# A pipeline MIPS-CPU for `Verilog` Implementation

***p5_afterclass*** | **2023/11/28**



## 1. Supported Instructions

| Instruction | add | sub | ori | lw | sw | beq | lui | jal | jr |
|---|---|---|---|---|---|---|---|---|---|
| Op | 000000 | 000000 | 001101 | 100011 | 101011 | 000100 | 001111 | 000011 | 000000 |
| Func | 100000 | 100010 | | | | | | | 001000 |

**Synchronized Reset, Overflow Not Considered**

# 2. Modules definition

*---------F---------*

## *(1) PC*

- **Ports definition**

| Port name | Direction | Width | Description |
|---|---|---|---|
| clk | input | | |
| reset | input | | |
| En | input | | *signal* |
| next_PC | input | [31:0] | NPC.next_PC -> |
| PC | output | [31:0] | -> Adder |
| InstrAddr | output | [11:0] | -> Im |

- **Behavioral Description**

```
if (En == 1'b1) then
  PC <= next_PC
else                    # when stall
  PC <= PC
```

## *(2) IM*

- **Ports Definition**

| Port name | Direction | Type | Description |
|---|---|---|---|
| InstrAddr | input | [11:0] | `PC ->` |
| Instr | output | [31:0] | `-> IF/ID` |

- **Behavioral Description**

```
# 4096 * 32bit, [31:0] Reg [0:4095]
Instr = IM[InstrAddr]
```

*---------D---------*

## *(3) Splitter*

- **Ports Definition**

| Port name | Direction | Type | Description |
|---|---|---|---|
| Instr | input | [31:0] | `ID_Instr ->` |
| Addr26 | output | [25:0] | `Instr[25:0]` |
| Imm16 | output | [15:0] | `Instr[15:0]` |
| func | output | [5:0] | `Instr[5:0]` |
| Rd | output | [4:0] | `Instr[15:11]` |
| Rt | output | [4:0] | `Instr[20:16]` |
| Rs | output | [4:0] | `Instr[25:21]` |
| Op | output | [5:0] | `Instr[31:26]` |

## Behavioral Description

None.

*(4) GRF*

## Ports Definition

| Port name | Direction | Type | Description |
|---|---|---|---|
| clk | input | | |
| reset | input | | |
| RegWrite | input | | *Signal* |
| pC | input | [31:0] | `ID_Pc4 ->` |
| WD | input | [31:0] | `WB_Result ->` |
| A1 | input | [4:0] | `Rs ->` |
| A2 | input | [4:0] | `Rt ->` |
| WA | input | [4:0] | `WB_WriteReg ->` |
| RD1 | output | [31:0] | `FWD -> CMP / NPC / ID_EX` |
| RD2 | output | [31:0] | `FWD -> CMP / ID_EX` |

## Behavioral Description

```
RD1 <= GRF[A1]
RD2 <= GRF[A2]

if RegWrite == 1 then
    GRF[WA] <= WD
```

*(5) CMP*

- ## Ports Definition

| Port name | Direction | Type | Description |
|---|---|---|---|
| srcA | input | [31:0] | `FWD_RF_RD1 ->` |
| srcB | input | [31:0] | `FWD_RF_RD2 ->` |
| CMPop | input | [2:0] | *`Signal`* |
| CmpResult | output | | `-> NPC` |

- ## Behavioral Description

```
if CMPop == 3'b000
  CmpResult = (srcA == srcB)
```

# *(6) NPC*

- ## Ports Definition

| Port name | Direction | Type | Description |
|---|---|---|---|
| Pc4 | input | [31:0] | `ID_Pc4 ->` |
| Pc4_F | input | [31:0] | `F_Pc4 ->` |
| ExtResult | input | [31:0] | `Extender ->` |
| RF_RD1 | input | [31:0] | `GRF ->` |
| Addr26 | input | [25:0] | `Splitter ->` |
| SelPCsrc | input | [2:0] | *`Signal`* |
| CmpResult | input | | `CMP ->` |
| next_PC | output | [31:0] | `-> PC` |

- ## Behavioral Description

```
if (SelPCsrc == 3'b000)                    # no branch or jump
  next_PC <= Pc4_F
else if (SelPCsrc == 3'b001 && CmpResult) # beq
  next_PC <= Pc4 + (sign_ext_offset << 2)
else if (SelPCsrc == 3'b010)               # jal
  next_PC <= {Pc4[31:28], Addr26, {2{1'b0}}}
else if (SelPCsrc == 3'b011)               # jr
  next_PC <= RF_RD1
```

---------*E*---------

## *(7) ALU*

- **Ports Definition**

| Port name | Direction | Type | Description |
|---|---|---|---|
| srcA | input | [31:0] | `(FWD)ID_EX ->` |
| srcB | input | [31:0] | `(FWD)ID_EX ->` |
| pc4 | input | [31:0] | `EX_Pc4 ->` |
| ALUop | input | [3:0] | *`Signal`* |
| Result | output | [31:0] | `-> EX_MEM` |

