处理器设计

5. 集成控制逻辑

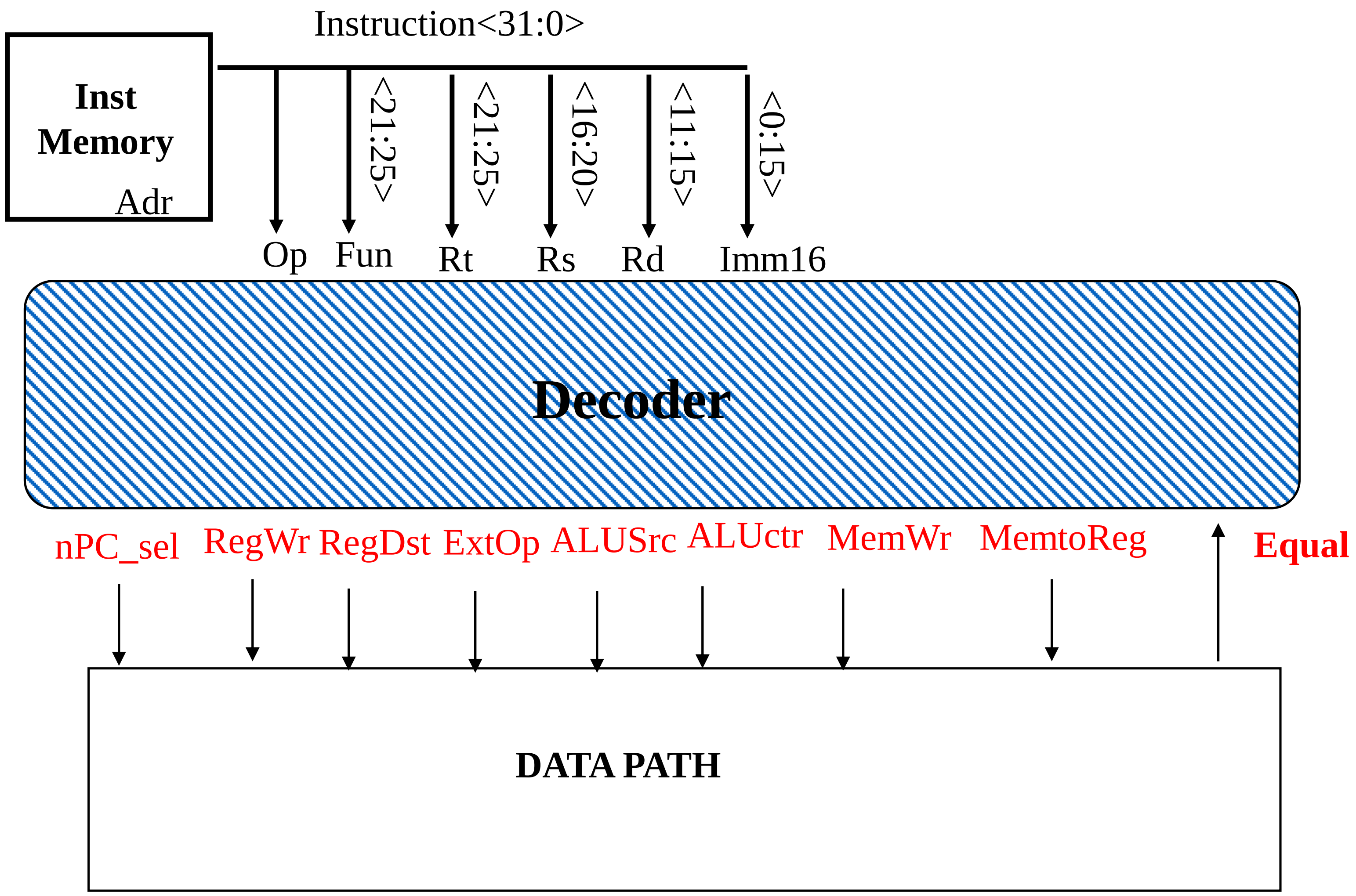
主讲人: 邓倩妮
上海交通大学

大部分内容来自于：
Computer Organization and Design, 4th Edition, Patterson & Hennessy,

设计处理器的五个步骤

1. 分析指令系统， 得出对数据通路的需求
   - The meaning of each instruction is given by the register transfers
   - 例如：ADDU指令的数据通路需求：R[rd] <– R[rs] + R[rt];
2. 选择数据通路上合适的组件
   - 例如：加法器？ 算术逻辑运算单元？寄存器堆？
3. 连接组件构成数据通路
4. 分析每一条指令的实现，以确定控制信号，
   - 不同的控制信号影响寄存器之间的数据传送
5. 集成控制信号，完成控制逻辑

