# 计算机体系结构 2022年春

# Project 5：处理器综合设计

## 目录




**教师：**

邹卓 zhuo@fudan.edu.cn

**助教：**

石勤振 qzshi19@fudan.edu.cn

朱子凯 zkzhu20@fudan.edu.cn

刘从阳 liucy20@fudan.edu.cn



最后一次修订时间：2022年6月1日

# 任务说明：

本实验要求你使用Verilog HDL或VHDL实现一个比教材第 4 章中Y86-64更简单的处理器，简称为处理器Z。这有助于你更好得理解一个处理器的基本系统架构，明白如何处理系统中的数据冒险，了解不同指令在处理器中的行为以及产生的结果。

本实验分为两大部分。5个基础任务要求你按照顺序实现一个基本的处理器，4个进阶任务则要求你处理系统中可能存在的数据冒险并在处理器Z的基础上拓展指令集。

# 任务要求与评分细则：

1. 作业共分为5 + 4个任务，请务必按照顺序完成。
2. 要求使用 **Vivado**完成本次实验，推荐使用Vivado17.4。
3. 在开始作业之前，请务必仔细阅读配套的是实验手册，并按照实验手册的指导熟悉软件操作以及使用Vivado 完成testbench仿真的具体流程。
4. 可以使用verilog HDL 或 VHDL完成本次作业。
5. 最终作业的提交仅需要一份**PDF**的实验报告，命名要求：**学号-姓名-处理器报告.pdf**。
6. 报告中应包括对这9个任务的设计思路，数据通路，testbench代码及代码说明，Vivado仿真结果截图及结果说明。
7. 报告的**附录**中应包括一份最终实现的源代码及代码说明。
8. **评分细则：本次作业共计 18分**
    - 5个基础任务每个任务的设计思路，数据通路，testbench代码以及代码说明，Vivado仿真结果及结果说明各占0.4分，共计 8 分。
    - 进阶任务1和进阶任务2各占 1 分，进阶任务2占4分，进阶任务3占3分，共计9分。需要体现设计思路，关键代码及代码说明，testbench代码以及代码说明，Vivado仿真结果及说明。
    - 9个任务的源代码及代码说明（注释）占 1 分。
    - 如存在抄袭，则抄袭者与被抄袭者均为 0 分。

# 处理器Z说明：

1. 处理器Z在基础任务中只包括取指、译码、执行、写回、更新PC五个操作。(Y86-64处理器包括如下六个操作：取指、译码、执行、访存、写回、更新PC。)
2. 处理器Z将指令字长和数据字长均固定为 4 个字节 (32bit)。 因此对于取指操作，每个时钟周期只需要从ram中读取 4 个字节的指令即可。
3. 指令集说明：处理器Z只包括 5 条指令，分别为IRMOV， ADD， SUB， AND， XOR。(Y86-64处理器的指令集如书4.1.2所示。)
4. 寄存器说明：处理器 Z 的片上寄存器只包括 8 个 32 位寄存器，分别命名为 %r0， %r1， ...， %r7。
5. 指令编码说明：处理器Z的指令长度固定为 4 个字节(32bit)
    - 第一个字节(31:24)表示指令类型，高 4 位表示指令代码icode，低 4 位表示指令功能ifun；其中IRMOV的指令代码为 1 ， 功能代码为 0 ； ADD的指令代码为 2 ， 功能代码为 0 ； SUB的指令代码为 2 ， 功能代码为 1 ； AND的指令代码为 2 ， 功能代码为 2 ； XOR的指令代码为 2 ， 功能代码为 3 。

- 第二个字节(23:16)表示操作数所在寄存器的ID，其中高 4 位表示rA，低4 位表示rB；
- 最后两个字节(15:0)表示立即数valC。(Y86-64处理器的指令编码如书 4.1.3所示。)
- 例：指令AND %r4， %r5，将被表示为22_45_00_00；指 令 IRMOV $16， %r0，将被表示为10_f0_00_10。

