

支持int指令， 请求发到这个外设，由它决定是否响应。

可以通过in、out指令设屏蔽字，读屏蔽字端口01，写屏蔽字端口11。

只增加了一个模块：中断管理模块。PmodCLP等显示模块的接口信号定义已在CPU报告中。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **信号名** | **位数** | **方向** | **来源/去向** | **意义** |
| **intr** | 8 | I | CPU | 中断请求 |
| **newImr** | 8 | I | IO管理 | 要设置的屏蔽字 |
| **intrUpdate** | 1 | I | CPU | 有新请求 |
| **imrUpdate** | 1 | I | IO管理 | 更新屏蔽字 |
| **isrUpdate** | 1 | I | CPU | 已退出中断 |
| **entered** | 1 | I | CPU | 已进入服务程序 |
| **nextService** | 1 | O | CPU | 允许中断信号 |
| **intServicePort** | 4 | O | PmodCLP | 最后执行的中断序号 |
| **nowimr** | 8 | O | IO管理 | 当前屏蔽字 |

只增加了一个模块：中断管理模块。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **信号名** | **位数** | **方向** | **来源/去向** | **意义** |
| **intr** | 8 | I | CPU | 中断请求 |
| **newImr** | 8 | I | IO管理 | 要设置的屏蔽字 |
| **intrUpdate** | 1 | I | CPU | 有新请求 |
| **imrUpdate** | 1 | I | IO管理 | 更新屏蔽字 |
| **isrUpdate** | 1 | I | CPU | 已退出中断 |
| **entered** | 1 | I | CPU | 已进入服务程序 |
| **nextService** | 1 | O | CPU | 允许中断信号 |
| **intServicePort** | 4 | O | PmodCLP | 最后执行的中断序号 |
| **nowimr** | 8 | O | IO管理 | 当前屏蔽字 |

对CPU增加了int、push、pop、iret四个指令的支持，四个指令的指令格式已经加在30页CPU指令格式设计中，不再赘述。下面是具体设计方案：

1. 内存分块

我们将内存分成中断向量表、程序区和堆栈区四个区域。为测试文件简单起见，我们支持8个中断，中断向量表在0x0000~0x000F中，第i（0<=i<=7）个字代表第i个中断服务程序的入口地址。程序区从0x0010处开始，即我们的PC初始值是0x0010。为了方便编写测试文件，中断服务程序也在这个区域中。堆栈区从0x0100处开始，每次压栈都覆盖一个字。

为了在内存中实现堆栈，我在访存控制模块(AC)增加了一个信号top，代表当前栈顶指针。当AC信号收到inctop信号时（上升沿），就根据操作类型（push还是pop）来修改top指针。

1. CPU支持指令
2. INT n指令

INT指令主要进行：在运算阶段（EX）向访存控制模块发读中断向量表的命令，从中断向量表对应位置读到对应中断服务程序的入口地址，并且向中断管理器发中断请求，若收到中断管理器nextService信号（高电平），则在存储阶段（MEM）向访存控制模块发PC压栈的命令，并在回写阶段（WB）将PCnew置为中断服务程序的入口地址。实际上也可以在MEM阶段访存读入口地址，WB阶段将PC压栈，这样可以避免在不响应中断的情况下仍然访存（取入口地址），减少了不必要的访存。

