计算机组织与结构 II：CPU 设计文档

李勃璘
吴健雄学院

版本：1.0
日期：2025 年 3 月 22 日

摘　要

本设计文档详细阐述了一款基于五级流水线的 CPU 体系结构及其 Verilog 实现。CPU 采用取指（IF）、译码（ID）、执行（EX）、访存（MEM）、写回（WB）五级流水线，以提高指令执行效率。文档首先介绍了 CPU 的总体架构，包括时钟与复位机制、关键存储单元、内部数据通路与控制信号，并详细说明了指令集架构及其编码格式。随后，针对流水线执行过程中可能出现的数据冒险、控制冒险、流水线暂停等问题，提出了旁路（Forwarding）、分支预测（Branch Prediction）、冒险检测（Hazard Detection）等优化方案，并给出了相应的 Verilog 设计。最后，文档分析了 CPU 与内存、总线及外部控制信号的交互方式，并探讨了 Verilog 在 FPGA 上的实现方案。该设计通过流水线优化与高效控制信号管理，提升了 CPU 的吞吐率，为后续硬件优化和扩展提供了良好的基础。

目录

表格目录

1 概述

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.[1]

2 体系结构设计

2.1 总体架构

CPU 由控制单元（CU），逻辑运算单元（ALU），内存（Memory）和寄存器组（Registers）组成，除内存以外，其余单元由被 CU 生成的控制信号控制的数据通路（Data Path）连接。另外，MAR 和 MBR 分别还和地址总线、数据总线相连接，用于与内存交互。控制单元和内存都和控制总线相连接，用于与外部控制信号交互。

为简单起见，CPU 的计算全部为 **16 位定点有符号数计算**。

这里需要一张图!!!

2.2 指令集架构

指令集是指 CPU 能够对数据进行的所有操作的集合。每一条指令都可以被解释为寄存器与寄存器、内存、I/O 端口之间的交互。交互方式由 CU 中的微指令（Micro-operation）给出，且每一条微指令都需要一个时钟执行（如不进行优化）。

2.2.1 位宽设计

地址段长为 **8** 位，指令码（Opcode）宽度为 **8** 位。因此，每一条指令的位宽为 **16** 位。

2.2.2 寻址方式

寻址方式指对地址段数据的解释方式。寻址方式由对应指令指定，支持表 1 中的全部寻址方式。由于给定的指令集高四位均空闲，使用最高位存储支持的寻址方式。

表 1: 指令集支持的寻址方式

寻址方式	描述	最高位
立即数寻址	地址字段是操作数本身，数据为补码格式	0
直接寻址	地址字段为存放操作数的地址	1

[1] 手搓 CPU 是人类文明的伟大工程——沃兹基硕德

2.2.3 指令集支持的指令

指令集共支持 13 条不同的指令，列于表 2。每一条指令包含一个指令码，使用 16 进制格式存储。指令码的最高位为 1 时，寻址方式为立即数寻址；指令码的最高位为 0 时，寻址方式为直接寻址。

评论. 指令中含 * 的仅支持直接寻址，因为立即数寻址对这些指令无意义。

表 2: 指令集包含指令及功能（直接寻址下）

助记符	指令码 (HEX)	描述
*STORE X	01	[X] ← ACC
*LOAD X	02	ACC ← [X]
ADD X	03	ACC ← ACC + [X]
SUB X	04	ACC ← ACC - [X]
JGZ X	05	ACC ≥ 0 ? PC ← X : PC ← PC + 1
JMP X	06	PC ← X
*HALT	07	Stop
MPY X	08	MR, ACC ← ACC * [X]，MR 用于存储高 16 位
AND X	09	ACC ← ACC & [X]
OR X	10	ACC ← ACC | [X]
NOT X	11	ACC ← ~[X]
SHIFTR	12	ACC ← ACC >>> 1，算术右移
SHIFTL	13	ACC ← ACC <<< 1，算术左移

