CPS104
Computer Organization and Programming
Lecture 15: Designing Single Cycle Control

Oct. 22, 1999

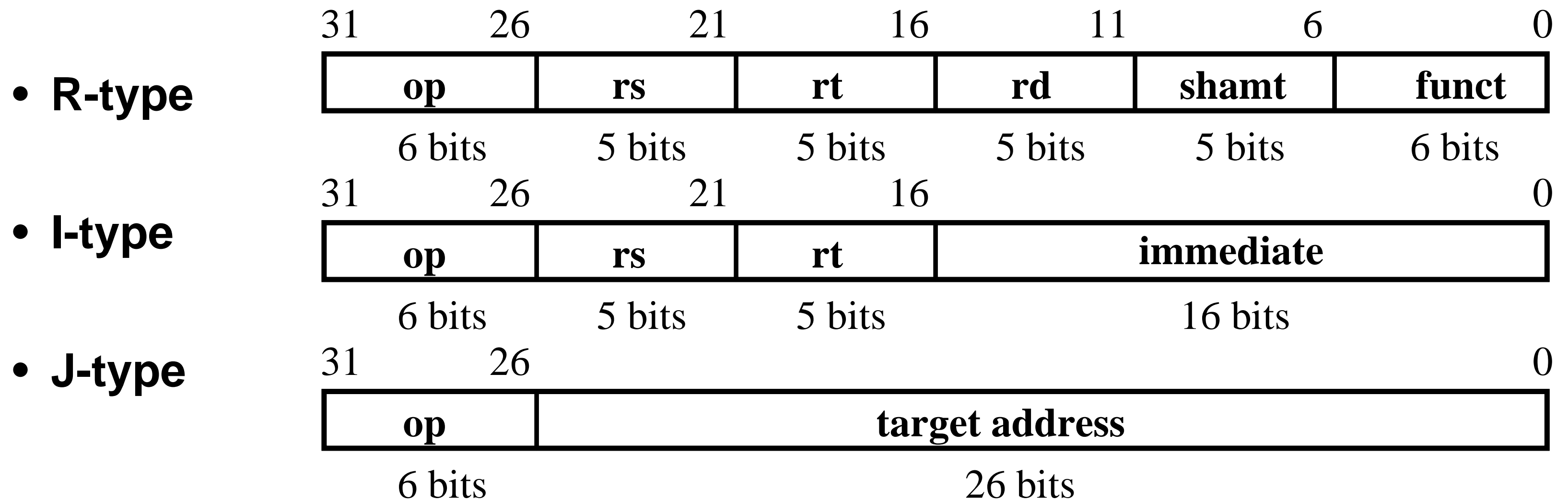
Dietolf (Dee) Ramm

http://www.cs.duke.edu/~dr/cps104.html

©GK&DR Fall 1999

Recap: The MIPS Instruction Formats

° All MIPS instructions are 32 bits long. The three instruction formats:



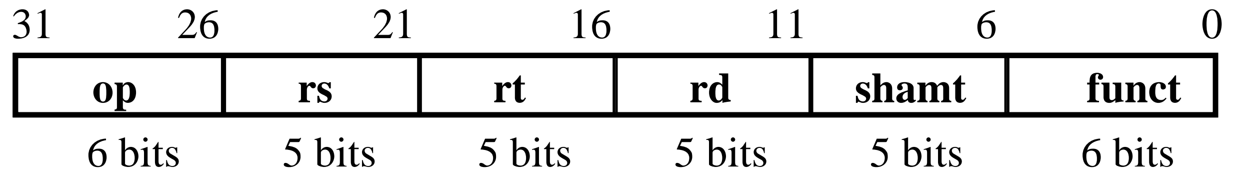
° The different fields are:

- **op**: operation of the instruction
- **rs, rt, rd**: the source and destination registers specifier
- **shamt**: shift amount
- **funct**: selects the variant of the operation in the "op" field
- **address / immediate**: address offset or immediate value
- **target address**: target address of the jump instruction