# 任务 1： 实现取指以及更新PC操作

## 具体要求：

1. 任务 1 只需要实现取指及更新PC即可。取指操作：处理器Z在每个时钟周期从存储器中读取 4 个字节的指令；更新PC：由于指令字长的固定，因此PC指针每个时钟周期只需要加 1 即可。
2. 首先先创建两个文件，一个是处理器Z的顶层文件processor.v，另一个是存储器文件ram.v。
3. 对于ram.v，用来存放和读取指令。
   - 端口描述：

```
module ram(
    input clock,
    input [8:0] addr,
    input wr,
    input [31:0] wdata,
    input rd,
    output reg[31:0] rdata
);
```

   - 功能描述：对于写入数据，当写使能信号wr有效时，将数据wdata写入地址为addr的存储单元中；对于读出数据，读使能信号rd有效时，将地址为addr的存储单元中的数据输出到数据总线rdata上。
4. 对于processor.v
   - 端口描述：

```
module processor(
    input clock,
    input [8:0] addr,
    input wr,
    input [31:0] wdata,
    input working,
    output [3:0] icode,
    output [3:0] ifun,
    output [3:0] rA,
    output [3:0] rB,
    output [15:0] valC
);
```

   - 功能描述：当working有效时，处理器Z在每个时钟周期从存储器中取出一条 32 位的指令，并解析指令，最终输出；当working无效时，并且在写使能信号wr有效时， 通过addr和wdata向指令存储器中写入数据。

## 仿真要求：

编写testbench 以完成仿真，时钟周期为50MHz，在前 5 个时钟周期令working信号为 0 ，使处理器不工作，并依次向指令存储器地址为0-4的存储单元中写入(IRMOV $16， %r0)， (ADD %r0， %r1)，(SUB %r2， %r3)，(AND %r4， %r5)， (XOR %r6， %r7)这 5 条指令。然后令working信号为 1 ，使处理器工作，完成这 5 条指令的取指操作，最终输出每个时钟周期取指的结果。

# 任务 2： 实现写回操作及指令IRMOV的操作

## 具体要求：

1. 在任务 1 完成的基础上，增加一个寄存器读写regfile.v 文件，并修改processor.v 文件，因此你的设计中应该至少包括以下文件，processor.v，ram.v，regfile.v。
2. 对于regfile.v
   - 端口描述：

```
module regfile(
    input [3:0] dstE,
    input [31:0] valE,
    input [3:0] dstM,
    input [31:0] valM,
    input reset,
    input clock,
    output [31:0] r0,
    output [31:0] r1,
    output [31:0] r2,
    output [31:0] r3,
    output [31:0] r4,
    output [31:0] r5,
    output [31:0] r6,
    output [31:0] r7
);
```

   - 功能描述：采用同步复位，当reset有效，将寄存器r0-r7的数据清零；每个时钟周期，将数据valE写入寄存器ID为dstE的寄存器；每个时钟周期，将数据valM写入寄存器ID为dstM的寄存器；每个时钟周期输出寄存器r0-r7的数据。
3. 对于processor.v，
   - 端口描述：

```
module processor(
    input clock,
    input [8:0] addr,
    input wr,
    input [31:0] wdata,
    input working,
    output [31:0] r0,
    output [31:0] r1,
    output [31:0] r2,
    output [31:0] r3,
    output [31:0] r4,
    output [31:0] r5,
```

```
    output [31:0] r6,
    output [31:0] r7
);
```

- 补充功能描述： 当取指结果为IRMOV(即指令代码为 1 ，功能代码为 0)时，通过寄存器文件提供的dstM和valM接口，将立即数ValC写入到寄存器ID为rB的寄存器中。(寄存器文件提供的dstE和valE接口将在任务 4 中使用，本任务中可以悬空)

## 仿真要求：

编写testbench以完成仿真，时钟周期为50MHz，在前 8 个时钟周期令working信号为 0 ，使处理器不工作，并依次向指令存储器地址为0-7的存储单元中写入 8 条IRMOV指令，分别修改寄存器%r0-%r7的值，使其等于 128 ， 129 ， 130 ， 131 ， 132 ， 133 ， 134 ， 135 。然后令working信号为 1 ，使处理器工作，完成这 8 条指令的执行，以修改每个寄存器的数据，最终输出每个时钟周期这 8 个寄存器的数据。