Step 5: Assemble Control logic



回顾：控制信号总结

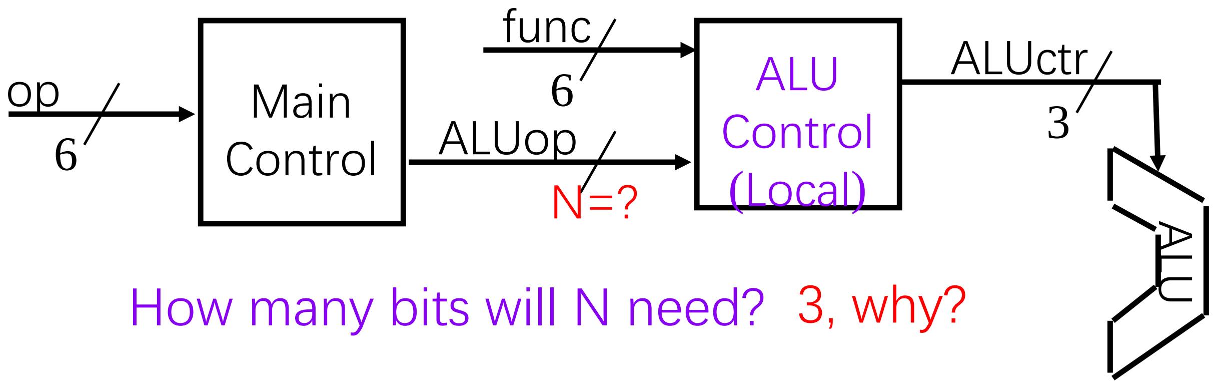
inst	Register Transfer	
ADD	R[rd] <– R[rs] + R[rt];	PC <– PC + 4
	ALUsrc = RegB, ALUctr = “add”, RegDst = rd, RegWr, nPC_sel = “+4”	
SUB	R[rd] <– R[rs] – R[rt];	PC <– PC + 4
	ALUsrc = RegB, ALUctr = “sub”, RegDst = rd, RegWr, nPC_sel = “+4”	
ORi	R[rt] <– R[rs] + zero_ext(Imm16);	PC <– PC + 4
	ALUsrc = Im, Extop = “Z”, ALUctr = “or”, RegDst = rt, RegWr, nPC_sel = “+4”	
LOAD	R[rt] <– MEM[R[rs] + sign_ext(Imm16)];	PC <– PC + 4
	ALUsrc = Im, Extop = “Sn”, ALUctr = “add”, MemtoReg, RegDst = rt, RegWr, nPC_sel = “+4”	
STORE	MEM[R[rs] + sign_ext(Imm16)] <– R[rs];	PC <– PC + 4
	ALUsrc = Im, Extop = “Sn”, ALUctr = “add”, MemWr, nPC_sel = “+4”	
BEQ	if (R[rs] == R[rt]) then PC <– PC + sign_ext(Imm16)] || 00 else PC <– PC + 4	
	nPC_sel = “Br”, ALUctr = “sub”	

控制信号总结

func	10 0000	10 0010	We Don't Care :-)				
op	00 0000	00 0000	00 1101	10 0011	10 1011	00 0100	00 0010
	add	sub	ori	lw	sw	beq	jump
RegDst	1	1	0	0	x	x	x
ALUSrc	0	0	1	1	1	0	x
MemtoReg	0	0	0	1	x	x	x
RegWrite	1	1	1	1	0	0	0
MemWrite	0	0	0	0	1	0	0
Branch	0	0	0	0	0	1	0
Jump	0	0	0	0	0	0	1
ExtOp	x	x	0	1	1	x	x
ALUctr<2:0>	Add	Subtr	Or	Add	Add	Subtr	xxx