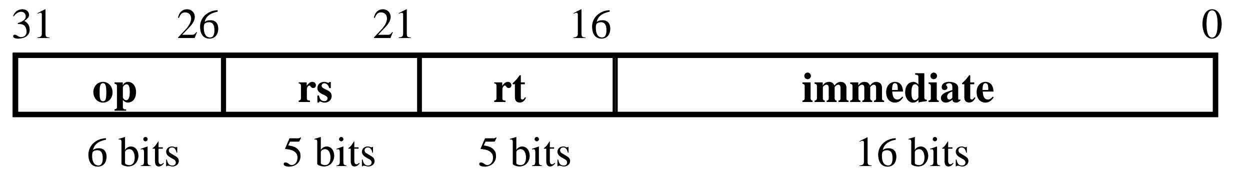
©GK&DR Fall 1999

Recap: The MIPS Subset

° ADD and subtract
  - add rd, rs, rt
  - sub rd, rs, rt



° OR Imm:
  - ori rt, rs, imm16



° LOAD and STORE
  - lw rt, rs, imm16
  - sw rt, rs, imm16

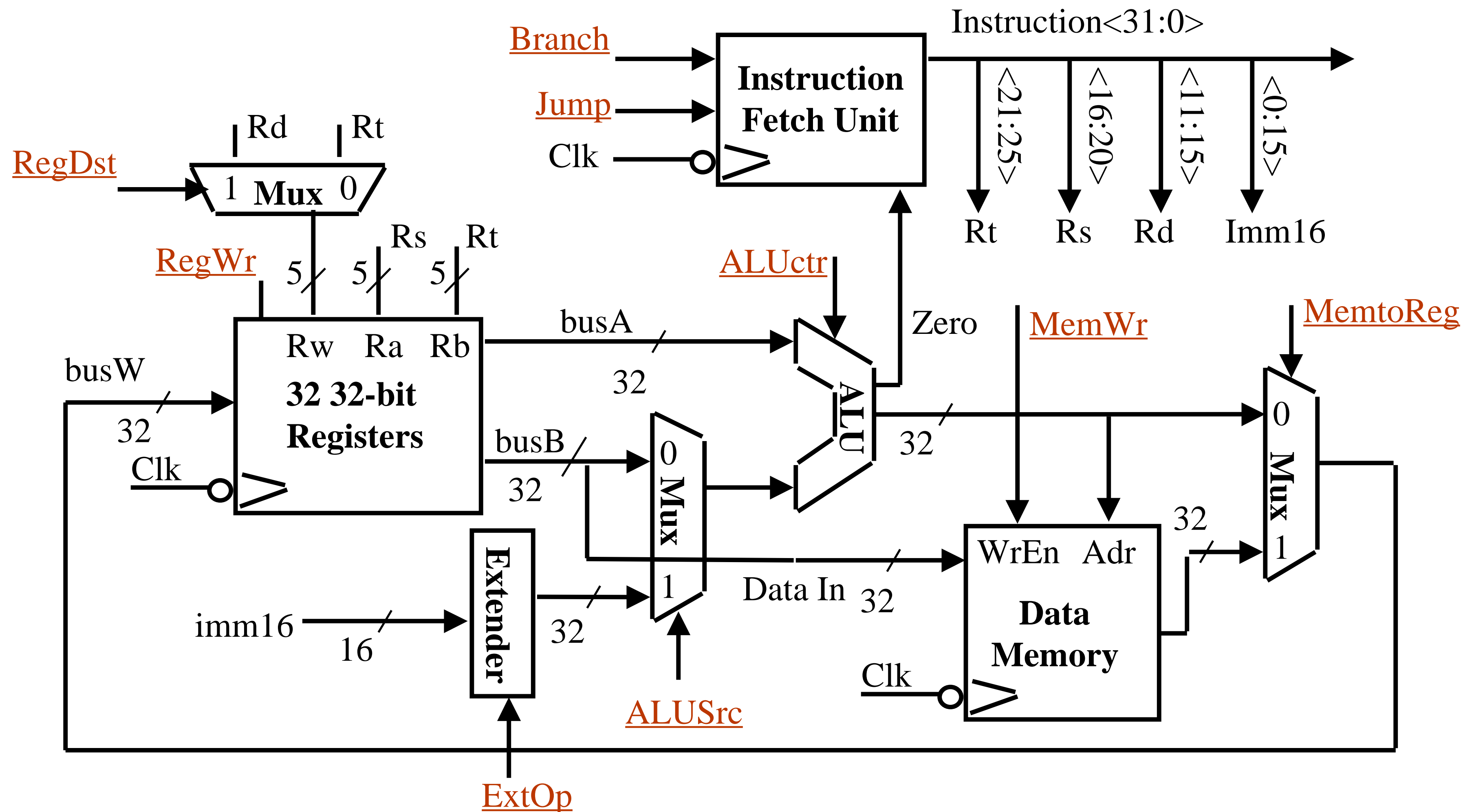
° BRANCH:
  - beq rs, rt, imm16

° JUMP:
  - j target

©GK&DR Fall 1999

Recap: A Single Cycle Datapath

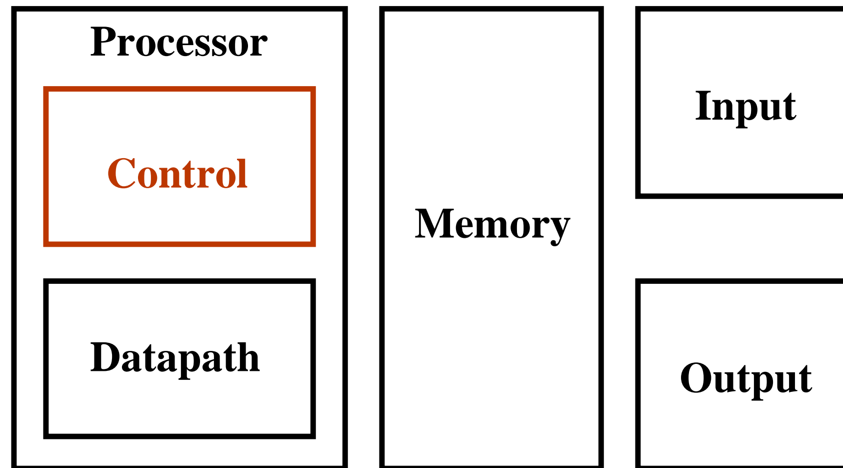
° We have everything except control signals (underline)
  - Today's lecture will show you how to generate the control signals



©GK&DR Fall 1999

The Big Picture: Where are We Now?

° The Five Classic Components of a Computer



° Today's Topic: Designing the Control for the Single Cycle Datapath

©GK&DR Fall 1999

Outline of Today's Lecture

- Control for Register-Register & Or Immediate instructions
- Control signals for Load, Store, Branch, & Jump
- Building a local controller: ALU Control
- The main controller
- Summary

