

简单多周期 CPU 实验：实验报告

刘泓尊 2018011446 计 84

一、实验目的

1. 加深对 SRAM 时序的理解
2. 掌握指令执行阶段的含义，熟悉执行阶段所需要的信号
3. 理解一个简单程序的执行过程，深入理解多周期 CPU 的特点

二、实验细节

整个项目分为 IF 模块，ID 模块，EX 模块，REG_FILE 模块，IO(MEM)模块和 CPU 控制模块。因为指令存储和数据存储共用一个总线，所以取指和访存不能同时进行，当然目前也没有引入流水的实现。多周期处理器相对于单周期 CPU 的优点是它可以根据指令类型来省略一部分阶段，从而提高运行效率。

时钟使用 50MHz 时钟。

• CPU 控制就是一个大的状态机：

IF 阶段：从 SRAM 读数据，地址是 $pc[21:2]$ 。可以使用多个周期完成。

ID 阶段：译码，判断指令类型，从指令中解析出寄存器地址、立即数等信息。同时在这一阶段判断 `beq` 是否进行跳转。

EX 阶段：按指令类型在 ALU 中执行。ALU 是组合逻辑。

MEM 阶段：将结果写入 SRAM，或者从 SRAM 取数据。（只有 `lw`, `sw` 指令涉及到这个阶段，其他指令可以跳过），可以使用多个周期完成。

WB 阶段：将运算结果写回寄存器。我选择在这一阶段更新 `pc`。

• 其他模块：

REG_FILE: 读寄存器是组合逻辑，写寄存器是时序逻辑。注意寄存器 0 始终要保持在 0。

MEM: 注意 SRAM 的地址以 32 位为单位，需要忽略 CPU 中地址的后两位。

• 对于通过测试的 5 种指令分析如下：

• ADD: `add rd, rs1, rs2`. R 类型，忽略溢出

31	25 24	20 19	15 14	12 11	7 6	0
0000000	rs2	rs1	000	Rd	0110011	

• ORI: `ori rd, rs1, immediate` I 类型

将 12 位立即数 `imm` 符号拓展之后和寄存器 `rs1` 按位或，将结果写入 `rd`

31	20 19	15 14	12 11	7 6	0
Immediate[11:0]		rs1	110	rd	0010011

• LW: lw rd, offset(rs1) I 类型

将 12 位立即数 **offset** 进行符号扩展后和寄存器 **rs1** 相加，并将结果对应地址开始的 32 位字写入寄存器 **rd**

31	20 19	15 14	12 11	7 6	0
offset[11:0]		rs1	010	rd	0000011

• SW sw rs2, offset(rs1) S 类型

将 12 位立即数 **offset** 符号扩展后和寄存器 **rs1** 相加，并将寄存器 **rs2** 中的内容写入对应地址。

31	25 24	20 19	15 14	12 11	7 6	0
offset[11:5]		rs2	rs1	010	offset[4:0]	0100011

• BEQ: beq rs1, rs2, offset

若 **rs1** 和 **rs2** 内容相等，则 **pc** 跳转到 **pc+offset**，否则 **pc** 跳转到 **pc + 4**

注意：beq 指令省略了 **offset** 末尾的 0，所以求 **offset** 时需要在末尾补个 0

31	25 24	20 19	15 14	12 11	7 6	0
offset[12:10:5]		rs2	rs1	000	offset[4:1:11]	1100011