两层次译码: Main Control and ALU Control

op	00 0000	00 1101	10 0011	10 1011	00 0100	00 0010
	R-type	ori	lw	sw	beq	jump
RegDst	1	0	0	x	x	x
ALUSrc	0	1	1	1	0	x
MemtoReg	0	0	1	x	x	x
RegWrite	1	1	1	0	0	0
MemWrite	0	0	0	1	0	0
Branch	0	0	0	0	1	0
Jump	0	0	0	0	0	1
ExtOp	x	0	1	1	x	x
ALUctr	Add/Subtr	Or	Add	Add	Subtr	xxx



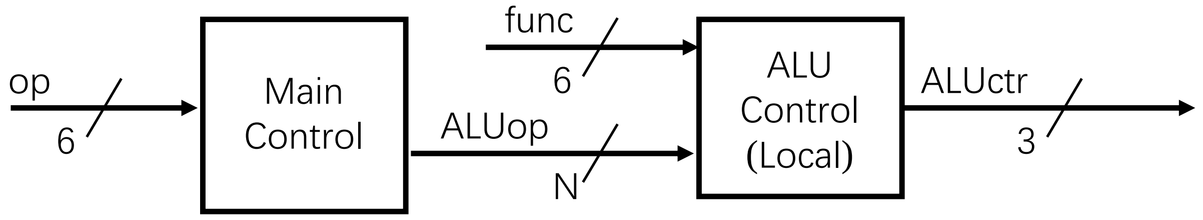
How many bits will N need? 3, why?

ALUctr 由ALUop 和 func,

其他控制信号由 op决定。

R、I-ori、I-lw/sw、I-beq、J

指令中“func” 域的译码



对 ALUop 编码

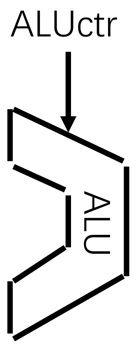
	R-type	ori	lw	sw	beq	jump
ALUop (Symbolic)	“R-type”	Or	Add	Add	Subtr	xxx
ALUop<2:0>	1 xx	0 10	0 00	0 00	0x1	xxx

R-型：1xx，使用1位表示R 型 指令

ALUop 只用两位 2 bits 可以吗？
Yes！既然 jump 是 X，那么使用2 bits足够了：
R:11, I-ori:10, I-beq:01, I-lw/sw:00, J-xx

	31	26	21	16	11	6	0
R-type	000000	rs	rt	rd	shamt	func	

func<5:0>	Instruction Operation	ALUctr<2:0>	ALU Operation
10 0000	add	000	Add
10 0010	subtract	001	Subtract
10 0100	and	100	And
10 0101	or	101	Or
10 1010	set-on-less-than	010	Subtract



ALUctr 的真值表

R-type Instructions determined by funct

Non-R-type Instructions determined by ALUop

ALUop	R-type	ori	lw	sw	beq
(Symbolic)	“R-type”	Or	Add	Add	Subtr
ALUop<2:0>	1 00	0 10	0 00	0 00	0 x1

funct<3:0>	Instruction Op.
0000	add
0010	subtract
0100	and
0101	or
1010	set-on-less-than

ALUop			func				ALU	ALUctr		
bit2	bit1	bit0	bit<3>	bit<2>	bit<1>	bit<0>	Operation	bit<2>	bit<1>	bit<0>
0	0	0	x	x	x	x	Add	0	0	0
0	x	1	x	x	x	x	Subtract	0	0	1
0	1	0	x	x	x	x	Or	1	1	0
1	x	x	0	0	0	0	Add	0	0	0
1	x	x	0	0	1	0	Subtract	0	0	1
1	x	x	0	1	0	0	And	0	1	0
1	x	x	0	1	0	1	Or	1	1	0
1	x	x	1	0	1	0	Subtract	0	0	1

ALUctr<0> 的逻辑表达式

- ALUctr<0> = !ALUop<2> & ALUop<0> +
  ALUop<2> & !func<2> & func<1> & !func<0>

ALUctr[0]=1 所在的行

ALUop			func				
bit<2>	bit<1>	bit<0>	bit<3>	bit<2>	bit<1>	bit<0>	ALUctr<0>
0	x	1	x	x	x	x	1
1	x	x	0	0	1	0	1
1	x	x	1	0	1	0	1