©GK&DR Fall 1999

RTL: The ADD Instruction

31 26	21	16	11	6	0
op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

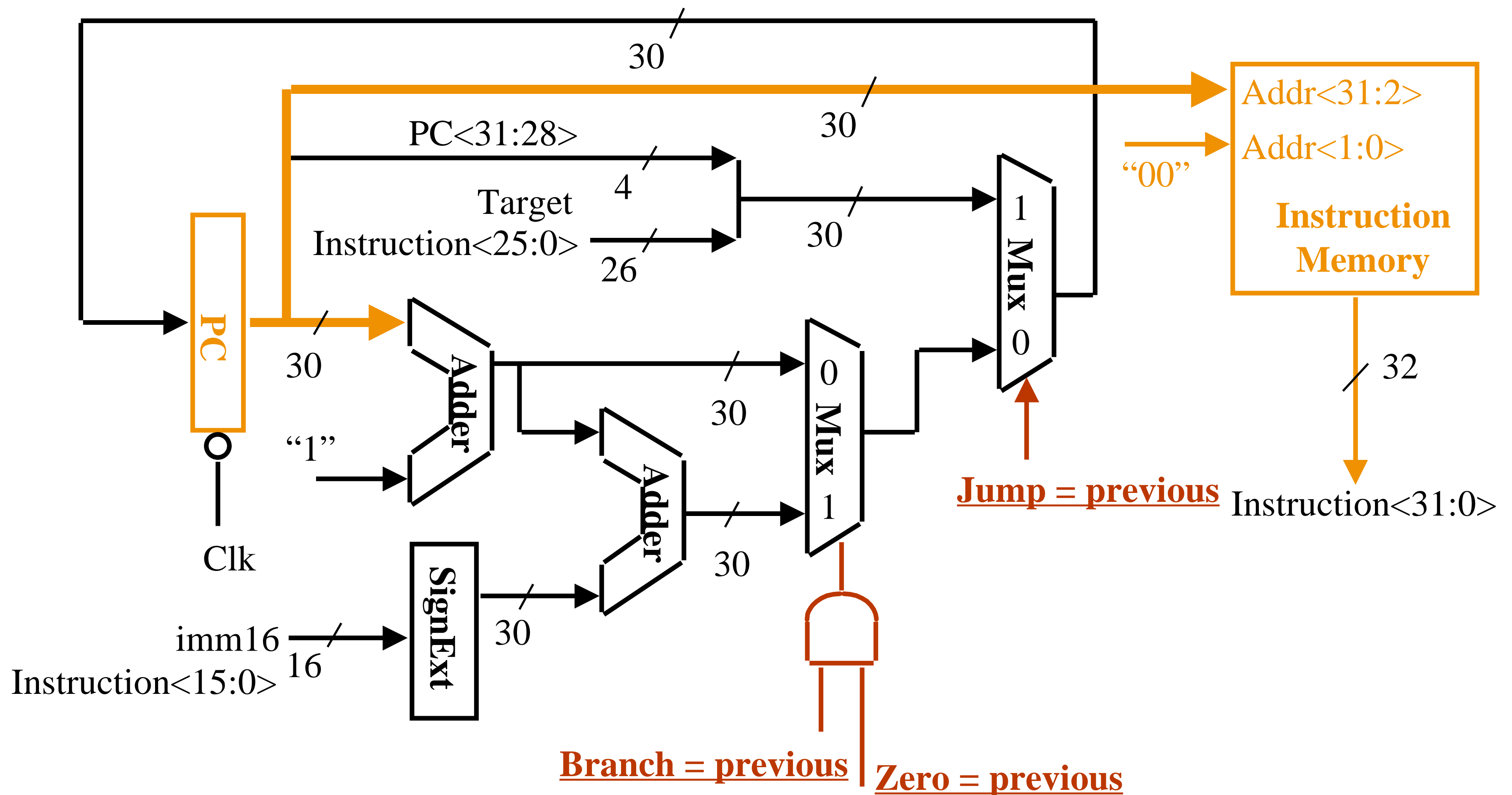
° add rd, rs, rt

- mem[PC] — Fetch the instruction from memory
- R[rd] <- R[rs] + R[rt] — The actual operation
- PC <- PC + 4 — Calculate the next instruction's address

©GK&DR Fall 1999

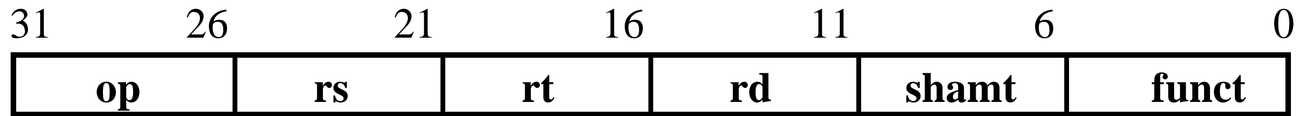
Instruction Fetch Unit at the Beginning of Add / Subtract