# 任务 3 ： 实现译码操作

## 具体要求：

1. 在任务 2 完成的基础上，修改regfile.v文件以及processor.v文件。
2. 对于regfile.v，
   - 端口描述：

```
module regfile(
    input [3:0] dstE,
    input [31:0] valE,
    input [3:0] dstM,
    input [31:0] valM,
    input [3:0] rA,
    output [31:0] valA,
    input [3:0] rB,
    output [31:0] valB,
    input reset,
    input clock,
    output [31:0] r0,
    output [31:0] r1,
    output [31:0] r2,
    output [31:0] r3,
    output [31:0] r4,
    output [31:0] r5,
    output [31:0] r6,
    output [31:0] r7
);
```

   - 功能描述：采用同步复位，当reset有效，将寄存器r0-r7的数据清零；每个时钟周期，将数据valE写入寄存器ID为dstE的寄存器；每个时钟周期，将数据valM写入寄存器ID为dstM的寄存器；每个时钟周期，将寄存器ID为srcA的寄存器数据输出到valA；每个时钟周期，将寄存器ID为srcB的寄存器数据输出到valB； 每个时钟周期输出寄存器r0-r7的数据。
3. 对于processor.v，
   - 端口描述：

```
module processor(
    input clock,
    input [8:0] addr,
    input wr,
    input [31:0] wdata,
    input working,
    output [31:0] valA,
    output [31:0] valB
);
```

- 补充功能描述： 无论取指结果如何，通过将寄存器ID为rA的寄存器数据输出到总线valA上，将寄存器ID为rB的寄存器数据输出到总线valB上。(寄存器文件提供的dstE和valE接口将在任务 4 中使用，本任务中可以悬空)

## 仿真要求：

编写 testbench 以完成仿真，时钟周期为 50MHz，在前 12 个时钟周期令working信号为 0 ，使处理器不工作，并依次向指令存储器地址为0-7的存储单元中写入 8 条IRMOV指令， 分别修改寄存器%r0-%r7的值，使其等于 128 ， 129 ， 130 ， 131 ， 132 ， 133 ， 134 ， 135 ，再依次向指令存储器地址为8-11的存储单元中写入(ADD %r0， %r1)， (SUB %r2， %r3)， (AND %r4， %r5)， (XOR %r6， %r7)这 4 条指令。然后令working信号为 1 ，使处理器工作，完成这 12 条指令的取指和译码：先修改 8 个寄存器的数据，再完成四条ALU指令的取指和译码，最终输出每个时钟周期寄存器rA和 rB 所对应的数据(valA和valB)。

# 任务 4： 实现执行操作以及其他指令的操作

## 具体要求：

1. 在任务 3 完成的基础上，增加一个算数逻辑运算 alu.v 文件，并修改processor.v 文件，因此你的设计中应该至少包括以下文件：processor.v，ram.v，regfile.v，alu.v。
2. 对于alu.v，
   - 端口描述：

```
module alu(
    input [31:0] aluA,
    input [31:0] aluB,
    input [3:0] alufun,
    output [31:0] valE
);
```

   - 功能描述：根据功能代码alufun的值，对操作数aluA和aluB做相应运算，并将结果通过valE输出。alufun为 0 ，做加法，为 1 做减法(A-B)，为 2 做按位与运算，为 3 做按位异或运算。
3. 对于processor.v，
   - 端口描述：

```
module processor(
    input clock,
    input [8:0] addr,
    input wr,
    input [31:0] wdata,
    input working,
    output [31:0] valE
);
```

- 补充功能描述：当取指结果为ADD， SUB， AND， XOR时， 通过寄存器文件提供的dstE和valE接口，将算数逻辑运算的结果valE写入到寄存器ID为rA的寄存器中。