This makes func<3> a don’t care

ALUctr<1> 的逻辑表达式

ALUctr[1]=1 所在的行

ALUop bit<2>	ALUop bit<1>	ALUop bit<0>	func bit<3>	func bit<2>	func bit<1>	func bit<0>	ALUctr<1>
0	1	0	x	x	x	x	1
1	x	x	0	1	0	0	1
1	x	x	0	1	0	1	1

- ALUctr<1> = !ALUop<2> & ALUop<1> & ! ALUop<0> +
  ALUop<2> & !func<3> & func<2> & !func<1>

ALUctr<2> 的逻辑表达式

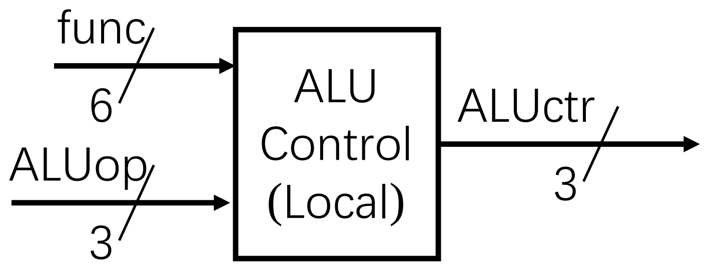
ALUctr[2]=1 所在的行

ALUop			func				
bit<2>	bit<1>	bit<0>	bit<3>	bit<2>	bit<1>	bit<0>	ALUctr<2>
0	1	0	x	x	x	x	1
1	x	x	0	1	0	1	1

- ALUctr<2> = !ALUop<2> & ALUop<1> & !ALUop<0>

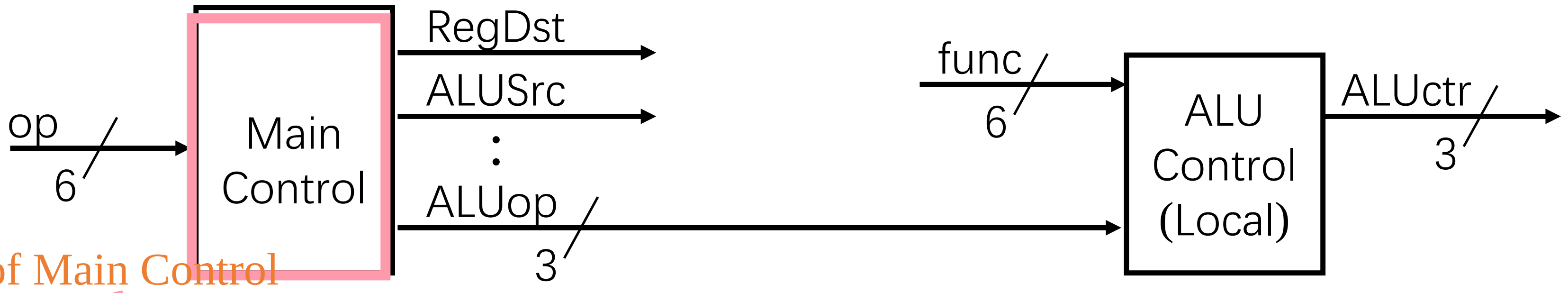
  + ALUop<2> & !func<3> & func<2> & !func<1> & func<0>

ALU Control 控制信号汇总



- ALUctr<0> = !ALUop<2> & ALUop<0> +
  ALUop<2> & !func<2> & func<1> & !func<0>
- ALUctr<1> = !ALUop<2> & ALUop<1> & !ALUop<0> +
  ALUop<2> & !func<3> & func<2> & !func<1>
- ALUctr<2> = !ALUop<2> & ALUop<1> & !ALUop<0> +
  ALUop<2> & !func<3> & func<2> & !func<1> & func<0>

