4.1

4.1.1

`RegWrite = 0`, `MemRead = 0`, `ALUMux = 1 (imm)`, `MemWrite = 1`, `ALUop = ADD`, `RegMux = X`, `Branch = 0`

ALUMux是控制ALU输入端多路复用器的控制信号，0（Reg）选择寄存器的输出，1（Imm）从指令字中选择直接作为ALU的第二个输入。RegMux是控制寄存器 le 的数据输入处的多路复用器的控制信号，0（ALU）选择ALU的输出，1（Mem）选择存储器的输出。X表示无关紧要。

4.1.2

除分支外的所有寄存器的添加单元和写入端口。

4.1.3

未使用的输出：分支添加，寄存器的写入端口；没有不产生任何输出的单元输出。

4.2

4.2.1

指令存储器，两个寄存器读取端口，ALU将Rd和R加在一起，数据存储器，寄存器中的写入端口。

4.2.2

没有。IS指令可以使用现有模块实现。

4.2.3

没有。无需添加新的控制信号即可实现指令。只要更改Control的逻辑。

4.7

4.7.1

Sign-extend: 00000000000000000000000000010100

Jumps shift-left-2: 00011000100000000000001010000

4.7.2

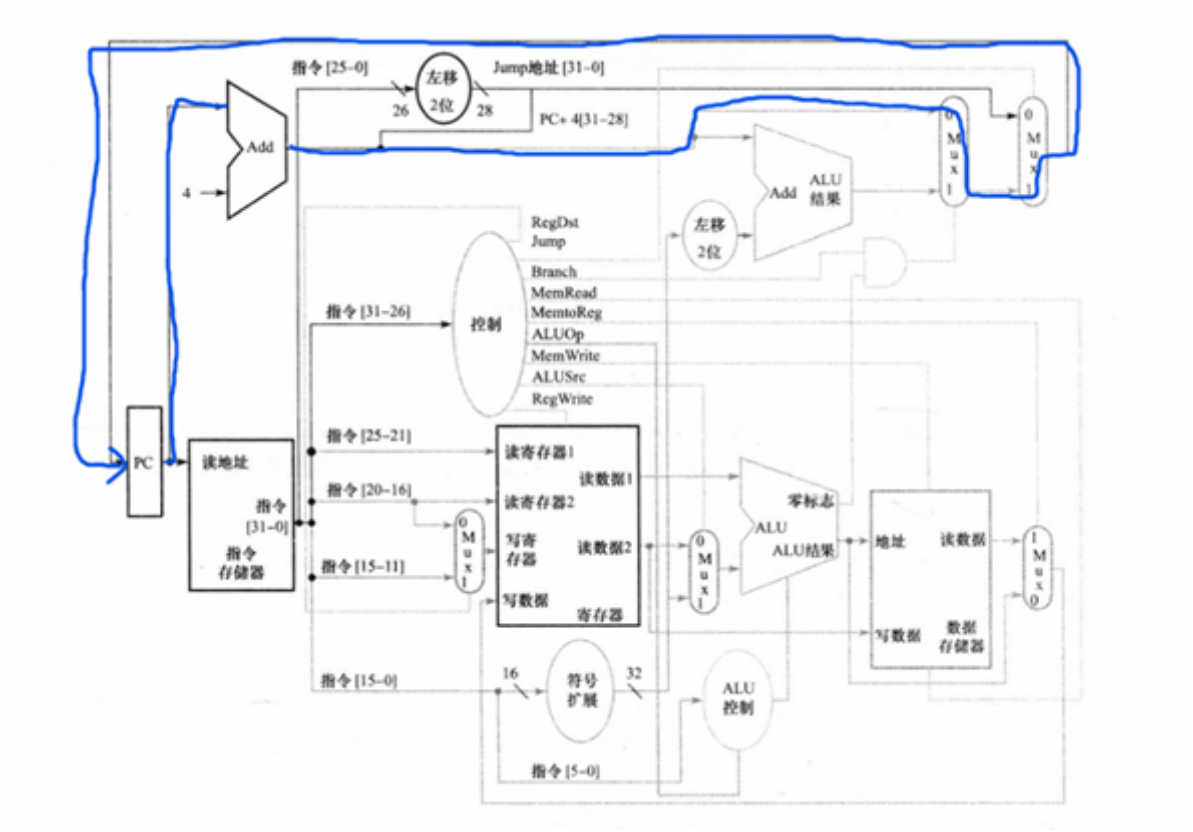
ALUOp[1:0]: 00

Instruction[5:0]: 010100

4.7.3

New PC: PC+4

Path:



4.7.4

```
WrReg Mux: 2 or 0 (RegDst is X)
ALU Mux: 20
Mem/ALU Mux: X
Branch Mux: PC+4
Jump Mux: PC+4
```

4.7.5

```
ALU: -3 and 20
Add(PC+4): PC and 4
Add(Branch): PC+4 and 20*4
```

4.7.6

Read Register 1, Read Register 2, Write Register, Write Data, RegWrite

4.9

4.9.1

I1: OR R1,R2,R3

I2: OR R2,R1,R4

I3: OR R1,R1,R2

写后读依赖 R1 from I1 to I2 and I3

写后读依赖 R2 from I2 to I3

读后写依赖 R2 from I1 to I2

读后写依赖 R1 from I2 to I3

写后写依赖 R1 from I1 to I3

4.9.2

OR R1,R2,R3

NOP(延迟 I2 以避免 I1 对 R1 造成 RAW 冒险)

NOP

OR R2,R1,R4

NOP(延迟 I3 以避免 I2 对 R2 的 RAW 冒险)

NOP

OR R1,R1,R2

4.9.3

没有产生冒险

4.9.4

无旁路: (7*4)*180ps=1980ps

有旁路: 7*240ps=1680ps

加速比: 1.18

4.9.5

无冒险

4.9.6

无旁路: (7*4)*180ps=1980ps

有旁路: 7*210ps = 1470ps

加速比: 1.35