2.3 CPU 内部寄存器

该部分描述 CPU 内部寄存器的含义、存储格式和数据被解释为的格式。这些寄存器通过 CPU 的内部数据通路相连接。寄存器操作是 CPU 快速操作的核心。

表 3: CPU 内部寄存器的含义、总存储条数、单位位宽和数据解释格式

寄存器	含义	条数	位宽	数据解释格式	归属模块
PC	程序计数器，存储当前指令地址	1	8	指令码（Opcode）	/
MAR	内存地址寄存器，存储要访问的内存地址	1	8	地址码（Address）	/
MBR	内存缓冲寄存器，存储从内存读取或写入的数据	1	16	二进制补码	/
IR	指令寄存器，存储当前正在执行的指令	1	8	指令码（Opcode）	/
BR	通用寄存器，存储 ALU 计算中间结果	1	16	二进制补码	ALU
ACC	累加寄存器，存储 ALU 运算结果	1	16	二进制补码	ALU
MR	乘法寄存器，存储 ALU 乘法高 16 位	1	16	二进制补码	ALU
CM	控制存储器，存储微指令控制信号	未定	14	控制信号	CU
CAR	控制地址寄存器，指向当前执行的微指令	1	4	CM 中的条数下标	CU
CBR	控制缓冲寄存器，存储当前微指令的控制信号	1	14	控制信号	CU

除上述寄存器以外，ALU 进行运算时还会更改**状态寄存器**（Flags），用于 CU 进行条件判断。例如，JGZ 命令需要判断上一步的运算结果是否是 0，CU 可以直接通过状态寄存器中的 ZF（Zero Flag）寄存器直接进行判断。本设计中使用的所有状态寄存器见表 4，它们都直接连向 CU，通路不受控制信号的控制。

表 4: 状态寄存器列表

寄存器	全称	行为
ZF	Zero Flag	ALU 运算结果（通常为 ACC）为 0 时置 1
CF	Carry Flag	存储算术移位移出的比特（由于有符号数不存储进位）
OF	Overflow Flag	ALU 运算结果发生溢出时置 1
NF	Negative Flag	ALU 运算结果为负数时置 1
MF	Multiply Flag	ALU 此轮运算出现乘法，且使用 MR 寄存器时置 1

2.4 算术逻辑单元 ALU

算术逻辑单元 ALU 负责进行大部分 CPU 内的计算[2]。它通过 MBR 获取运算的第二个数据，计算方式受状态寄存器影响，计算结果存入 ACC 寄存器中（若有乘法则可能存入 MR 寄存器）。

2.4.1 ALU 的数据通路

表 5: ALU 的数据通路

源寄存器	目标寄存器	传递的数据

2.4.2 ALU 的内部控制信号

ALU 与外围寄存器的控制通路见第 2.5.1 节。为了让 ALU 内部获取要执行的运算操作，需要在 ALU 内部根据 IR 提供的指令类型翻译为 ALU 的控制信号。在本设计中，每一种运算都对应一种控制信号。

2.4.3 ALU 的内部控制逻辑

这一部分描述 ALU 的内部控制信号与来自状态寄存器的 Flags 如何控制 ALU 内部数据通路的开关。

2.5 CPU 内部数据通路、控制信号与微操作指令（Micro-Operations）

CPU 中总线、寄存器、内存和 ALU 等关键器件需要以合理的时序执行不同的指令，而这需要数据通路和控制通路共同实现。数据通路负责在各个器件中传递数据，而控制通路负责控制数据通路的开关。因此，在本节中，先定义 CPU 需要的数据通路，再进一步定义控制数据通路的控制信号，最后描述承载控制信号的微指令。

2.5.1 数据通路与控制信号

关键存储单元之间通过数据通路进行连接。每条数据通路都由一位控制信号控制。控制信号为 1 时表示通路打开，数据沿指定流向进行传输。该 CPU 共有 **14** 位控制信号。