Main Control 的真值表



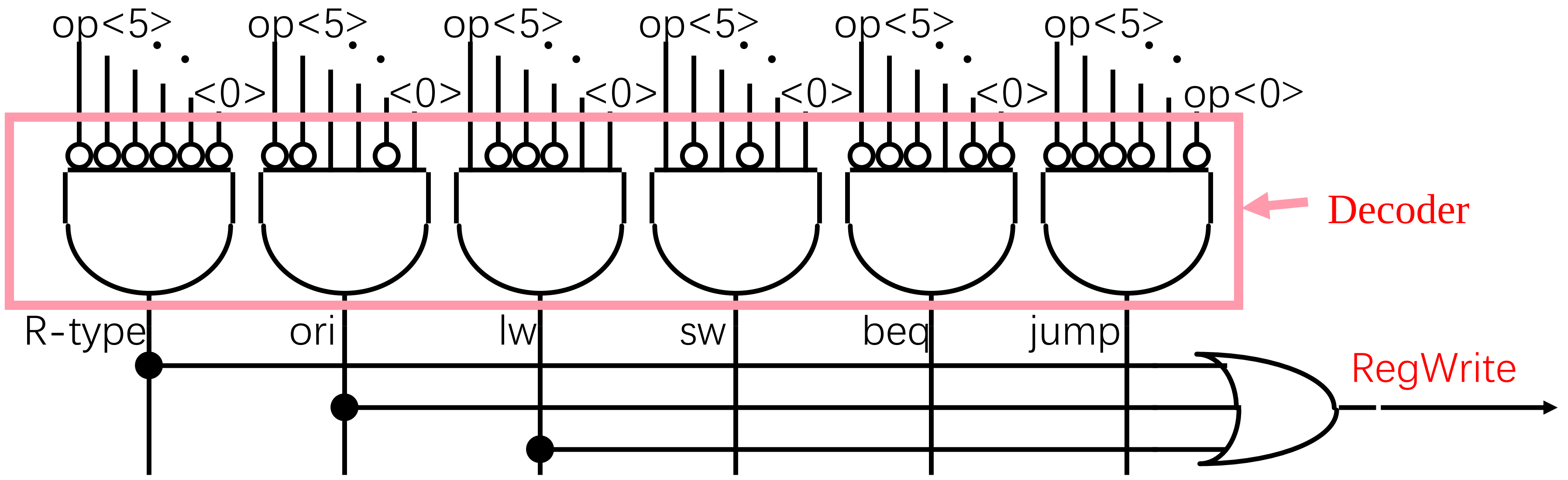
Output of Main Control

Input of main control

op	00 0000	00 1101	10 0011	10 1011	00 0100	00 0010
	R-type	ori	lw	sw	beq	jump
RegDst	1	0	0	x	x	x
ALUSrc	0	1	1	1	0	x
MemtoReg	0	0	1	x	x	x
RegWrite	1	1	1	0	0	0
MemWrite	0	0	0	1	0	0
Branch	0	0	0	0	1	0
Jump	0	0	0	0	0	1
ExtOp	x	0	1	1	x	x
ALUop (Symbolic)	"R-type"	Or	Add	Add	Subtr	xxx
ALUop <2>	1	0	0	0	0	x
ALUop <1>	x	1	0	0	x	x
ALUop <0>	x	0	0	0	1	x

RegWrite 信号的真值表

op	00 0000	00 1101	10 0011	10 1011	00 0100	00 0010
	R-type	ori	lw	sw	beq	jump
RegWrite	1	1	1	0	0	0