- **Behavioral Description**

```
if ALUop == 4'b0000
    Result <= A + B
else if ALUop == 4'b0001
    Result <= A - B
else if ALUop == 4'b0010
    Result <= A & B
else if ALUop == 4'b0011
    Result <= A | B
else if ALUop == 4'b0100    # lui
    Result <= B << 16
else if ALUop == 4'b0101    # jal
    Result <= pc4 + 4
```

*---------M---------*

## *(6) DM*

- **Ports Definition**

| Port name | Direction | Type | Description |
|---|---|---|---|
| clk | input | | |
| reset | input | | |
| DMWriteEn | input | | *signal* |
| DMReadEn | input | | *signal* |
| Addr | input | [31:0] | EX_MEM -> |
| WD | input | [31:0] | EX_MEM -> |
| pC | input | [31:0] | PC -> |
| RD | output | [31:0] | -> MEM_WB |

- **Behavioral Description**

```
if WriteData == 1'b1
    RD <= DM[Addr[13:2]]
else if ReadData == 1'b1
    DM[Addr[13:2]] <= WD
```

*---------General---------*

## *(7) CtrlUnit*

- **Ports Definition**

| Port name | Direction | Type | Description |
|---|---|---|---|
| RegWriteEN | output | | `WB_CtrlU -> D_GRF` |
| SelExtRes | output | | `D_Ext` |
| SelALUsrc | output | | `E_MUX` |
| ALUop | output | [3:0] | `E_ALU` |
| SelRegDst | output | [2:0] | `E_MUX` |
| DMReadEN | output | | `M_DM` |
| DMWriteEN | output | | `M_DM` |
| SelPCsrc | output | [2:0] | `D_NPC` |
| SelRegWD | output | [3:0] | `WB_CtrlU -> D_GRF` |
| CMPop | output | [2:0] | `D_CMP` |

- **Decoding Method**

```
RegWriteEN = (add | sub | ori | lui | lw | jal);
......
```

## *(8) ClassifyUnit*

```
cal_r = add | sub;
cal_i = ori | lui;
load = lw;
store = sw;
branch = beq;
j_r = jr;
j_addr = jal;
```

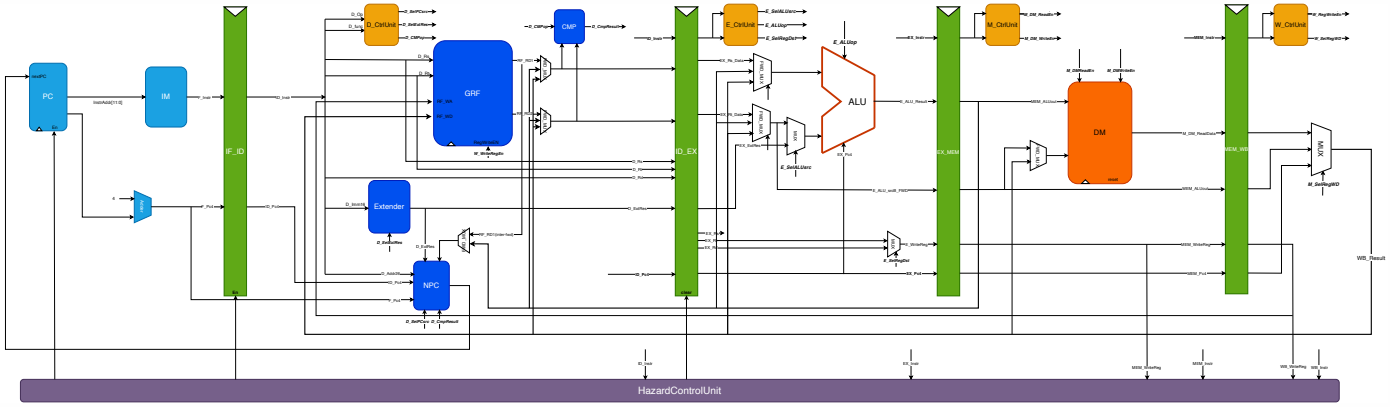
| | cal_r | cal_i | load | store | branch | j_r | j_addr |
|---|---|---|---|---|---|---|---|
| T_use | rs/rt: 1 | rt: 1 | rs: 1 | rs: 1 | rs/rt: 0 | rs: 0 | $\infty$ |
| T_new_E | rd: 1 | rt: 1 | rt: 2 | 0 | 0 | 0 | $31: 1 |

### *(9) HazardControlUnit*

- `stall_cal: (D_cal_r | D_cal_i) & E_load & WriteReg identical & not $0`
- `stall_load/stall_store: (D_load | D_store) & E_load & WriteReg iden. & not $0`
- `stall_branch/stall_j_r: T_new_E > 0 & WriteReg iden. & not $0`