## 仿真要求：

编写 testbench 以完成仿真，时钟周期为 50MHz，在前 12 个时钟周期令working信号为 0 ，使处理器不工作，并依次向指令存储器地址为0-7的存储单元中写入 8 条IRMOV指令，分别修改寄存器%r0-%r7的值，使其等于 128 ， 129 ，130 ， 131 ， 132 ， 133 ， 134 ， 135 ，再依次向指令存储器地址为8-11的存储单元中写入(ADD %r0， %r1)， (SUB %r2， %r3)， (AND %r4， %r5)， (XOR %r6， %r7)这 4 条指令。然后令working信号为 1 ，使处理器工作，完成这 12 条指令的执行，先修改 8 个寄存器的数据，再执行ALU操作，最终输出每个时钟周期ALU得到的结果valE的值。

# 任务 5 ： 处理器Z的最终实现

## 具体要求：

1. 在任务 1 ， 2 ， 3 ， 4 全部完成的基础上，修改processor.v文件。
2. 对于processor.v，
   - 端口描述：

```
module processor(
    input clock,
    input [8:0] addr,
    input wr,
    input [31:0] wdata,
    input working,
    input [3:0] rID,
    output [31:0] rdata
);
```

- 补充功能描述：使用四级流水线，例化各个模块，第一级在每个时钟周期取指并解析，第二级在每个时钟周期译码得到操作数rA和rB，第三级在每个时钟周期执行算数逻辑运算得到ALU的运算结果valE，第四级在每个时钟周期将ALU的运算结果valE写回寄存器文件；将立即数valC写回寄存器文件。

## 仿真要求：

编写 testbench 以完成仿真，时钟周期为 50MHz，在前 20 个时钟周期令working信号为 0 ，使处理器不工作，并依次向指令存储器地址为0-7的存储单元中写入 8 条IRMOV指令，分别修改寄存器%r0-%r7的值，使其等于 128 ， 129 ，130 ， 131 ， 132 ， 133 ， 134 ， 135 ，再依次向指令存储器地址为8-19的存储单元中写入(ADD %r0， %r1)， (SUB %r2， %r3)， (AND %r4， %r5)， (XOR %r6， %r7)， (SUB %r5， %r4)， (AND %r7， %r6)， (ADD %r3， %r2)， (XOR %r1， %r0)，(ADD %r3， %r5)，

(SUB %r2， %r4)， (XOR %r0， %r6)， (AND %r1， %r7)这 12 条指令。然后令working信号为 1 ，使处理器工作，完成这 20 条指令的执行，先修改 8 个寄存器的数据，再执行 12 次ALU操作。当处理器完成所有指令的执行之后，令working信号为 0 ， 最后 8 个时钟周期通过rID依次读出 8 个寄存器的值。

# 进阶任务1：增加条件码、HALT指令、NOP指令

## 具体要求：

1. 在任务4完成的基础上，修改alu.v文件，和processor.v文件。
2. 增加条件码cc、HALT指令、NOP指令。指令代码和功能代码请自行定义，不做强制要求。
3. 对于alu.v，
   - 端口描述：

```
module alu(
    input [31:0] aluA,
    input [31:0] aluB,
    input [3:0] alufun,
    output [31:0] valE,
    output [2:0] cc
);
```

   - 功能描述：根据功能代码alufun的值，对操作数aluA和aluB做相应运算，并将结果通过valE输出。alufun为0，做加法，为1做减法(A-B)，为2做按位与运算，为3做按位异或运算；对于cc，要求cc[2] = zf, cc[1] = sf, cc[0] = of。
4. 对于processor.v，
   - 端口描述：

```
module processor(
    input clock,
    input [8:0] addr,
    input wr,
    input [31:0] wdata,
    input working,
    output [31:0] valE,
    output [2:0] cc
);
```

   - 补充功能描述：当取指结果为ADD, SUB, AND, XOR时，通过寄存器文件提供的dstE和valE接口，将算数逻辑运算的结果valE写入到寄存器ID为rA的寄存器中。当取指结果为HALT时，暂停处理器的运行；当取指结果为NOP时，使处理器执行一个时钟周期的空闲操作。