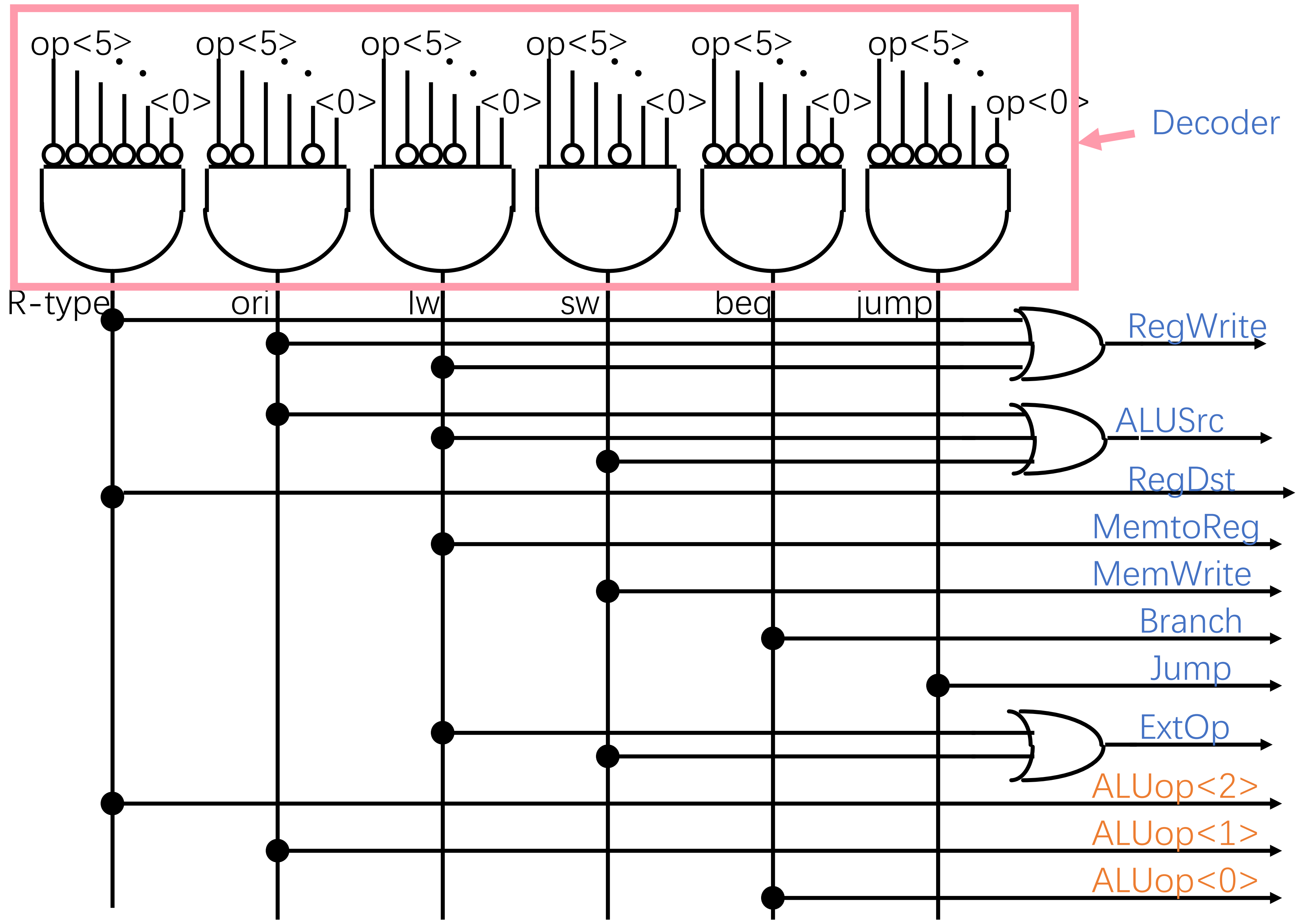
- RegWrite = R-type + ori + lw

  = !op<5> & !op<4> & !op<3> & !op<2> & !op<1> & !op<0> (R-type)

