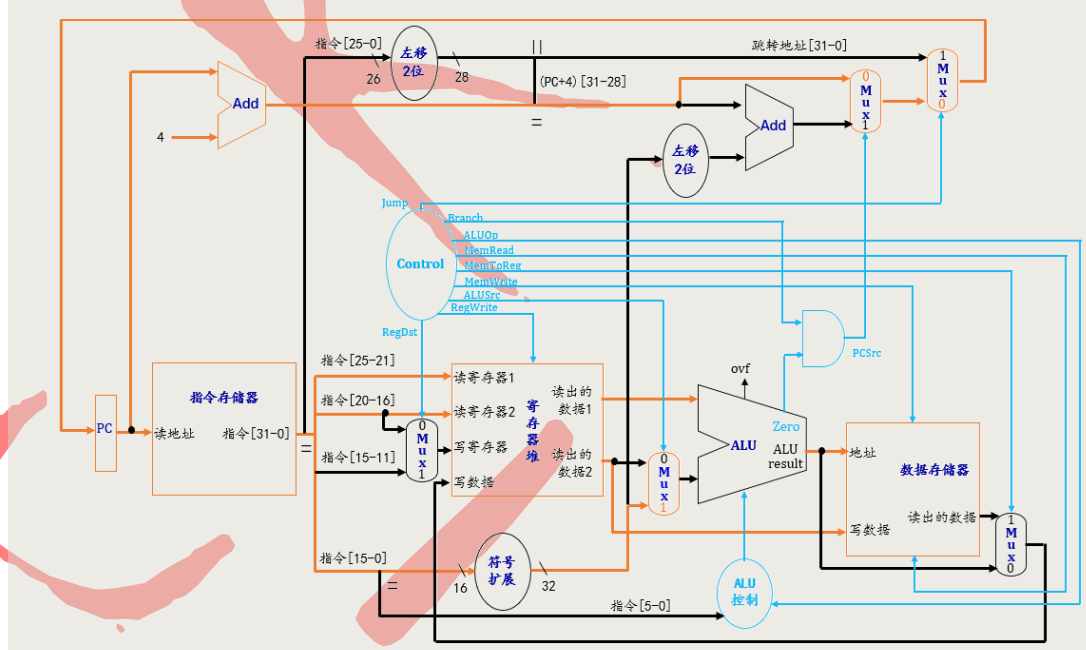

Figure 4-25 The Datapath in Operation for a Save Instruction

(2)

PC+4

(3)

Mux	Control	Input0	Input1	Output
ALUSrc	1	Reg[0x12]	0X014	0X014
RegDst	x	Reg[0x12]	Instruction[15-11]	<undefined>
MemToReg	x	Reg[0x13]+0X014	<undefined>	<undefined>
PCSrc	0	PC+4	PC+0X054	PC+4
Jump	0	PC+4	jump address	PC+4

(4)

Unit	Src1	Src2
ALU	Reg[0x13]	0x014
PC+4 Adder	PC	4
Branch Adder	PC	0X050

(5)

输入：ALUOp=00

输出：ALU Control Lines=0010

(6)

控制信号 RegWrite=0

读寄存器 1=0x13

读寄存器 2=0x12

写寄存器 <undefined>（无关）

写入数据 <undefined>（无关）

4.2.5. 评估单周期实现

现在再次回看本章 MIPS 指令核心子集的单周期实现的五个步骤。

指令	IF	ID	EX	MEM	WB
R 型	取指 PC+4	取源操作数 rs 和 rt	执行运算	——	将 ALU 运算结果写回 rd
lw		取基址寄存器 rs 对偏移量符号扩展	计算内存地址	访存，读地址指向的数据	将访存数据写回 rt
sw				访存，将 rt 的数据写入地址指向位置	——
beq		取 rs 和 rt 字地址符号扩展 字地址左移两位	比较 rs 和 rt 计算分支目标地址	更新 PC	——
j		字地址左移两位 与 PC+4 高 4 位拼接	更新 PC	——	——

Figure 4-26 Instructions' Implementation in a Single-Cycle

在单周期实现中，每条指令都在一个时钟周期内完成，CPI=1。

时钟周期必须满足所有指令中最坏的情况（完成指令时间最长），也就是本章 MIPS 核心子集的 lw，经历了全部 5 个阶段。此外，由于在同一时钟内不能复用某些功能单元（如加法器），因此我们必须使用多个这类功能单元。所以，单周期实现不仅时钟周期太长，从而导致 CPU 执行时间（性能的评价指标）过长，而且浪费了空间上的资源。