[2]PC 自增与 PC 赋值在设计中不引入 ALU。

评论. CU 内部的数据通路不需要受到微指令的控制，因此不需要在此列出。请参照模块设计部分。

表 6: 数据通路与控制信号一览

控制信号位	源寄存器/单元	目的寄存器/单元
0	MAR	地址总线
1	PC	MBR
2	PC	MAR
3	MBR	PC
4	MBR	IR
5	数据总线	MBR
6	MBR	ALU（待处理数据）
7	ACC 和 MR	ALU
8	MBR	MAR
9	ALU	ACC
10	MBR	ACC
11	ACC	MBR
12	MBR	数据总线
13	IR	CU
ALU 内部控制通路		
14	Why am I doing this	Confusing

2.5.2 微操作指令（Micro-Operations）

为了实现指令集中所有指令，需要将指令集的指令分解为多步微操作指令。为了和后续流水线部分对齐，在微操作指令的设计上，也分为和五步流水线相同的步骤。

在本设计中，采用水平微指令（Horizontal Micro-operation）设计。这意味着每一个微操作指令携带所有控制信号位的开关和下一个微操作指令的地址。

所有用逗号隔开的指令均**从左到右顺序执行**，不能并行执行。所有的微操作指令见表 7。

表 7: 控制单元微操作指令一览

阶段	指令	微操作	控制信号
IF	ALL	MAR ← PC ; PC ← PC + 1 ; MBR ← Mem[MAR]	0, 2, 3, 5
ID	ALL	IR ← MBR, CU ← IR	4, 13
EX	ALL EXCEPT MPY	MF 清零（此条附加到所有指令的 EX 步骤）	无
EX	STORE X	MAR ← X	6
MEM	STORE X	Mem[MAR] ← ACC	8
WB	STORE X	无	无
EX	LOAD X	MAR ← X	6
MEM	LOAD X	MBR ← Mem[MAR]	7
WB	LOAD X	ACC ← MBR	10

续下页

表 7:（续表）控制单元微操作指令一览

阶段	指令	微操作	控制信号
EX	ADD X	MAR ← X	6
MEM	ADD X	MBR ← Mem[MAR]	7
EX	ADD X	ALU: ACC ← ACC + MBR ; 更新 ZF, CF, OF, NF	ALU: 控制信号待补充
WB	ADD X	ACC ← ALU 结果	9
EX	SUB X	MAR ← X	6
MEM	SUB X	MBR ← Mem[MAR]	7
EX	SUB X	ALU: ACC ← ACC - MBR ; 更新 ZF, CF, OF, NF	ALU: 控制信号待补充
WB	SUB X	ACC ← ALU 结果	9
EX	MPY X	MAR ← X	6
MEM	MPY X	MBR ← Mem[MAR]	7
EX	MPY X	ALU: ACC, MR ← ACC × MBR ; 更新 ZF, OF, NF, MF	ALU: 控制信号待补充
WB	MPY X	ACC, MR ← ALU 结果	9
EX	AND X	MAR ← X	6
MEM	AND X	MBR ← Mem[MAR]	7
EX	AND X	ALU: ACC ← ACC AND MBR ; 更新 ZF, NF	ALU: 控制信号待补充
WB	AND X	ACC ← ALU 结果	9
EX	OR X	MAR ← X	6
MEM	OR X	MBR ← Mem[MAR]	7
EX	OR X	ALU: ACC ← ACC OR MBR ; 更新 ZF, NF	ALU: 控制信号待补充
WB	OR X	ACC ← ALU 结果	9
EX	NOT X	MAR ← X	6
MEM	NOT X	MBR ← Mem[MAR]	7
EX	NOT X	ALU: ACC ← NOT MBR ; 更新 ZF, NF	ALU: 控制信号待补充
WB	NOT X	ACC ← ALU 结果	9
EX	SHIFTR	ALU: ACC ← ACC ⋙ 1 ; CF ← 位移出位; 更新 ZF, NF	ALU: 控制信号待补充
MEM	SHIFTR	无	无
WB	SHIFTR	ACC ← ALU 结果	9
EX	SHIFTL	ALU: ACC ← ACC ⋘ 1 ; CF ← 位移出位; 更新 ZF, NF	ALU: 控制信号待补充
MEM	SHIFTL	无	无
WB	SHIFTL	ACC ← ALU 结果	9
EX	JMP X	PC ← X	6
MEM	JMP X	无	无
WB	JMP X	无	无
EX	JGZ X	若 ZF=0 且 NF=0，则 PC ← X；否则 PC ← PC	6
MEM	JGZ X	无	无