1. PUSH Rx指令

MEM阶段修改栈顶指针并将Rx压栈。因为Rx是八位的，而一个栈帧为16位，所以我们将Rx向高位复制了一遍。

1. POP Rx指令

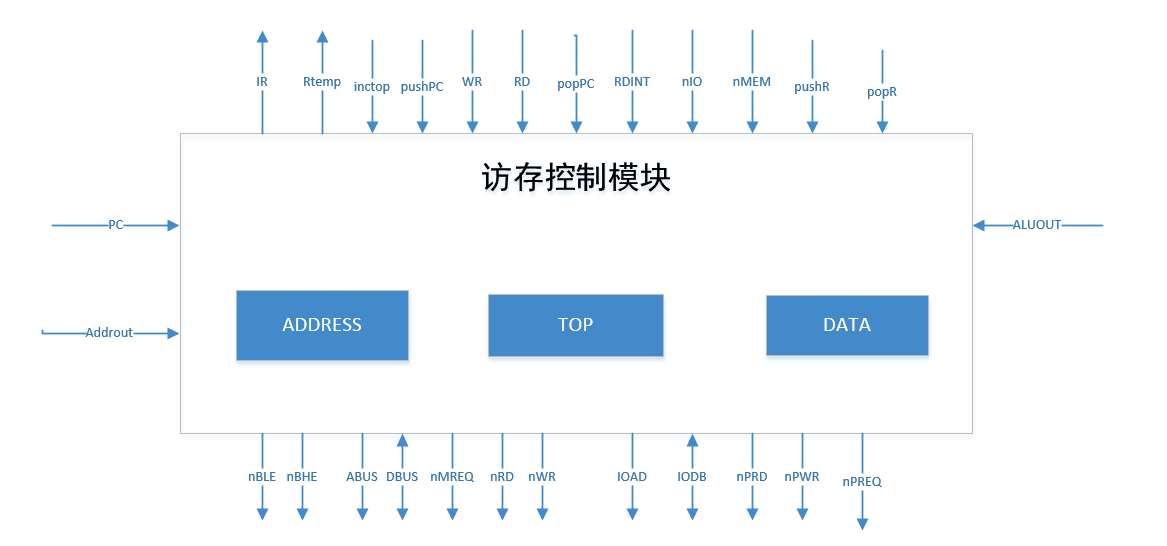
MEM阶段修改栈顶指针并读数据，WB阶段回写Rx。

1. IRET指令

EX阶段向中断管理器发退出中断的信号，MEM阶段PC弹栈，WB阶段将PCnew置为MEM阶段读出来的PCnext。

1. 各模块结构设计框图

CPU各模块结构设计并没有变化，参见实验五，访存控制模块有少许变化，修改后如下：



对CPU增加了int、push、pop、iret四个指令的支持，四个指令的指令格式已经加在30页CPU指令格式设计中，不再赘述。下面是具体设计方案：

1. 内存分块

我们将内存分成中断向量表、程序区和堆栈区四个区域。为测试文件简单起见，我们支持8个中断，中断向量表在0x0000~0x000F中，第i（0<=i<=7）个字代表第i个中断服务程序的入口地址。程序区从0x0010处开始，即我们的PC初始值是0x0010。为了方便编写测试文件，中断服务程序也在这个区域中。堆栈区从0x0100处开始，每次压栈都覆盖一个字。

为了在内存中实现堆栈，我在访存控制模块(AC)增加了一个信号top，代表当前栈顶指针。当AC信号收到inctop信号时（上升沿），就根据操作类型（push还是pop）来修改top指针。

1. CPU支持指令
2. INT n指令

INT指令主要进行：在运算阶段（EX）向访存控制模块发读中断向量表的命令，从中断向量表对应位置读到对应中断服务程序的入口地址，并且向中断管理器发中断请求，若收到中断管理器nextService信号（高电平），则在存储阶段（MEM）向访存控制模块发PC压栈的命令，并在回写阶段（WB）将PCnew置为中断服务程序的入口地址。实际上也可以在MEM阶段访存读入口地址，WB阶段将PC压栈，这样可以避免在不响应中断的情况下仍然访存（取入口地址），减少了不必要的访存。