° Fetch the instruction from Instruction memory: Instruction <- mem[PC]
  • This is the same for all instructions



©GK&DR Fall 1999

The Single Cycle Datapath during Add and Subtract



31	26	21	16	11	6	0
op	rs	rt	rd	shamt	funct	

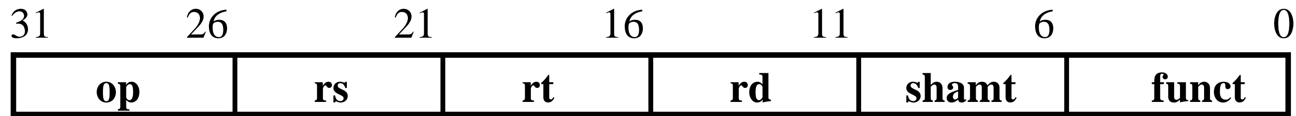
° R[rd] <- R[rs] + / - R[rt]

Branch = ?
Jump = ?
RegDst = ?
RegWr = ?
ALUctr = ?
MemWr = ?
MemtoReg = ?
ALUSrc = ?
ExtOp = ?

Instruction<31:0>
Instruction Fetch Unit
Clk
<21:25> <16:20> <11:15> <0:15>
Rt Rs Rd Imm16
Rd Rt
1 Mux 0
Rs Rt
5 5 5
Rw Ra Rb
32 32-bit Registers
busW 32 Clk
busA 32
busB 32
ALU
Zero
32
0 Mux 1
Extender
imm16 16
32
Data In 32
WrEn Adr
Data Memory
Clk
32
0 Mux 1

©GK&DR Fall 1999

The Single Cycle Datapath during Add and Subtract



° R[rd] <- R[rs] + / - R[rt]

