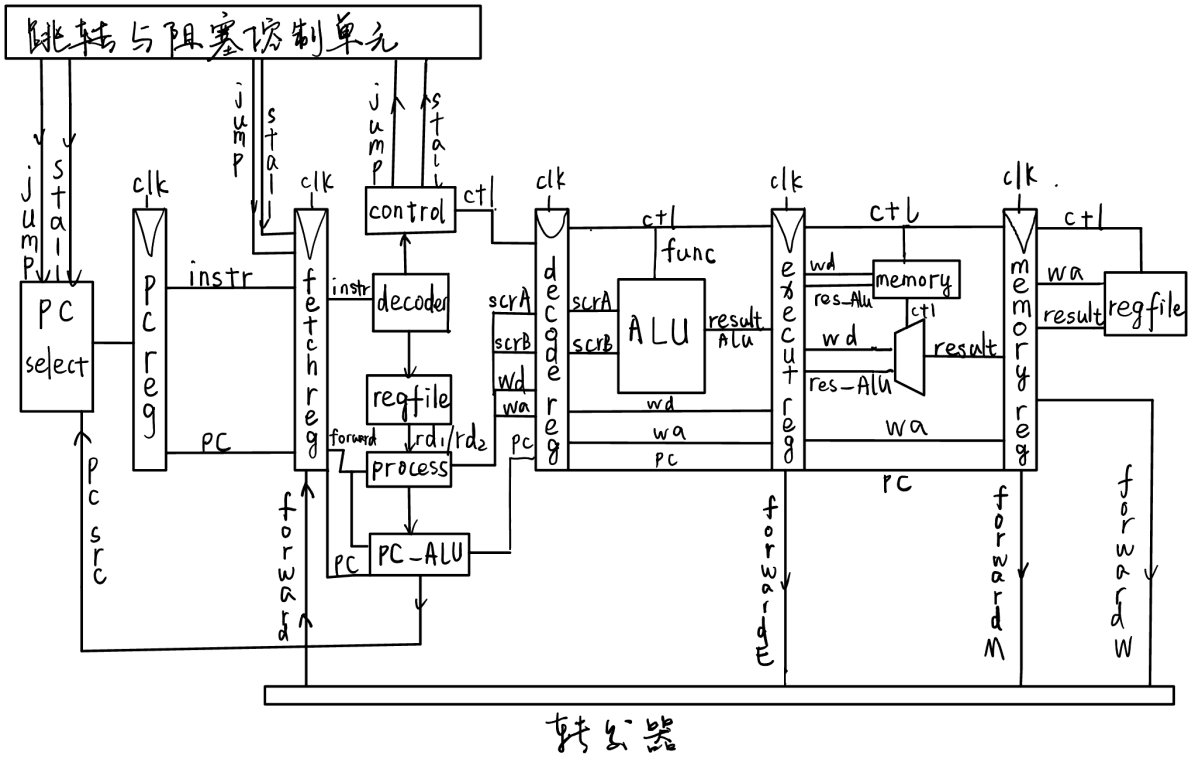
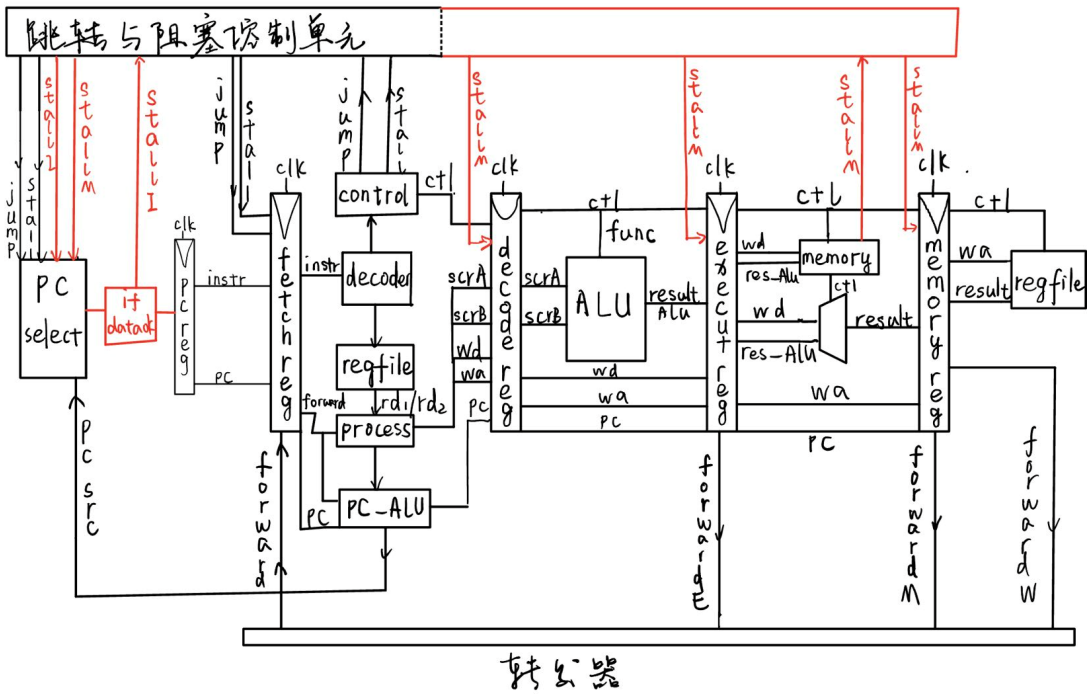
支持握手总线的CPU 实验报告