**`when stall: PC.En = 1'b0 && IF_ID.En = 1'b0 && ID_EX.clear = 1'b1`**

## 3. Implementation Diagram




## 4. Questions

**Q-1: 我们使用提前分支判断的方法尽早产生结果来减少因不确定而带来的开销，但实际上这种方法并非总能提高效率，请从流水线冒险的角度思考其原因并给出一个指令序列的例子。**

A: 将分支判断提前至D流水级是为了减少由于*误判*带来的开销。事实上，因为判断的提前导致指令的T_use减小，当前序指令的数据尚未准备好时，会导致更为严重的阻塞。一种可能的序列如下:

```
lw $1, 0($0)
beq $1, $2, label
# more instructions
```

我们将误判带来的分支预测错误代价从3条降为1条，但是由于阻塞导致的暂停又持续了两周期。这实际上并没有提高效率。

**Q-2: 因为延迟槽的存在，对于`jal`等需要将指令地址写入寄存器的指令，要写回`PC + 8`，请思考为什么这样设计?**

A: 延迟槽使得跳转时的下一条指令总是被执行，这是为了提高流水线的性能而设计的。因此，由于下一条指令已经被执行，那么链接的地址自然不必是下一条指令(`PC+ 4`)，而是下下条指令(`PC + 8`)。

### Q-3: 我们要求大家所有转发数据都来源于流水寄存器而不能是功能部件（如 DM 、 ALU ），请思考为什么?

A: 如果在每一级结果计算出来以后进行转发，会使得这些流水级的延迟进一步增加。特别是考虑到E级与M级的延迟本身就相当长，那么在级内转发无疑会使得使得整个CPU的效率相当低下，这也与转发的初衷相违背。

相反，如果经过流水级寄存器再进行转发，很大程度上这个转发过程是一种“并行”的过程，也就提高了流水线的效率。

### Q-4: 我们为什么要使用 GPR 内部转发? 该如何实现?

A: 使用内部转发时，我们可以完成从W级向D级的数据转发而不需要另外附加转发单元的判断。相关代码如下:

```
wire [31:0] D_RF_RD1 = (D_Rs == WB_WriteReg && D_Rs != 5'b0 && W_RegWriteEN) ? WB_Result
: D_RF_RD1_original;   // forward inside
wire [31:0] D_RF_RD2 = (D_Rt == WB_WriteReg && D_Rt != 5'b0 && W_RegWriteEN) ? WB_Result
: D_RF_RD2_original;
```

### Q-5: 我们转发时数据的需求者和供给者可能来源于哪些位置? 共有哪些转发数据通路?

A: 数据需求者与供给者:

| | cal_r | cal_i | load | store | j_r | j_addr |
|---|---|---|---|---|---|---|
| 需求 | E: rs/rt | E: rs | E: rs | E: rs/M: rt | D: rs | |
| 供给 | M: rd | M: rt | W: rt | | | M: 31/$W$ :31 |

转发通路:

- 从M向D/E级的转发。
- 从W向D/E/M的转发。
- 内部转发。

### Q-6: 在课上测试时，我们需要你现场实现新的指令，对于这些新的指令，你可能需要在原有的数据通路上做哪些扩展或修改? 提示：你可以对指令进行分类，思考每一类指令可能修改或扩展哪些位置。

A:

- 对于计算型指令：cal_r/cal_i，在ALU中增加运算通路，在控制信号中新增对应信号，在冒险控制单元中新增相关检测信号。
- 对于load/store型指令，在DM中新增一些数据处理逻辑，在主模块中增加一些控制信号，其余同上。
- 对于branch/j_r/j_addr型指令，需要在CMP中增加新的比较信号，同时在顶层中输出对应的比较结果等。如果需要清空延迟槽等，则还需要增加对应的控制信号。

### Q-7: 确定你的译码方式，简要描述你的译码器架构，并思考该架构的优势以及不足。

A: 采取分布式译码，相关代码如下：

```
G_GeneralController D_GeneralCtrl(
  .op(D_Op),
  .func(D_func),
  .SelExtRes(D_SelExtRes),
  .SelPCsrc(D_SelPCsrc),
  .CMPop(D_CMPop),
  .RegWriteEN(), .SelALUsrc(), .ALUop(), .SelRegDst(), .DMWriteEN(), .DMReadEN(),
.SelRegWD()
);
// more controllers
```

优势：代码逻辑清晰，只在对应的流水级译出对应的信号。且不用将信号一起流水，比较方便。

劣势：在新增某些混合类型指令时，不如集中式译码方便，同时集中式译码的成本小，关键路径更短。

### Q-8: 在冒险的解决中，我们引入了 AT 法，如果你有其他的解决方案，请简述你的思路，并给出一段指令序列，简单说明你是如何做到尽力转发的。

A: 指令分类法是AT法的一种变种，或者说实际上就是把AT法具像化了一下。我使用暴力转发的方式，也就是只要检测到有寄存器相同且不为零，那么就将数据转发过来。此时数据通路的正确性由暂停机制保证，也就是暂停机制可以保证所有数据在它的最晚使用时已经产生。

另：由于使用的测试数据过多，且都辅以覆盖率分析，在此不再赘述。