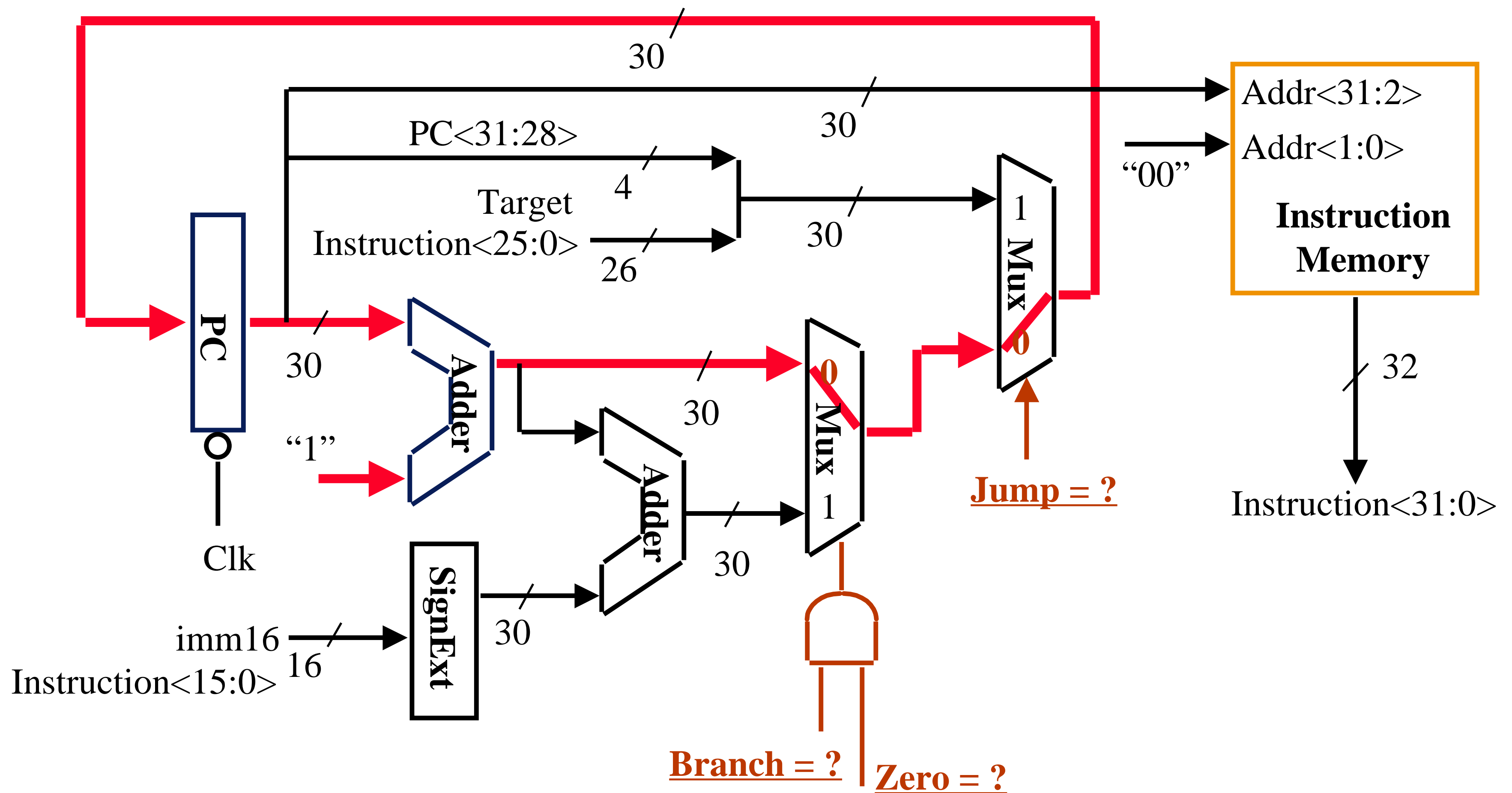
Branch = 0
Jump = 0
Clk
Instruction Fetch Unit
Instruction<31:0>
<21:25>
<16:20>
<11:15>
<0:15>
Rt
Rs
Rd
Imm16
RegDst = 1
Rd
Rt
1 Mux 0
RegWr = 1
Rs
Rt
5
5
5
ALUctr = Add or Sub
Zero
MemtoReg = 0
MemWr = 0
busA
32
Rw Ra Rb
32 32-bit Registers
busW
32
Clk
busB
32
0
Mux
1
ALU
32
0
Mux
1
32
WrEn Adr
Data In
32
Data Memory
Clk
imm16
16
Extender
32
ALUSrc = 0
ExtOp = x

