A **full version** of pipeline MIPS-CPU for `Verilog` Implementation

p6_afterclass | **2023/12/07**

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

Instruction	and	or	andi	addi	slt	sltu	lh	lb
Op	000000	000000	001000	001100	000000	000000	100001	100000
Func	100100	100101			101010	101011		

Instruction	sh	sb	bne	mfhi	mflo	mult	multu	div
Op	101001	101000	000101	000000	000000	000000	000000	000000
Func				010000	010010	011000	011001	011010

Instruction	divu	mthi	mtlo
Op	000000	000000	001101
Func	011011	010001	010011

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 (Has been moved to CPU outside)

---------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**

```
(ALUop == `add) ? srcA + srcB :
(ALUop == `sub) ? srcA - srcB :
(ALUop == `and) ? srcA & srcB :
(ALUop == `or) ? srcA | srcB :
(ALUop == `lui) ? srcB << 16 :
(ALUop == `jal) ? pc4 + 32'h0000_0004 :
(ALUop == `slt) ? $signed(srcA) < $signed(srcB) :
(ALUop == `sltu) ? srcA < srcB :
32'h0000_0000;
```

(8) MultDivUnit

- **Ports Definition**

Port name	Direction	Type	Description
rs	input	[31:0]	ID_EX ->
rt	input	[31:0]	ID_EX ->
HL_Op	input	[3:0]	*Signal*
HL_out	output	[31:0]	-> EX_MEM
HL_busy	output		*Signal*

- **Behavioral Description**

```
`HL_Mult:
    {temp_hi, temp_lo} <= $signed(rs) * $signed(rt);
`HL_Multu:
    {temp_hi, temp_lo} <= rs * rt;
`HL_Div:
    temp_lo <= $signed(rs) / $signed(rt);
```

```
    temp_hi <= $signed(rs) % $signed(rt);
`HL_Divu:
    temp_lo <= rs / rt;
    temp_hi <= rs % rt;
`HL_Mthi:
    HI <= rs;
`HL_Mtlo:
    LO <= rs;
```

---------M---------

(9) DM (Has been moved to CPU outside)

(10) DM_EXT

- **Ports Definition**

Port name	Direction	Type	Description
Addr	input	[31:0]	`ALU ->`
DM_RD_raw	input	[31:0]	`Outside_DM ->`
DMReadEN	input	[2:0]	*`Signal`*
DM_RD	output	[31:0]	`-> MEM_WB`

- **Behavioral Description**

```
if (`lw)
    WriteData = DM_RD_raw;
else if (`lh)
    if (Addr[1] == 1)
        WriteData = {{16{DM_RD_raw[31]}}, DM_RD_raw[31:16]};
    else
        WriteData = {{16{DM_RD_raw[15]}}, DM_RD_raw[15:0]};
else if (`lb)
    if (Addr[1:0] == 3)
        WriteData = {{24{DM_RD_raw[31]}}, DM_RD_raw[31:24]};
    else if (Addr[1:0] == 2)
        WriteData = {{24{DM_RD_raw[23]}}, DM_RD_raw[23:16]};
    else if (Addr[1:0] == 1)
```

```
        WriteData = {{24{DM_RD_raw[15]}}, DM_RD_raw[15:8]};
    else
        WriteData = {{24{DM_RD_raw[7]}}, DM_RD_raw[7:0]};
```

---------General---------

(11) 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`
HL_Op	output	[3:0]	`E_MultDivUnit`
SelALUResult	output		`E_Mux`
SelRegDst	output	[2:0]	`E_MUX`
DMReadEN	output	[2:0]	`M_outside_DM`
DMWriteEN	output	[2:0]	`M_outside_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);
......
```

(12) ClassifyUnit