三、实验数据

下图可以看到 **0x100** 开始的地址处已经正确写入 Fib 数列。

Hex Dump	
00000100:	02 00 00 00 03 00 00 00 05 00 00 00 08 00 00 00
00000110:	0d 00 00 00 15 00 00 00 22 00 00 00 37 00 00 00 7...
00000120:	59 00 00 00 90 00 00 00 e9 00 00 00 79 01 00 00 Y.....y...
00000130:	62 02 00 00 db 03 00 00 3d 06 00 00 18 0a 00 00 b.....=.....
00000140:	55 10 00 00 6d 1a 00 00 c2 2a 00 00 2f 45 00 00 U...m.....*/E...
00000150:	f1 6f 00 00 20 b5 00 00 11 25 01 00 31 da 01 00 .o.....%.1...
00000160:	42 ff 02 00 73 d9 04 00 b5 d8 07 00 28 b2 0c 00 B...s.....(...
00000170:	dd 8a 14 00 05 3d 21 00 e2 c7 35 00 e7 04 57 00 !=...5...W...
00000180:	c9 cc 8c 00 b0 d1 e3 00 79 9e 70 01 29 70 54 02 y.p.)pT...
00000190:	a2 0e c5 03 cb 7e 19 06 6d 8d de 09 38 0c f8 0f ~.m...8...
000001a0:	a5 99 d6 19 dd a5 ce 29 82 3f a5 43 5f e5 73 6d ).?.C...sm
000001b0:	e1 24 19 b1 40 0a 8d 1e 21 2f a6 cf 61 39 33 ee .\$.@...!/.a93...
000001c0:	82 68 d9 bd e3 a1 0c ac 65 0a e6 69 48 ac f2 15 .h.....e...iH...
000001d0:	ad b6 d8 7f f5 62 cb 95 a2 19 a4 15 97 7c 6f ab b..... o...
000001e0:	39 96 13 c1 d0 12 83 6c 09 a9 96 2d d9 bb 19 9a 9.....1...-....
000001f0:	e2 64 b0 c7 bb 20 ca 61 9d 85 7a 29 58 a6 44 8b .d.....a..z)X.D...
00000200:	f5 2b bf b4 4d d2 03 40 42 fe c2 f4 8f d0 c6 34 .+.M..@B.....4
00000210:	d1 ce 89 29 60 9f 50 5e 31 6e da 87 91 0d 2b e6 (...)`P^1n....+...
00000220:	c2 7b 05 6e 53 89 30 54 15 05 36 c2 68 8e 66 16 .{.nS.0T..6.h.f...
00000230:	7d 93 9c d8 e5 21 03 ef 62 b5 9f c7 47 d7 a2 b6 }.....!..b...G...
00000240:	a9 8c 42 7e f0 63 e5 34 99 f0 27 b3 89 54 0d e8 ..B~.c.4...'.T...
00000250:	22 45 35 9b ab 99 42 83 cd de 77 1e 78 78 ba a1 "E5...B...w.xx...
00000260:	45 57 32 c0 bd cf ec 61 02 27 1f 22 bf f6 0b 84 EW2....a.'."....
00000270:	c1 1d 2b a6 80 14 37 2a 41 32 62 d0 c1 46 99 fa ..+...7*A2b...F...
00000280:	02 79 fb ca c3 bf 94 c5 c5 38 90 90 88 f8 24 56 .y.....8....\$V...
00000290:	4d 31 b5 e6 d5 29 da 3c 22 5b 8f 23 f7 84 69 60 M1...).<"[.#.i...
000002a0:	19 e0 f8 83 10 65 62 e4 29 45 5b 68 39 aa bd 4c eb.)E[h9...L
000002b0:	62 ef 18 b5 9b 99 d6 01 fd 88 ef b6 98 22 c6 b8 b....."....
000002c0:	95 ab b5 6f 2d ce 7b 28 c2 79 31 98 ef 47 ad c0 ...o-.(.y1..G...

四、思考题

1. 指令同样被分为 5 个执行阶段，那么多周期相比于流水线的不同地方在哪里？（流水线多做了什么事情？）

流水线的 5 个模块是并行执行的，可以看做每个模块都是一个状态机，它们同步工作。同时每个阶段之后设置了流水线寄存器，保存上一个节拍的输出结果。

对于多周期来讲，因为 5 个阶段串行执行，所以不存在太多的数据冲突、结构冲突和控制冲突等，指令存储器和数据存储器也可以共用一个总线，5 个阶段称为一个大状态机即可。

但是对于流水线 CPU，要设计合理的流水线暂停和数据旁路等，来解决可能出现的冲突。并且因为取指和访存同步进行，需要有指令总线 and 数据总线来避免冲突。

2. 在实验的过程中遇到什么样的困难，是如何解决的。

- a. 因为不太熟悉硬件设计，为了把每个模块独立出来，进行了很多次代码重构，最后勉强将每个阶段做成一个模块，最后在顶层文件中用 `wire` 连接起来
- b. 一开始并不了解 `latch`，导致一直无法调通代码。最后根据老师和助教为微信群中的提醒才得以解决。
- c. 仿真尚不熟练，只会进行很基本的仿真操作，花费了一些时间。

五、实验小结

有了上次 SRAM 实验的基础，本次实验上手起来没有那么困难了，我主要的时间花在 CPU 的设计上，其实先把实现细节想好，实现起来并没有太大阻力。此外我复用了上一个实验的 SRAM 模块，也给本次实验减轻了不少压力。

本次实验初步实践了一个简单的 CPU，收获良多。老师和助教们在微信群里对 verilog 规范的提醒也让我收益很多，感谢老师和助教的悉心指导！