但单周期数据通路简单且易于理解。

Example 4.2

假设在实现一个处理器的数据通路时，逻辑模块的延时如下：

I-Mem/ D-Mem	Register File	Mux	ALU	Adder	Single Gate	Register Setup	Register Read	Sign Extend	Control
250ps	150ps	25ps	200ps	150ps	5ps	20ps	30ps	50ps	50ps

“寄存器读”是指时钟上升沿到来后寄存器值出现在输出端所需的时间，只适用于 PC。

“寄存器建立”是指时钟上升沿到来前，寄存器输入数据必需保持稳定的时间，适用于 PC 和寄存器堆。

(1) 分别计算 R 型、lw、sw、beq 和 I 型其他指令（逻辑和算术指令）的延迟时间；

(2) 该 CPU 的最小时钟周期是多少？

(3) 假设有一个新的 CPU，其每条指令的时钟周期都不同。对于下面指令的使用频度，该新 CPU 相对于本题中的 CPU 的性能加速比是多少？

R-Type / I-Type(not lw/sw/beq)	lw	sw	beq
52%	25%	11%	12%

(1)

所有指令首先从 PC 中获取当前指令地址，并从指令存储器取值 30+250=280ps。

然后将指令映射为控制信号传输到被控制的单元（本题忽略不计）

开始分路径并行执行

计算指令周期，要充分考虑路径并行（关键路径，AOE 网），使指令周期最小。

R 型：

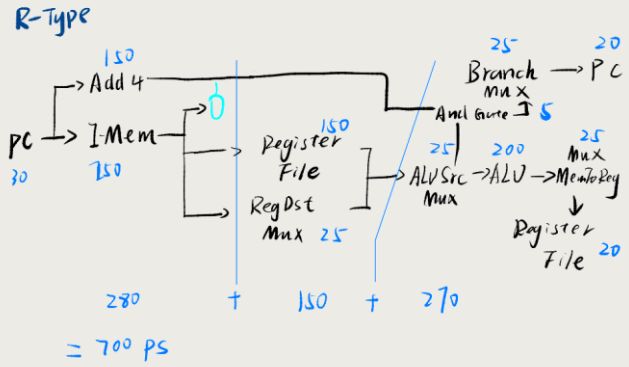

Figure 4-27 The Datapath and its delay for a R-Type Instruction

lw：

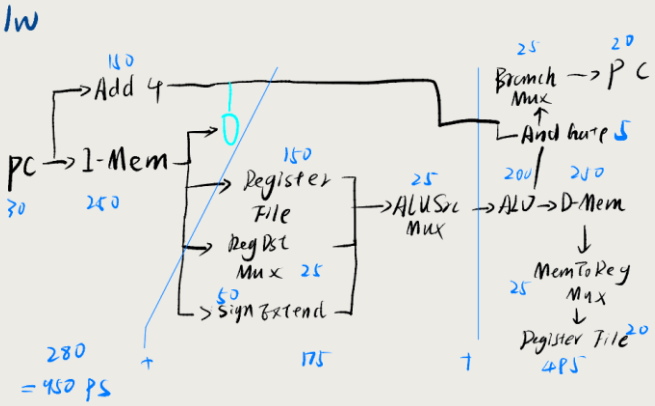

Figure 4-28 The Datapath and its delay for a lw Instruction

sw：

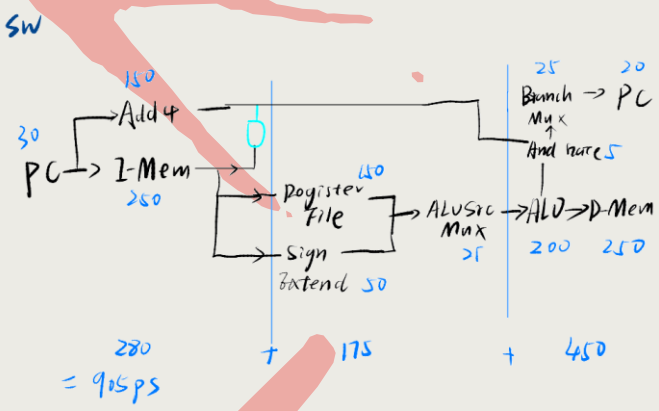

Figure 4-29 The Datapath and its delay for a sw Instruction

beq：

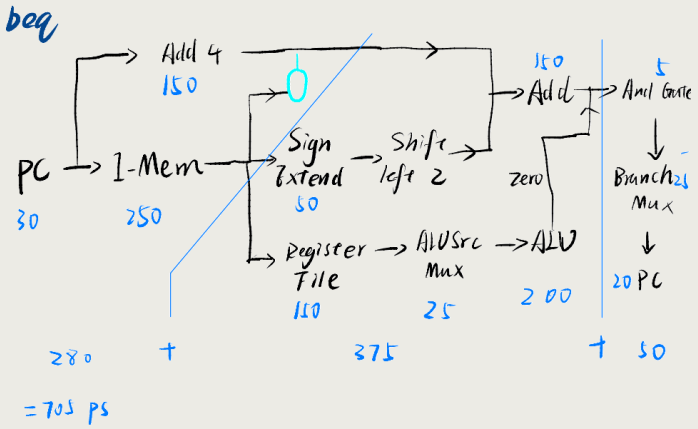

Figure 4-30 The Datapath and its delay for a beq Instruction

I-Type（not lw/sw/beq）：

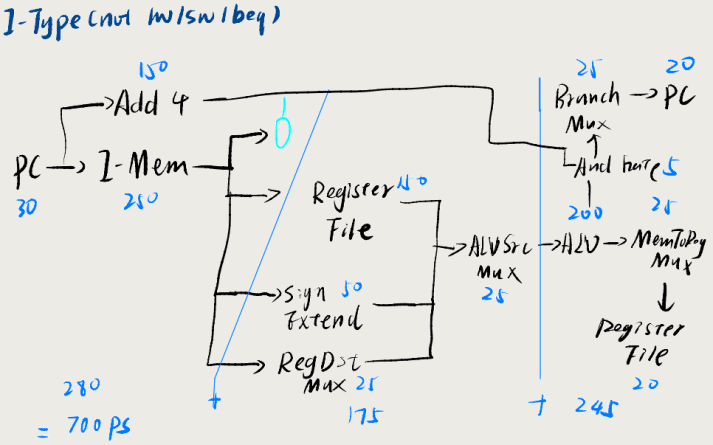