©GK&DR Fall 1999

Instruction Fetch Unit at the End of Add and Subtract

° PC <- PC + 4

- This is the same for all instructions except: Branch and Jump

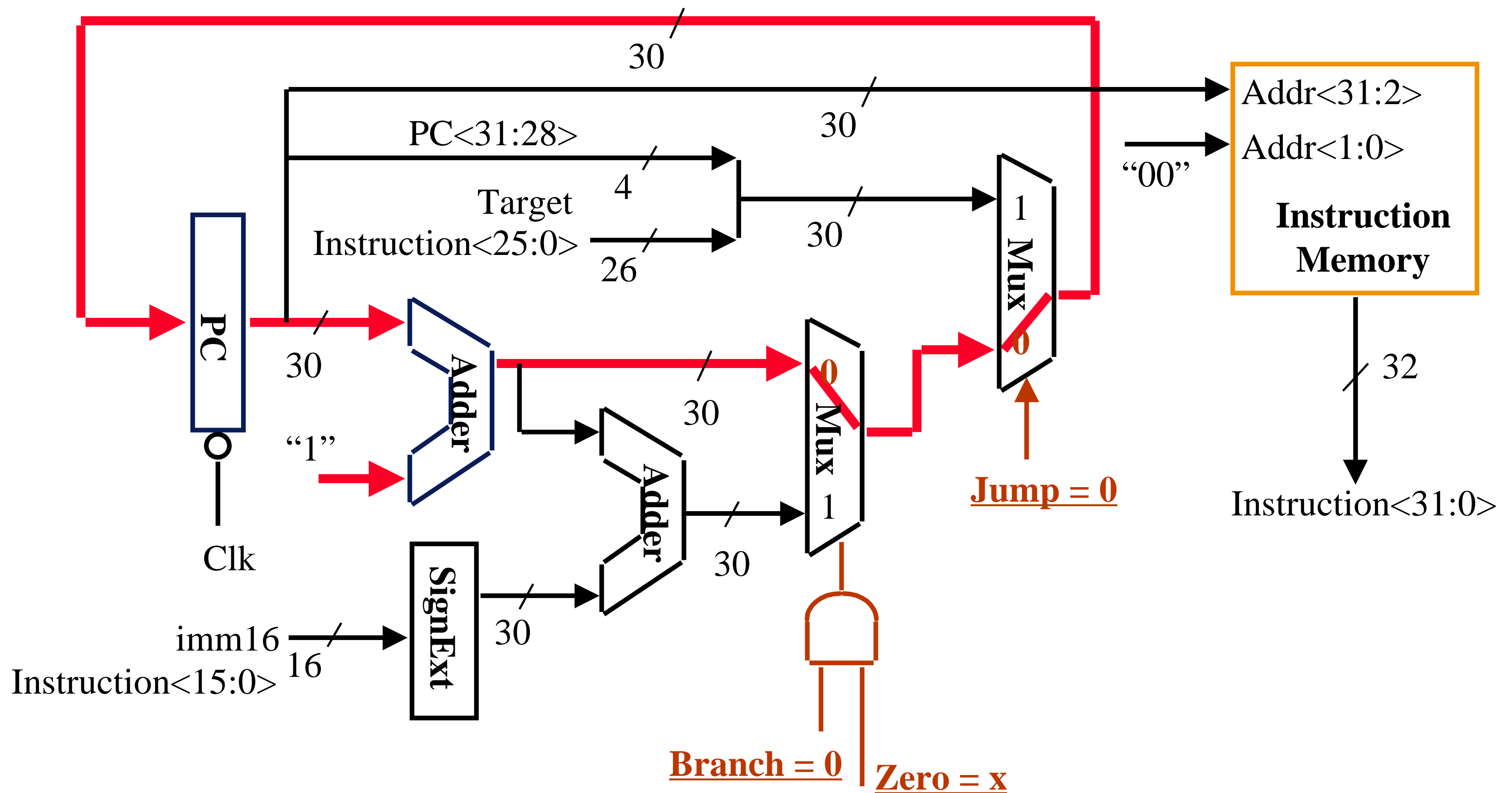


©GK&DR Fall 1999

Instruction Fetch Unit at the End of Add and Subtract

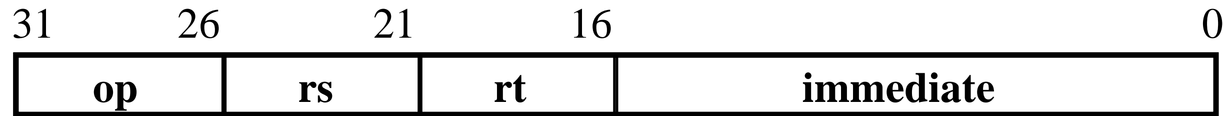
° PC <- PC + 4

- This is the same for all instructions except: Branch and Jump



©GK&DR Fall 1999

The Single Cycle Datapath during Or Immediate

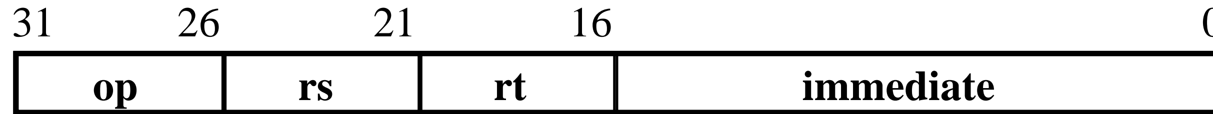


- R[rt] <- R[rs] or ZeroExt[Imm16]

