A single-style CPU documentation

P3-Afterclass **| *2023.10.26***

1. Supported Instruction

`add`, `sub`, `ori`, `lw`, `sw`, `beq`, `lui`, `nop`

OpCode:

- `add: 000000`, `func: 100000`
- `sub: 000000`, `func: 100010`
- `ori: 001101`
- `lw: 100011`
- `sw: 101011`
- `beq: 000100`
- `lui: 001111`

2. Modules Definition

- ### Program Counter

Instruction address control.

Ports	**NextPC**	**PC**	**InstrAddr**	**Clk**	**Reset**
Width	[31:0]	[31:0]	[11:0]	1	1
In	PC + 4 | PC + 4 + AddrGap			clk	reset
Out		Adder(+4)	IM_ReadAddr		
CtrlSignal	Mux: `PCsrc & ALUzero`				

- ### Instr Memory

Store Instruction.

Ports	**ReadAddr**	**Instruction**
Width	[11:0]	[31:0]
In	InstrAddr	
Out		Splitter
CtrlSignal		

- ## Instr Splitter

Divide Instruction into different parts.

Ports	**Instr[31:26]**	**Instr[25:21]**	**Instr[20:16]**	**Instr[15:11]**	**Instr[5:0]**	**Instr[15:0]**
Width	6	5	5	5	6	16
In						
Out	Op	Rs	Rt	Rd	Func	Imm16

- **Register File**

32 registers of 32bit, with `Clk` and `Reset` signals.

Ports	Addr1	Addr2	Addr3	WD	RD1	RD2	Clk	Reset	RegWrite
Width	[4:0]	[4:0]	[4:0]	[31:0]	[31:0]	[31:0]	1	1	1
In	Instr[25:21]	Instr[20:16]	Instr[15:11] | Instr[20:16]	DM | Shifter | ALU			clk	reset	*signal*
Out					ALU_srcA	ALU_srcB			
CtrlSignal			Mux: `RegDst`	Mux: `ShfToReg`, `MemToReg`					

- **ALU**

Allows 32-bit ***addition***, ***subtraction***, ***bitwise-and***, ***bitwise-or*** operations.

Use `ALUop` as control signal.

Ports	SrcA	SrcB	Zero	Result
Width	[31:0]	[31:0]	1	[31:0]
In	RD1	RD2 | Signext(Imm16)		
Out			PC jumpctrl	DM | GRF
CtrlSignal		Mux: `ALUsrc`		

ALUop	`00:Add`	`01:Sub`	`10:And`	`11:Or`

- **Data Memory**

3072 * 32bit **RAM** for storing data.

Ports	Addr	WriteData	ReadData	WriteEn	ReadEn	Clk	Reset
Width	[11:0]	[31:0]	[31:0]	1	1	1	1
In	ALU	GRF		*signal*	*signal*	clk	reset
Out			GRF				
CtrlSignal							

- **Controler**

Generate signals for datapath control.

Use the ***and or*** structure to judge the instruction and output the corresponding signal.

- **Bit Extender**

Shift 2bits left for address or 16bits left for `lui`.

Ports	0 Extend	Sign Extend	Out
Width	32	32	32
In	16 Imm	16 Imm	
Out			2-shifter | 16-shifter
CtrlSignal			Mux: `ExtRes`

3. Control Signal

	RegDst	ALUsrc	ALUop[1:0]	PCsrc	ReadData	WriteData	MemToReg	ShfToReg	RegWrite	ExtRes
ADD	1	0	00(Add)	0	x	x	1	0	1	x
SUB	1	0	01(Sub)	0	x	x	1	0	1	x
ORI	0	1	11(Or)	x	x	x	1	0	1	1
LW	0	1	00(Add)	0	1	x	0	0	1	0
SW	x	1	00(Add)	0	x	1	x	0	0	0
BEQ	x	0	01(Sub)	1	x	x	x	x	0	0
LUI	0	x	xx	0	x	x	x	1	1	x

