# 概述

本次设计中，我们将load和store系列指令的字节写使能发生模块放到了EX段，还把store系列指令的数据移位器放到了EX段，load系列指令的移位器仍然在MEM段。（注意：为了冲刷流水段方便，我们的写使能信号都是1有效，0无效。）

# load系列指令的写使能信号

Load系列指令：

|  |  |
| --- | --- |
| LB | GPR[rt] ← sign\_extend(memory[addr]) , addr[1:0]可以为0~3 |
| LBU | GPR[rt] ← zero\_extend(memory[addr]) , addr[1:0]可以为0~3 |
| LH | GPR[rt] ← sign\_extend(memory[addr]) , addr[1]可以为0或1；addr[0]必须为0 |
| LHU | GPR[rt] ← zero\_extend(memory[addr]) , addr[1]可以为0或1；addr[0]必须为0 |
| LW | GPR[rt] ← memory[addr] , addr[1:0]必须为0 |
| LWL | 见手册 |
| LWR |  |

LB、LBU、LH、LHU这4条指令会对取出的数进行扩展，LW的取数位宽本身为4byte，所以他们对应的字节写使能都是4’b1111。而LWL和LWR指令的偏移量和addr[1:0]有关：

LWL：

|  |  |
| --- | --- |
| addr[1:0] | Reg\_byte\_write\_en |
| 0 | 1111 |
| 1 | 1110 |
| 2 | 1100 |
| 3 | 1000 |

LWR:

|  |  |
| --- | --- |
| addr[1:0] | Reg\_byte\_write\_en |
| 0 | 0001 |
| 1 | 0011 |
| 2 | 0111 |
| 3 | 1111 |

# store系列指令的写使能信号

Store系列指令：

|  |  |  |
| --- | --- | --- |
| 指令 | 行为 | 数据位宽 |
| SB | memory[addr] ← GPR[rt] , addr[1:0]可以为0~3 | 1byte |
| SH | memory[addr] ← GPR[rt] , addr[1]可以为1或0，addr[0]必须为0 | 2bytes |
| SW | memory[addr] ← GPR[rt] , addr[1:0]必须为0 | 4bytes |
| SWL | 见手册 | 不确定 |
| SWR |  |  |

store系列指令中的SB和SH指令不会对数据进行拓展。出于简化存储器设计的考虑，我希望送给存储器的**地址总是4字节对齐**的，即只有30位，然后再通过**字节写使能信号**控制每个字节是否被写入，通过移位得到要送入存储器的数据。

用一个例子来进一步说明我的想法：SB指令的addr计算结果为0x40000001，需要在0x40000001处写入一个字节，要写入的数据是0xff。按照我的思路，应该把地址(0x40000001 & 0xfffffffc)送给存储器，将字节写使能信号4’b0100送给存储器（大端方式），将数据0x0000ff00送给存储器。

SW：1111

SB：

|  |  |
| --- | --- |
| addr[1:0] | mem\_byte\_write\_en |
| 0 | 1000 |
| 1 | 0100 |
| 2 | 0010 |
| 3 | 0001 |

SH：

|  |  |
| --- | --- |
| addr[1] | mem\_byte\_write\_en |
| 0 | 1100 |
| 1 | 0011 |

SWL：

|  |  |
| --- | --- |
| addr[1:0] | mem\_byte\_write\_en |
| 0 | 1111 |
| 1 | 0111 |
| 2 | 0011 |
| 3 | 0001 |

SWR：

|  |  |
| --- | --- |
| addr[1:0] | mem\_byte\_write\_en |
| 0 | 1000 |
| 1 | 1100 |
| 2 | 1110 |
| 3 | 1111 |

# STORE指令的数据移位器

SW：不需要移位

SB: out = in << addr [1:0] \* 8

SH: out = in << addr[1] \* 16

SWL: out = in >> addr [1:0]

SWR: out = in << ~addr [1:0]

注：in表示来自寄存器的数据，out表示输出的数据

# LOAD指令的数据移位器

LW：不需要移位

LB: in = in >> addr [1:0] \* 8; out = in [7]24 + in [7:0]

LBU: in = in >> addr [1:0] \* 8; out = 024 + in [7:0]

LH: in = in >> addr [1]\* 16; out = in [15]16 + in [15:0]

LHU: in = in >> addr [1]\* 16; out = 016 + in [15:0]

LWL: out = in << addr [1:0]

LWR: out = in >> ~addr [1:0]

注：in表示来自存储器的数据，out表示输出的数据；取自存储器的数据宽度都是4bytes