```
cal_r = add | sub | _and | _or | slt | sltu;
cal_i = andi | ori | addi | lui;
load = lw | lh | lb;
store = sw | sh | sb;
branch = beq | bne;
j_r = jr;
j_addr = jal;
md = mult | multu | div | divu;
mt = mthi | mtlo;
mf = mfhi | mflo;
```

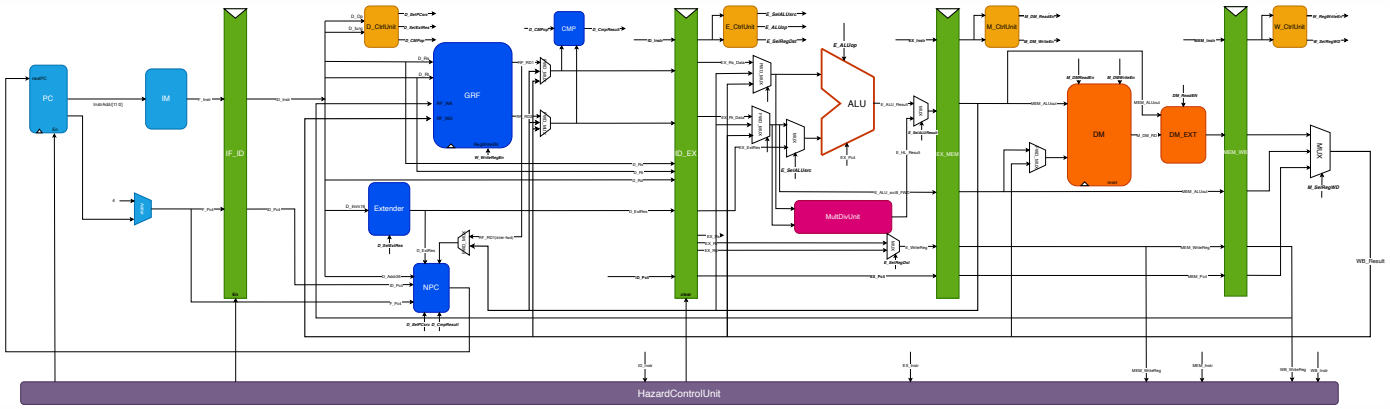
	cal_r	cal_i	load	store	branch	j_r	j_addr	md	mt	mf
T_use	rs / rt: 1	rt: 1	rs: 1	rs: 1	rs / rt: 0	rs: 0	∞	rs / rt: 1	rs / rt: 1	∞
T_new_E	rd: 1	rt: 1	rt: 2	0	0	0	$31: 1	0	0	rd: 1

(13) 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`
- `stall_md/stall_mt: (D_md | D_mt) && E_load & WriteReg iden. && not $0`
- `stall_md_busy: (D_mf | D_md | D_mt) && HL_busy`

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

3. Implementation Diagram



4. Questions

Q-1: 为什么需要有单独的乘除法部件而不是整合进 ALU？为何需要有独立的 HI、LO 寄存器？

A: 乘除法的运算延迟非常长，使用一条通路进行处理会使得CPU的延迟增大很多，不利于整体的效率。而独立的HI、LO寄存器使得并行运算成为可能，提高了CPU的效率。

Q-2: 真实的流水线 CPU 是如何使用实现乘除法的？请查阅相关资料进行简单说明。

A: 在真实的CPU中，乘法运算是通过加法器与移位器循环运算实现的，通过被乘数的每一位进行判断并运算后进行移位再进行下一位的运算，直到迭代结束。对于有符号数的运算，先进行符号的运算，再进行无符号数的运算。

除法运算是由循环移位与减法器实现的。对于有符号数的处理同乘法，由于循环进行的位运算导致乘除法的时钟延迟非常之大。

Q-3: 请结合自己的实现分析，你是如何处理 Busy 信号带来的周期阻塞的？

A: 实现逻辑：

```
当前正在执行?
    busy置1，  计数器 -= 1
当前将要执行?
    start置1，  计数器置为对应的延迟
当前即将结束运算?
    将结果赋给HI、LO寄存器
    计数器、busy信号归零

stall_busy = start | busy;
```

Q-4: 请问采用字节使能信号的方式处理写指令有什么好处？（提示：从清晰性、统一性等角度考虑）

A: 使用最少的信号可以表示所有情况，降低了控制的复杂度，同时独热码的使用也使得控制信号非常清晰，阅读时对于字节的写入控制一目了然。

Q-5: 请思考，我们在按字节读和按字节写时，实际从 DM 获得的数据和向 DM 写入的数据是否是一字节？在什么情况下我们按字节读和按字节写的效率会高于按字读和按字写呢？

A: 实际上获取的是一个字（4字节），通过后续处理得到对应的字节数据。考虑一个字符串类型的数据（char），由于每个字符都是一字节，那么我们访存时获取单字节的效率应该高于获取整个字的内存数据。

Q-6: 为了对抗复杂性你采取了哪些抽象和规范手段？这些手段在译码和处理数据冲突的时候有什么样的特点与帮助？

A: 按模块、按功能分类实现各个单元。同时，由于信号数量较多，我对每一级和每一种信号的命名都加以规范：凡当级模块的信号，使用该级名称作为前缀：`E_ALUop`，`M_DMReadEN`等；凡由流水寄存器生成的信号，全部使用该级全称作为前缀：`MEM_WriteReg`，`EX_Pc`等。

使用这样的命名方式进行整理后，所有信号的所属级、所属来源就变的非常清晰了，增强了扩展性与可读性。

Q-7: 在本实验中你遇到了哪些不同指令类型组合产生的冲突？你又是如何解决的？相应的测试样例是什么样的？

A: 很多，这里用一个我改了很久的冲突作为例子：`lui - bne / beq`，由于我之前将lui归类为单独的指令，而不属于cal_i类，导致在我后来进行了一些数据通路的修改后忘记将其再归为cal_r类，致使转发错误。

下面列举一些典型的冲突类型：

```
load ~-> store  # stall
cal_i(lui) ~-> beq / bne  # stall
load ~-> mt   # stall
cal_i ~-> j_addr  # fwd
......
```

相应的测试样例：

```
lw $1, 0($0)
sw $1, 0($1)

lui $2, 0xffff
bne $0, $3, label_1

lw $22, 0($0)
mthi $22
mtlo $22

andi $11, $10, 0xffff
jal label_2
```

Q-8: 如果你是手动构造的样例，请说明构造策略，说明你的测试程序如何保证覆盖了所有需要测试的情况；如果你是完全随机生成的测试样例，请思考完全随机的测试程序有何不足之处；如果你在生成测试样例时采用了特殊的策略，比如构造连续数据冒险序列，请你描述一下你使用的策略如何结合了随机性达到强测的效果。

A: 手动生成数据赋役自动生成数据。在手动编写测试数据时，我使用排列组合的方式进行构造，尽量使得不同类指令的不同位置的寄存器都相互用到，以达到尽量覆盖全部冲突。使用课程组提供的覆盖率分析工具，我对每次不同的类别进行分析，看两种不同类别指令之间是否能达到全覆盖。

在最后，使用了一些自动化生成数据的手段，期望能用完全随机的方式再测试是否会有没有覆盖到的冲突。