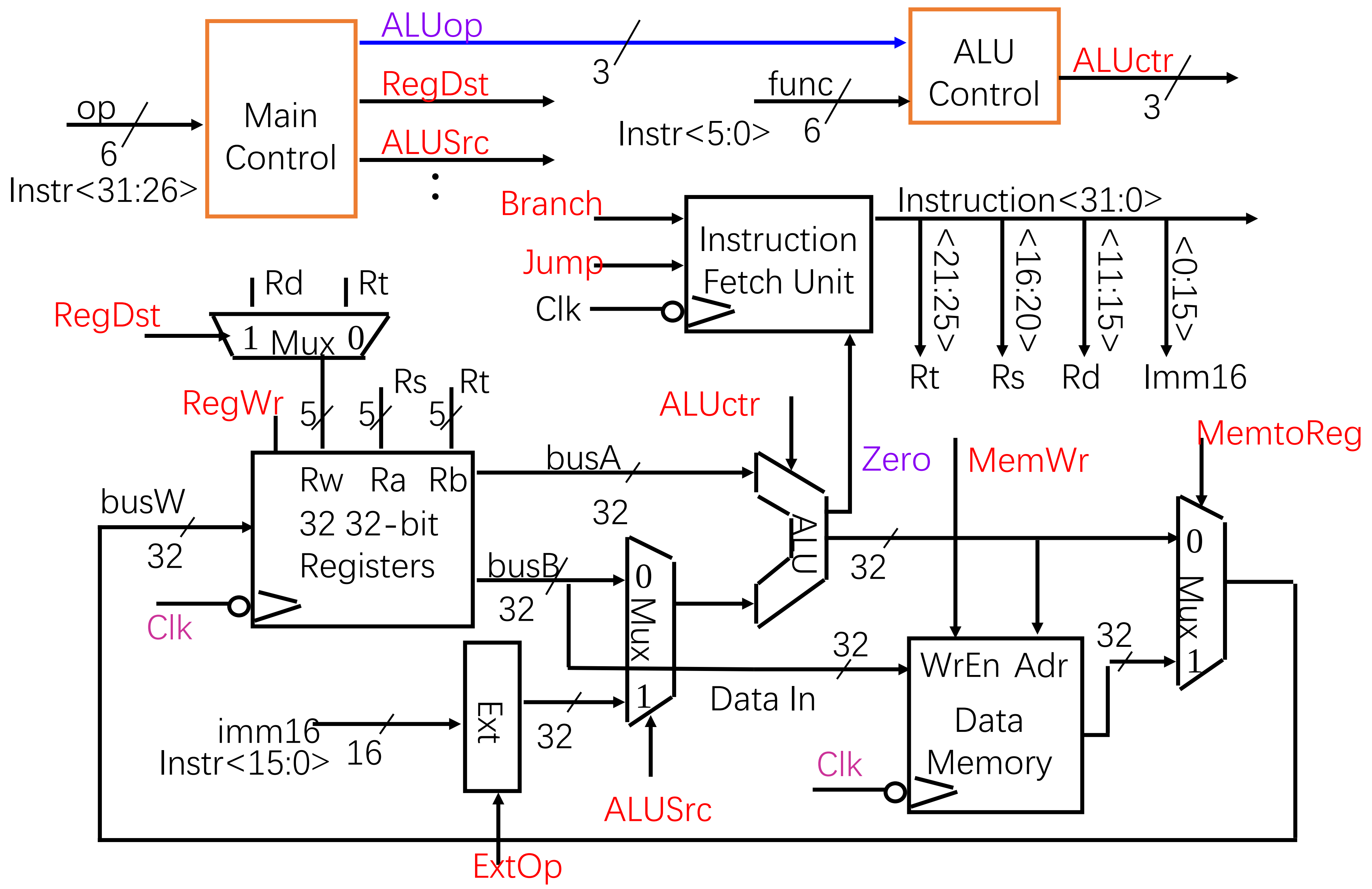
  + !op<5> & !op<4> & op<3> & op<2> & !op<1> & op<0> (ori)

  + op<5> & !op<4> & !op<3> & !op<2> & op<1> & op<0> (lw)

其他控制信号真值表


op<5>
<0>
op<0>
Decoder
R-type
ori
lw
sw
beq
jump
RegWrite
ALUSrc
RegDst
MemtoReg
MemWrite
Branch
Jump
ExtOp
ALUop<2>
ALUop<1>
ALUop<0>

完整的数据通路



ALUop
3
Main Control
op
6
Instr<31:26>
RegDst
ALUSrc
func
6
Instr<5:0>
ALU Control
ALUctr
3
Branch
Jump
Clk
Instruction Fetch Unit
Instruction<31:0>
<21:25>
<16:20>
<11:15>
<0:15>
Rt
Rs
Rd
Imm16
RegDst
Rd
Rt
1 Mux 0
RegWr
5
5
5
Rs
Rt
ALUctr
Zero
MemWr
MemtoReg
busA
32
Rw Ra Rb
32 32-bit Registers
busW
32
Clk
busB
32
0 Mux 1
ALU
32
0 Mux 1
32
32
WrEn Adr
Data In
Data Memory
Clk
imm16
Instr<15:0>
16
Ext
32
ALUSrc
ExtOp

小结：设计处理器的五个步骤

1. 分析指令系统， 得出对数据通路的需求
   - The meaning of each instruction is given by the register transfers
   - 例如：ADDU指令的数据通路需求：R[rd] <– R[rs] + R[rt];
2. 选择数据通路上合适的组件
   - 例如：加法器？ 算术逻辑运算单元？寄存器堆？
3. 连接组件构成数据通路
4. 分析每一条指令的实现，以确定控制信号，
   - 不同的控制信号影响寄存器之间的数据传送
5. 集成控制信号，完成控制逻辑

SHANGHAI JIAO TONG UNIVERSITY

谢谢！