## 仿真要求：

编写testbench以完成仿真，时钟周期为50MHz，在前12个时钟周期令working信号为0，使处理器不工作，并依次向指令存储器地址为0-7的存储单元中写入8条IRMOV指令，分别修改寄存器%r0-%r7的值，使其等于128，129，130，131，132，133，134，135，再依次向指令存储器地址为8-17的存储单元中写入(ADD %r0, %r1), (HALT), (HALT), (SUB %r2, %r3), (NOP), (NOP), (AND %r4, %r5) , (HALT), (NOP), (XOR %r6, %r7)这10条指令。然后令working信号为1，使处理器工作，完成这12条指令的执

行，先修改8个寄存器的数据，再执行ALU操作，HALT操作，NOP操作，最终输出每个时钟周期ALU得到的结果valE的值和cc的值。

# 进阶任务2：实现处理器的转发或暂停操作

## 具体要求：

在发生数据冒险时，需要能够正确处理数据冒险。

## 仿真要求：

编写 testbench 以完成仿真，时钟周期为 50MHz，在前 20 个时钟周期令working信号为 0 ，使处理器不工作，并依次向指令存储器地址为0-7的存储单元中写入 8 条IRMOV指令，分别修改寄存器%r0-%r7的值，使其等于 128 ， 129 ， 130 ， 131 ， 132 ， 133 ， 134 ， 135 ，再依次向指令存储器地址为8-19的存储单元中写入(ADD %r1， %r0)， (SUB %r2， %r1)， (AND %r3， %r2)， (XOR %r4， %r3)，(SUB %r4， %r3)， (AND %r4， %r3)， (ADD %r4， %r3)， (XOR %r4， %r3)，(ADD %r3， %r4)，(SUB %r4， %r3)， (XOR %r3， %r4)， (AND %r4， %r3)这 12 条指令。然后令working信号为 1 ，使处理器工作，完成这 20 条指令的执行，先修改 8 个寄存器的数据，再执行 12 次ALU操作。当处理器完成所有指令的执行之后，令working信号为 0 ， 最后 8 个时钟周期通过rID依次读出 8 个寄存器的值。

# 进阶任务3：增加RRMOV指令、CMOVXX指令

## 具体要求：

1. 在任务1，2，3，4，5和扩展任务1完成的基础上，修改processor.v文件。
2. 增加RRMOV、CMOVLE、CMOVL、CMOVE、CMOVNE、CMOVGE、CMOVG七条指令。指令代码和功能代码请自行定义，不做强制要求。
3. 对于processor.v，
    - 端口描述：

```
module processor(
    input clock,
    input [8:0] addr,
    input wr,
    input [31:0] wdata,
    input working,
    output [31:0] r0,
    output [31:0] r1,
    output [31:0] r2,
    output [31:0] r3,
    output [31:0] r4,
    output [31:0] r5,
    output [31:0] r6,
    output [31:0] r7
);
```

- 补充功能描述：当取指结果为RRMOV或CMOVXX时，执行相应的指令功能。具体指令功能以及功能实现请参考书上第四章，由于是扩展任务，这里不再提示。（提示：需要考虑数据冒险，报告中应明确指出数据冒险的处理方式，不考虑数据冒险的话会酌情扣分）