目录

内容

CPU电路图对比





两次实验CPU电路图对比。红色是新添部分。

测试通过结果截图

```
2K performance run parameters for coremark.
CoreMark Size    : 666
Total time (ms)  : 16579
Iterations       : 10
Compiler version : GCC9.4.0
seedcrc          : 0xe9f5
[0]crclist       : 0xe714
[0]crcmatrix     : 0x1fd7
[0]crcstate      : 0x8e3a
[0]crcfinal      : 0xfcaf
Finised in 16579 ms.
==================================================
CoreMark Iterations/Sec 0
Run dhrystone
Dhrystone Benchmark, Version C, Version 2.2
Trying 10000 runs through Dhrystone.
Finished in 14055 ms
==================================================
Dhrystone PASS          1 Marks
                    vs. 100000 Marks (i7-7700K @ 4.20GHz)
Run stream
------------------------------------------------------------
STREAM version $Revision: 5.10 $
------------------------------------------------------------
This system uses 8 bytes per array element.
------------------------------------------------------------
Array size = 2048 (elements), Offset = 0 (elements)
Memory per array = 0.0 MiB (= 0.0 GiB).
Total memory required = 0.0 MiB (= 0.0 GiB).
Each kernel will be executed 10 times.
 The *best* time for each kernel (excluding the first iteration)
 will be used to compute the reported bandwidth.
------------------------------------------------------------
* checktick: start=15.142000
* checktick: start=15.184000
* checktick: start=15.226000
* checktick: start=15.268000
* checktick: start=15.310000
* checktick: start=15.352000
* checktick: start=15.395000
* checktick: start=15.437000
```

```
                       **
                      * *
                       *
[src/cpu/cpu-exec.c,320,cpu_exec] nemu: HIT GOOD TRAP at pc = 0x00000000800152c0
[src/cpu/cpu-exec.c,321,cpu_exec] trap code:0
[src/cpu/cpu-exec.c,62,monitor_statistic] host time spent = 12,939,554 us
[src/cpu/cpu-exec.c,64,monitor_statistic] total guest instructions = 59,171,279
[src/cpu/cpu-exec.c,65,monitor_statistic] simulation frequency = 4,572,899 instr/s
Program execution has ended. To restart the program, exit NEMU and run again.
```