续下页

表 7:（续表）控制单元微操作指令一览

阶段	指令	微操作	控制信号
WB	JGZ X	无	无
EX	HALT	`enable` ← 0	无
MEM	HALT	无	无
WB	HALT	无	无

2.6 内存（RAM）

内存（RAM）存储指令集和 CPU 保存的数据。内存的大小为 512 Byte，每条存入内存的数据位宽为 16，共能存入 256 条数据。其中，内存地址 0 到内存地址 12 预留为指令集。为了节省内存，不同寻址方式下的相同指令不占用两条内存，由 CU 改变指令的 Opcode。

内存支持总线读写，并受三条总线控制：地址总线、数据总线和控制总线。**控制总线**中的外部控制信号决定在这个周期中内存的读/写状态，是否向数据总线写入，同步时序等功能。CPU 与内存（RAM）通过两条总线交互，分别为**地址总线**和**数据总线**。内存通过读取地址总线决定写入内存中的地址，通过读取数据总线决定写入指定地址中的数据。关于总线的具体配置见 2.7。

2.7 总线与外部控制信号

2.7.1 地址总线

地址总线为 **8** 位单向总线，提供 CPU（即 MAR）到内存的地址传送通路。由于其为单向总线，仅需内存侧读使能信号与 CPU 侧 MAR 的控制信号控制即可，无需复用。

2.7.2 数据总线

数据总线为 **16** 位双向总线，提供 CPU（即 MBR）与内存的双向数据通路。数据总线采用分时复用的方式进行设计。如果 MEM 和 WB 在同一时钟下，会出现流水线冲突，需要考虑是否引入旁路。

2.7.3 控制总线与外部控制信号（需要等待流水线设计好）

外部控制信号是一组单比特信号，通过控制总线控制 CU 和 RAM 的行为，它们受流水线周期的控制置 0 或置 1。CU 和 RAM 通过内部映射决定监视哪些位的信号。外部控制信号主要包括以下功能：

- RAM 读写控制；
- 流水线控制信号。

所有外部控制信号列于表 8。

表 8: 外部控制信号一览

控制信号位/类型	别名	有效模块	高电平时作用	低电平时作用
RAM 读写控制				
0	MemoryWrite	RAM	RAM 写数据总线	无
1	MemoryRead	RAM	RAM 读数据总线和地址总线	无

续下页

表 8: （续表）外部控制信号一览

控制信号位/类型	别名	有效模块	高电平时作用	低电平时作用
分支预测				
2	BranchTaken	CU	执行 Branch	顺序执行
3	Jump	CU	执行 Jump	顺序执行
流水线控制				
4	PipelineStall	CU	流水线暂停（如数据冒险、存储器访问延迟）	无
5	PipelineFlush	CU	流水线清空（如错误预测、异常发生）	无

3 流水线架构与优化策略

3.1 总体架构

为了加速 CPU 的指令执行速度，采用 **5 级同步流水线**完成 CPU 执行指令的全流程。分别为：取指令（IF），指令译码（ID），指令执行（EX），内存访问（MEM）和写回寄存器（WB）五个阶段。流水线的五个阶段如下所示。

$$\text{IF} \rightarrow \text{ID} \rightarrow \text{EX} \rightarrow \text{MEM} \rightarrow \text{WB}$$

流水线各阶段的主要工作如下：

- **IF(Instruction Fetch):** 从指令存储器中取出指令，同时确定下一条指令地址（指针指向下一条指令）；
- **ID(Instruction Decode):** 翻译指令，同时让计算机得出要使用的寄存器，或者让立即数进行拓展（方便后续指令执行），亦或者（转移指令）是给出转移目的寄存器与转移条件；
- **EX(Execution):** 按照微操作指令指示打开数据通路。
- **MEM(Memory):** 若为 `LOAD/STORE` 指令，这个阶段就要访问存储器。此外，指令从 EX 向下执行到 WB 阶段。另外，在这个阶段还要判定是否有异常要处理，如果有，那么就清除流水线，然后转移到异常处理例程入口地址处继续执行。
- **WB(Write Back):** 将运算结果保存到目标寄存器。