# 仿真要求：

编写testbench完成以下15*6=90条指令的仿真，最终输出每个时钟周期寄存器r0-r7的值。

IRMOV $128, %r0

IRMOV $129, %r1

IRMOV $130, %r2

IRMOV $131, %r3

IRMOV $132, %r4

IRMOV $133, %r5

IRMOV $134, %r6

IRMOV $135, %r7

RRMOV %r0, %r1

SUB %r3, %r2

CMOVLE %r4, %r5

SUB %r0, %r1

CMOVLE %r4, %r5

SUB %r3, %r2

CMOVLE %r6, %r7

IRMOV $136, %r0

IRMOV $137, %r1

IRMOV $138, %r2

IRMOV $139, %r3

IRMOV $140, %r4

IRMOV $141, %r5

IRMOV $142, %r6

IRMOV $143, %r7

RRMOV %r0, %r1

SUB %r3, %r2

CMOVL %r4, %r5

SUB %r0, %r1

CMOVL %r4, %r5

SUB %r3, %r2

```
CMOVL %r6, %r7

IRMOV $144, %r0
IRMOV $145, %r1
IRMOV $146, %r2
IRMOV $147, %r3
IRMOV $148, %r4
IRMOV $149, %r5
IRMOV $150, %r6
IRMOV $151, %r7
RRMOV %r0, %r1
SUB %r3, %r2
CMOVE %r4, %r5
SUB %r0, %r1
CMOVE %r4, %r5
SUB %r3, %r2
CMOVE %r6, %r7

IRMOV $144, %r0
IRMOV $145, %r1
IRMOV $146, %r2
IRMOV $147, %r3
IRMOV $148, %r4
IRMOV $149, %r5
IRMOV $150, %r6
IRMOV $151, %r7
RRMOV %r0, %r1
SUB %r3, %r2
CMOVNE %r4, %r5
SUB %r0, %r1
CMOVNE %r4, %r5
SUB %r3, %r2
CMOVNE %r6, %r7
```

IRMOV $144, %r0

IRMOV $145, %r1

IRMOV $146, %r2

IRMOV $147, %r3

IRMOV $148, %r4

IRMOV $149, %r5

IRMOV $150, %r6

IRMOV $151, %r7

RRMOV %r0, %r1

SUB %r3, %r2

CMOVGE %r4, %r5

SUB %r0, %r1

CMOVGE %r4, %r5

SUB %r3, %r2

CMOVGE %r6, %r7

IRMOV $144, %r0

IRMOV $145, %r1

IRMOV $146, %r2

IRMOV $147, %r3

IRMOV $148, %r4

IRMOV $149, %r5

IRMOV $150, %r6

IRMOV $151, %r7

RRMOV %r0, %r1

SUB %r3, %r2

CMOVG %r4, %r5

SUB %r0, %r1

CMOVG %r4, %r5

SUB %r3, %r2

CMOVG %r6, %r7

# 进阶任务4：增加JXX指令

# 具体要求：

1. 在任务1，2，3，4，5和扩展任务1，2完成的基础上，修改processor.v文件。
2. 增加JMP、JLE、JL、JE、JNE、JGE、JG七条指令。指令代码和功能代码请自行定义，不做强制要求。
3. 对于processor.v，
    - 端口描述：

```
module processor(
    input clock,
    input [8:0] addr,
    input wr,
    input [31:0] wdata,
    input working,
    output [31:0] PC
);
```

    - 补充功能描述：当取指结果为JXX时，执行相应的指令功能。指令格式：第一个字节表示指令类型，其余三个字节表示24位的跳转地址，注意PC指针是32位的，因此需要做补零扩展。具体指令功能以及功能实现请参考书上第四章，由于是扩展任务，这里不再提示。（提示：必须要考虑数据冒险，报告中应明确指出数据冒险的处理方式）

# 仿真要求：

编写testbench完成以下32条指令的仿真，最终输出每个时钟周期PC指针的值。

A: IRMOV $1, %r0

IRMOV $2, %r1

IRMOV $3, %r2

IRMOV $4, %r3

IRMOV $5, %r4

IRMOV $6, %r5

IRMOV $7, %r6

IRMOV $8, %r7

JMP C

B: SUB %r7, %r6

C: ADD %r7, %r6

D: ADD %r1, %r0

E: SUB %r7, %r0

F: JGE D

G: ADD %r7, %r6

H: ADD %r2, %r0

I: SUB %r7, %r0

J: JG H

K: ADD %r7, %r5

L: ADD %r3, %r0

M: SUB %r7, %r0

N: JE L

O: ADD %r7, %r5

P: ADD %r3, %r0

Q: SUB %r7, %r0

R: JNE P

S: SUB %r6, %r0

T: SUB %r6, %r7

U: JLE S

V: SUB %r5, %r0

W: SUB %r5, %r7

X: JL V