Figure 4-31 The Datapath and its delay for a I-Type Instruction

(2)

CPU 时钟最小周期等于最慢指令的时间，即 lw 指令，950ps

(3)

原 CPU 时间=$IC \times CPI \times T_{old}$

$= IC \times 1 \times 950ps$

新 CPU 时间=$IC \times CPI \times \sum T_i$

$= IC \times 1 \times (52\% \times 700 + 25\% \times 950 + 11\% \times 905 + 12\% \times 705)ps$

加速比=$\frac{950}{785.6} = 1.174$

Note

以上是对于参考答案的理解。显然，参考答案为了课本知识的连贯性，有意地进行了阶段的划分。比如 ALU Src 多选器必须等到译码阶段完成后才能进行选择，寄存器写信号可以等到写回阶段作用，写寄存器的选择也不必在译码阶段决定（学过流水线再回看会比较清楚这些刻意的阶段划分）。

实际上，后两者对关键路径上的时延计算没什么影响，因为都是为了最大并行。而 ALU Src 多选器必须等到译码阶段完成后才能进行选择却是为了流水线结构的整齐，实际上并非最大并行。

对于组合逻辑电路，任何输入都会有对应的输出，只不过我们希望一个稳定的、我们所需要的输出。考虑 I 型指令，一旦 ALU Src 多选器的控制信号让其输出 1 所对应的数据，无须等待寄存器堆输出多选器的另一个输入，即可得到一个稳定的正确输出，所以它也可以和寄存器堆并行。

Figure 4-25 是一个 sw 指令，将其中的 ALU Src 多选器放到寄存器堆下方和符号扩展一起同寄存器堆并行，也是可以的。如此，总时延会再次减少一个多选器的时延 25 ps（控制单元和多选器只能串行，但 50+25<150）。其他的 I 型指令也是类似，也更符合实际。

4.2.6. 多周期实现

在多周期实现（multi-cycle implementation）中，一条指令可以在不同的时钟周期多次使用同一个功能部件，这种共享可以减少所需的硬件数量。区别于 Example 4.2 中假设的 CPU，一般一个 CPU 只有一个时钟周期。多周期实现不是有多个时钟周期，而是一条指令的不同阶段分在多个周期实现。

例如，在单周期实现中的 5 个阶段，我们可以缩减时钟周期到 5 个阶段实现时间的最大值。这样 R 型、lw、sw、beq 和 j 的 CPI 虽然增大到 4、5、4、4、1，但时钟周期大幅缩短，CPU 执行时间减小，性能反而更高。

虽然多周期实现能够降低硬件开销，然而当今几乎所有的芯片都采用流水线技术来提升性能。