为了支持随机延迟，流水线的改动

为了支持随机延迟，流水线的改动主要聚集在阻塞控制单元上。

阻塞控制单元的改动

如CPU电路图所示，本次实验新增两个阻塞信号$stallI$、$stallM$，分别表示访问指令内存阶段产生的阻塞和访问数据内存阶段产生的阻塞。

在访存阶段，由于会产生随机延迟，因此在等待数据期间，流水线需阻塞。由于访问指令内存和数据内存所在的流水线阶段不同，故其阻塞逻辑也有些微不同。

- $stallI$

```
assign stallI = ireq.valid && ~iresp.data_ok;
```

$stallI$在$fetch$阶段产生，依照助教给的思路，其产生逻辑如上。当$stallI$生效时，此时需要阻塞的阶段应当只有$fetch$，而不应该刷新后面阶段的数据，否则会造成数据丢失。因此，$stallI$仅需接回pc选择器上，使得$pc-4$以保持一直取得这一条指令；同时，$fetch$阶段应该向后传递一个空指令NOP，使得阻塞期间流水线空转，直至数据准备完毕。$stallI$逻辑的实现是简单的。

- $stallM$

$stallM$逻辑的实现相较于前者略复杂。由于$stallM$在$memory$阶段产生，因此其需要的效果是：当延迟产生时，$memory$阶段之前的阶段所有数据流水应该停滞且保留，$memory$阶段之后的阶段即写回阶段应该读到一个空指令以维持流水线空转。其产生逻辑同$stallI$。

```
assign stallM = dreq.valid && ~dresp.data_ok;
```

$stallM$应该接到所有流水线寄存器上，并针对不同阶段流水线寄存器生成不同的效果。此外还需接入pc选择器上，逻辑同$stallI$，使得$pc-4$以保持一直取得这一条指令。

对于$fetch$、$decode$、$excute$阶段的寄存器，其逻辑均相同，当$stallM$生效时，需要让数据流水停滞且保留。以$excute$阶段为例：

```
else if(stallM)
begin
    dataE_nxt <= dataE_nxt;
end
else
begin
    dataE_nxt <= dataE;
end
```

其余两个阶段同理。

对于写回阶段的控制，仅需控制$memory$阶段的输出或在$memory_reg$中调整传入写回阶段的数据。本次实验采用的是后者，逻辑很简单，当$stallM$生效时，将$memory_reg$要传入写回阶段的指令改为NOP即可。由于上次实验在控制单元中设置了nop_signal信号，因此只需令`dataM_nxt.ctl.nop_signal <= 1'b1`即可。

- 与旧$stall$信号的兼容问题

回顾旧$stall$，可知其仅在需要访问数据内存且需要写回寄存器的指令出现时产生，因为这个指令会访问数据内存，故其一定会产生$stallM$，因此两个阻塞信号将会重叠，而旧$stall$信号会使得$pc-4$并且使$fetch$阶段向下传递一个空指令，我们要考察操作的兼容性。对于pc，我们只需要在pc选择器里将两个阻塞信号用或逻辑连接即可，这样就不会出现$pc-8$的情况。对于空指令，其实不需要管它，因为在$stallM$生效时，流水线会滞留，此时$excute_reg$里的内容与$excute$操作的结果根本无关，故不会对流水线产生任何影响。

那么，既然旧$stall$信号的功能被$stallM$信号完全覆盖，是否可以将旧$stall$信号删除呢？我认为没有必要，因为万一某一个周期的访存即时便获得了数据，那么旧$stall$信号产生的阻塞就十分重要了。

其他改动

由于实际上内存访问只有一个接口，故当指令访存与数据访存同时生效时，需要提供一个先后顺序，这就需要一个仲裁器。本次实验使用了助教提供的仲裁器，没有进行个人编写。

总结

本次实验通过扩展阻塞控制单元，新增两个阻塞信号，解决了握手总线的延迟问题；同时，本次实验还新实现了一系列指令，扩展了CPU功能。

本实验报告提供了两次实验的CPU对比图，并对为支持随机延迟而进行的流水线改动进行了解释。

谢谢助教提供的仲裁器代码、$readdata$代码以及$writedata$代码。

> 陈义桐 20CS 20307130250