4.Datapath

Instr	Adder		PC	IM Addr			Reg Files		ALU		DM		Bit Extender	Shift	Nadder	
	A	B			Reg1	Reg2	WriteReg	WriteData	A	B	Addr	WriteData			A	B
R	PC	4	Adder	PC	Rs	Rt	Rd	ALU	RD1	RD2						
lw	PC	4	Adder	PC	Base(Rs)		Rt	DM	RD1	signExt(16Imm)	ALU		16Imm			
sw	PC	4	Adder	PC	Base(Rs)	Rt			RD1	signExt(16Imm)	ALU	RD2	16Imm			
beq	PC	4	Adder/NAdder	PC	Rs	Rt			RD1	RD2			16Imm	left 2	PC + 4	signExt(16Imm) <<2
ori	PC	4	Adder	PC	Rs		Rt	ALU	RD1	0Ext(16Imm)						
lui	PC	4	Adder	PC			Rt	16Leftshift								
j																
General	PC	4	Adder/Nadder	PC	Rs	Rt	Rd/Rt	ALU/DM/shifter	RD1	RD2/signExt/0Ext			16Imm			
			Mux				Mux	Mux*2		Mux*2						
			`PCsrc & Zero`				`RegDst`	`MemToReg/ExtRes/ShfToReg`		`ALUsrc/ExtRes`						

5. Test Program

Both Logisim logging and MIPS programs were used for testing.

- **Test1**

```
.text
ori $zero, $zero, 0xffff# $0 test
ori $t0, $zero, 0x0     # ori test
ori $t1, $t0, 0x1234
ori $t1, $t0, 0xffff
ori $t2, $t0, 0xf4ca

lui $25, 0xfff          # lui test and 32bit test
ori $25, 0xffff
lui $24, 0x2cbe
ori $24, 0x1234

add $t1, $t0, $zero     # add test
lui $t0, 0xffff
ori $t0, $t0, 0xfff0    # t0 = 0xfffffff0
ori $t2, $zero, 0xf     # t2 = 0xf
add $t2, $t2, $t0
add $t1, $zero, $t0
ori $t2, $t2, 0x1234
ori $t3, $t3, 0x4321
add $t3, $t3, $t2

ori $t0, $zero, 0x11fa  #sub test
```

```
sub $t1, $t0, $zero
ori $t2, $t2, 0x21fa
sub $t2, $t2, $t1

sw $t2, 0($zero)        # sw test
lui $t0, 0x0
ori $t0, $t0, 0x4
sw $t2, -4($t0)
sw $t2, 0($t0)
sw $t2, 4($t0)

lw $t3, 0($t0)          # lw test
lui $t3, 0x0
lw $t3, -4($t0)
lui $t3, 0x0
lw $t3, 4($t0)

previous:
beq $t0, $zero, next    # beq test
beq $t0, $t0, next1
next:
lui $t0, 0xffff
next1:
lui $t0, 0x1234
beq $t0, $t0, previos   # jump to previous instr
```

- **Test2(Adapted from CSCORE)**

```
ori $a0, $0, 0xfffa
ori $a1, $a0, 0xac38
lui $a2, 123            # 符号位为 0
lui $a3, 0xffff         # 符号位为 1
ori $a3, $a3, 0xffff    # $a3 = -1
add $s0, $a0, $a2      # 正正
add $s1, $a0, $a3      # 正负
add $s2, $a3, $a3      # 负负
ori $t0, $0, 0x0000
sw $a0, 0($t0)
sw $a1, 4($t0)
sw $a2, 8($t0)
sw $a3, -12($t0)
sw $s0, 16($t0)
sw $s1, 20($t0)
sw $s2, 24($t0)
lw $a0, 0($t0)
lw $a1, -12($t0)
sw $a0, 28($t0)
sw $a1, 32($t0)
ori $a0, $0, 1
ori $a1, $0, 2
ori $a2, $0, 1
pre:
beq $a0, $a1, loop1     # 不相等
beq $a0, $a2, loop2     # 相等
loop1:
sw $a0, 36($t0)
loop2:
sw $a1, 40($t0)
beq $zero, $zero, pre
```

5. Questions

(1) 可以认为IM，PC储存状态；而ALU等组合电路负责状态转移。

(2) 使用ROM作为指令寄存器不合理，属于过度简化。指令集应该允许增加和修改。使用寄存器和RAM则是可以接受的，参考现代CPU的结构，可以考虑加入缓存或多级缓存来减小CPU与储存元件之间的速度差距，减小对性能的影响。

(3) 暂时没有设计更多的模块。后续会考虑为PC的处理增加新的模块，在`j`类指令引入后。

(4) `nop`指令不需要加入，因所有的寄存器默认的初始值都为0，那么`nop`指令就会自动被忽略，而无需做任何特别的判断。

(5) 强度不足。第一缺少对于边界数据的测试，比如接近32位数最大值的测试，接近16位数最大值的测试。`add`指令，`sub`指令缺少对自身加减运算的测试。`lw`指令，`sw`指令缺少对`offset`为负数时的测试，`beq`指令缺少向前跳转的测试（当然，目前缺少`j`指令，向前跳转可能会陷入死循环）。