1. PUSH Rx指令

MEM阶段修改栈顶指针并将Rx压栈。因为Rx是八位的，而一个栈帧为16位，所以我们将Rx向高位复制了一遍。

1. POP Rx指令

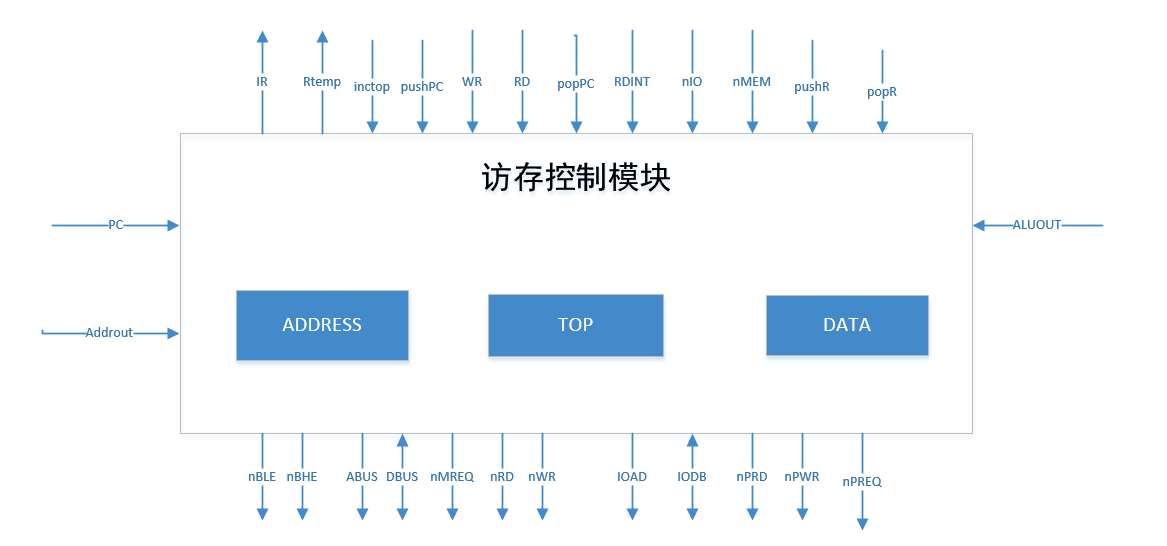
MEM阶段修改栈顶指针并读数据，WB阶段回写Rx。

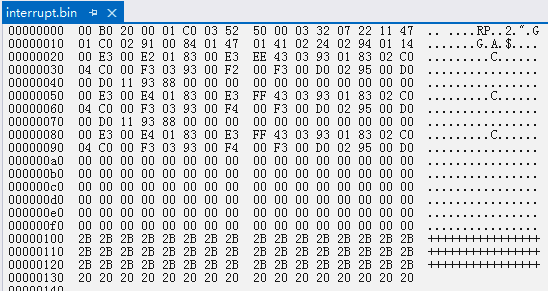
1. IRET指令

EX阶段向中断管理器发退出中断的信号，MEM阶段PC弹栈，WB阶段将PCnew置为MEM阶段读出来的PCnext。

1. 各模块结构设计框图

CPU各模块结构设计并没有变化，参见实验五，访存控制模块有少许变化，修改后如下：





测试程序中只用到了两个中断服务程序，分别是一号中断和四号中断。一号中断的服务程序入口地址保存在内存0x0002处，可以看到是0x0020，图中第三行和第四行就是一号中断的服务程序。四号中断的入口地址保存在内存0x0008中，可见是0x0050，图中第六行和第七行就是其服务程序。

测试指令序列如下。

地址0x0010:

INT 1

OUT R1, 2

IN R4, 0

MVI R7, 01

地址0x0020（一号中断）:

PUSH R3

PUSH R2

IN R3, 1 //读屏蔽字

PUSH R3 //保存原屏蔽字

MVI R3, EE

OUT R3, 3 //置屏蔽字

IN R3, 1 //读进来看看对不对

INT 2 //被屏蔽，不会进入

INT 4 //进入

POP R3

OUT R3, 3 //恢复屏蔽字

POP R2

POP R3 //恢复现场

IRET //回到主程序0x0012处

地址0x0050（四号中断）:

PUSH R3

PUSH R4

IN R3, 1 //读屏蔽字

PUSH R3 //保存原屏蔽字

MVI R3, FF

OUT R3, 3 //置屏蔽字

IN R3, 1 //读进来看看对不对

INT 2 //被屏蔽，不会进入

INT 4 //被屏蔽，不会进入

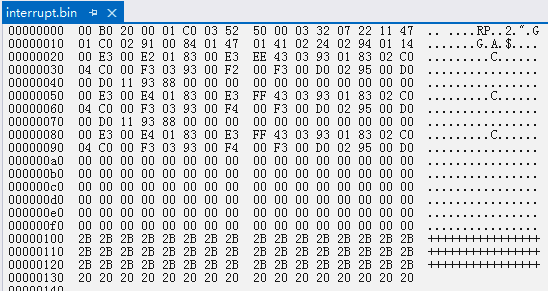
POP R3

OUT R3, 3 //恢复屏蔽字

POP R4

POP R3 //恢复现场

IRET //回到上一层中断0x0032处



测试程序中只用到了两个中断服务程序，分别是一号中断和四号中断。一号中断的服务程序入口地址保存在内存0x0002处，可以看到是0x0020，图中第三行和第四行就是一号中断的服务程序。四号中断的入口地址保存在内存0x0008中，可见是0x0050，图中第六行和第七行就是其服务程序。

测试指令序列如下。

地址0x0010:

INT 1

OUT R1, 2

IN R4, 0

MVI R7, 01

地址0x0020（一号中断）:

PUSH R3

PUSH R2

IN R3, 1 //读屏蔽字

PUSH R3 //保存原屏蔽字

MVI R3, EE

OUT R3, 3 //置屏蔽字

IN R3, 1 //读进来看看对不对

INT 2 //被屏蔽，不会进入

INT 4 //进入

POP R3

OUT R3, 3 //恢复屏蔽字

POP R2

POP R3 //恢复现场

IRET //回到主程序0x0012处

地址0x0050（四号中断）:

PUSH R3

PUSH R4

IN R3, 1 //读屏蔽字

PUSH R3 //保存原屏蔽字

MVI R3, FF

OUT R3, 3 //置屏蔽字

IN R3, 1 //读进来看看对不对

INT 2 //被屏蔽，不会进入

INT 4 //被屏蔽，不会进入

POP R3

OUT R3, 3 //恢复屏蔽字

POP R4

POP R3 //恢复现场

IRET //回到上一层中断0x0032处

经测试，程序依次正常地进入了一号中断和四号中断，读屏蔽字、置屏蔽字被屏蔽的中断都没有进入，保护现场的push和pop操作都正常，回到断点处继续执行也正常，显示器也正常显示了进入1->进入4->回到1的过程。

1. 堆栈操作栈顶指针没有正常变化。

我们一开始使用pushPC、popPC、pushr、popr信号对访存进行控制，同时用它们的下降沿对栈顶指针进行更新。但是下载到板子上运行时，事情并不像我们仿真的那样，栈顶指针根本没有发生任何变化。经过思考，我们增加了一个inctop信号，和栈操作信号同时发出，在这个信号的上升沿，根据收到的栈操作类型，让访存模块对栈顶指针进行相应的修改。最后我们果然成功了。

1. POP指令得到的数据不对。

在实际测试时，我们发现push指令压栈后内存中的数据是正确的，但是用pop指令取出来回写到寄存器时，寄存器的值会变成全0。在很长一段时间内，我们百思不得其解。最后，经过仔细核查代码，发现是因为数据从访存模块的DATA到存储模块的Rtemp再到回写模块的Rdata实际上经过了两个模块，而在访存模块到存储模块这一阶段中，我们忘记了将pop指令的操作码加到赋值语句的执行条件里面，导致Rtemp始终为0，也就无法把数据送到回写模块。这告诉我们，一定要先从头到尾设计好数据流，再编写程序，不然很容易在毫无技术含量的地方因为大意而栽了跟头。

1. 中断管理器发出的nextService信号不能保持。

我们设计的int指令是在EX阶段发intr，在MEM和WB阶段根据nextService是否为高电平决定是否将PC压栈以及更新PC为入口地址。因此，中断管理器发出的nextService信号应该至少保持到WB阶段。但是在我们写的中断管理器中，nextService信号是与intr信号同时产生、同时消失的，而intr过了EX阶段就消失了。于是我们在WB模块中添加了一个entered信号（在T3后半拍即时钟为0时发），把nextService信号改为在entered信号的上升沿才置为0。这样就成功实现了nextService信号的保持，从而成功将PC压栈并跳入中断服务程序。

经测试，程序依次正常地进入了一号中断和四号中断，读屏蔽字、置屏蔽字被屏蔽的中断都没有进入，保护现场的push和pop操作都正常，回到断点处继续执行也正常，显示器也正常显示了进入1->进入4->回到1的过程。

1. 堆栈操作栈顶指针没有正常变化。

我们一开始使用pushPC、popPC、pushr、popr信号对访存进行控制，同时用它们的下降沿对栈顶指针进行更新。但是下载到板子上运行时，事情并不像我们仿真的那样，栈顶指针根本没有发生任何变化。经过思考，我们增加了一个inctop信号，和栈操作信号同时发出，在这个信号的上升沿，根据收到的栈操作类型，让访存模块对栈顶指针进行相应的修改。最后我们果然成功了。

1. POP指令得到的数据不对。

在实际测试时，我们发现push指令压栈后内存中的数据是正确的，但是用pop指令取出来回写到寄存器时，寄存器的值会变成全0。在很长一段时间内，我们百思不得其解。最后，经过仔细核查代码，发现是因为数据从访存模块的DATA到存储模块的Rtemp再到回写模块的Rdata实际上经过了两个模块，而在访存模块到存储模块这一阶段中，我们忘记了将pop指令的操作码加到赋值语句的执行条件里面，导致Rtemp始终为0，也就无法把数据送到回写模块。这告诉我们，一定要先从头到尾设计好数据流，再编写程序，不然很容易在毫无技术含量的地方因为大意而栽了跟头。

1. 中断管理器发出的nextService信号不能保持。

我们设计的int指令是在EX阶段发intr，在MEM和WB阶段根据nextService是否为高电平决定是否将PC压栈以及更新PC为入口地址。因此，中断管理器发出的nextService信号应该至少保持到WB阶段。但是在我们写的中断管理器中，nextService信号是与intr信号同时产生、同时消失的，而intr过了EX阶段就消失了。于是我们在WB模块中添加了一个entered信号（在T3后半拍即时钟为0时发），把nextService信号改为在entered信号的上升沿才置为0。这样就成功实现了nextService信号的保持，从而成功将PC压栈并跳入中断服务程序。