3.2 流水线冒险

包含三种冒险情况：结构冒险、数据冒险、控制冒险（分支冒险）。[1]

结构冒险，即**某指令引用了前一次的运算结果，但其结果值还未产生**。对于这种情况，采用旁路（Forwarding）技术解决。

3.3 流水线优化：分支预测

在本设计中，采用 1 比特分支预测进行流水线优化。分支预测的控制信号由**控制总线**输入到 CU 中。分支预测的步骤如算法 1。

Algorithm 1 1-bit 分支预测

Note: 使用 1-bit 预测器预测分支是否跳转
Input: $PC, BranchTaken$
Output: $NextPC$
Internal Registers: $PredictorBit, BranchTarget$
procedure BRANCH PREDICTION
    if $PredictorBit == 1$ **then**
        $PredictedTaken \leftarrow$ True
        $NextPC \leftarrow BranchTarget$
    else
        $PredictedTaken \leftarrow$ False
        $NextPC \leftarrow PC + 2$
    end if
end procedure
procedure BRANCH RESOLUTION
    if $BranchTaken \neq PredictedTaken$ **then** ▷ 预测错误
        $Flush \leftarrow 1$ ▷ 清空错误指令
        $NextPC \leftarrow$ **if** $BranchTaken$ **then** $BranchTarget$ **else** $PC + 2$
    end if
    Update Predictor:
    $PredictorBit \leftarrow BranchTaken$ ▷ 更新 1-bit 预测器
end procedure

4 模块设计

4.1 时钟、复位与停止信号

CPU 由**全局同步时钟**控制，所有控制逻辑与计算逻辑全部在时钟上升沿进行。CPU 设有**全局异步复位**信号，低电平有效。当异步复位时，内存中除指令集数据以外所有数据清空，所有寄存器清空，控制信号全部归为断开（0）。

当 CPU 执行 07 号指令 HALT 时，CPU 处于**暂停**状态。与复位不同的是，此时所有寄存器不清空，但所有通路断开。在模块中使用 `enable` 信号标识（低电平有效）。恢复程序运行的方法是全局复位或继续运行信号（绑定 FPGA 的按键）。当该按键被按下时，`enable` 信号恢复为 1。

4.2 ALU

4.2.1 ALU 运算结果

ALU 运算结果存放于 MR 寄存器和 ACC 寄存器中。其中 MR 寄存器存放乘法运算的高位结果，ACC 寄存器存放乘法运算的低位结果。ALU 与两个寄存器的数据通路都由 7 号控制位控制。MR 连向 ALU 的高位结果，ACC 连向 ALU 的低位结果。当控制信号 MF 为 1 时

4.3 CU

参考表 6 ，确定指令集中的每一条指令对应的控制信号，并存储到 CU 的内部寄存器中，即可完成 CU 的主要功能设计。

4.4 CPU 内部寄存器

4.4.1 MAR

4.4.2 MBR

4.4.3 IR

每条指令在 ID 阶段被传入 IR 寄存器。IR 寄存器可通过判断最高位来确定寻址方式。即：

- IR 最高位为 1，立即数寻址，在 MEM 阶段不再进行访存，而是直接传入 MBR。
- IR 最高位为 0，直接寻址，在 MEM 阶段进行访存。

4.4.4 PC

4.4.5 ACC

4.5 Memory

5 仿真验证

5.1 时延分析

5.2 并行计算加速比（Speed-up Factor）分析

5.3 激励设置

6 FPGA 实现

References

[1] Ansis11. 基于 *RISC-V* 架构-五级流水线 *CPU*》知乎专栏，访问于 2025 年 4 月 10 日. 2022. URL: https://zhuanlan.zhihu.com/p/453232311.

[2] 赖兆磬. “基于 FPGA 流水线 CPU 的设计与实现”. MA thesis. 桂林电子科技大学, 2008.

A 完整设计代码