Branch = ?
Jump = ?
Clk
Instruction Fetch Unit
Instruction<31:0>
<21:25>
<16:20>
<11:15>
<0:15>
Rt
Rs
Rd
Imm16
MemtoReg = ?
RegDst = ?
Rd
Rt
1 Mux 0
RegWr = ?
5
5
5
Rs
Rt
ALUctr = ?
Zero
MemWr = ?
busW
32
Clk
Rw Ra Rb
32 32-bit Registers
busA
32
busB
32
0
Mux
1
ALU
32
0
Mux
1
32
WrEn Adr
Data In
32
Data Memory
Clk
imm16
16
Extender
32
ALUSrc = ?
ExtOp = ?

©GK&DR Fall 1999

The Single Cycle Datapath during Or Immediate

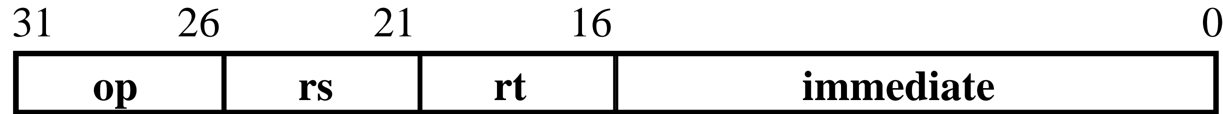


° R[rt] <- R[rs] or ZeroExt[Imm16]

Branch = 0
Jump = 0
Clk
Instruction Fetch Unit
Instruction<31:0>
<21:25>
<16:20>
<11:15>
<0:15>
Rt
Rs
Rd
Imm16
RegDst = 0
Rd
Rt
1 Mux 0
RegWr = 1
5
5
Rs
5
Rt
ALUctr = Or
Zero
MemWr = 0
MemtoReg = 0
busW
32
Clk
Rw Ra Rb
32 32-bit Registers
busA
32
busB
32
0 Mux 1
ALU
32
WrEn Adr
Data In
32
Data Memory
Clk
32
0 Mux 1
imm16
16
Extender
32
ALUSrc = 1
ExtOp = 0

©GK&DR Fall 1999

The Single Cycle Datapath during Load

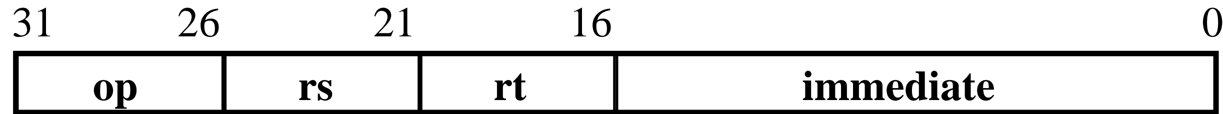


° R[rt] <- Data Memory {R[rs] + SignExt[imm16]}

Instruction<31:0>
Branch = ?
Jump = ?
Clk
Instruction Fetch Unit
<21:25>
<16:20>
<11:15>
<0:15>
Rt
Rs
Rd
Imm16
RegDst = ?
Rd
Rt
1 Mux 0
RegWr = ?
5
5
Rs
5
Rt
ALUctr = ?
MemtoReg = ?
Zero
MemWr = ?
busA
32
Rw Ra Rb
32 32-bit Registers
busW
32
Clk
busB
32
0
Mux
1
ALU
32
0
Mux
1
WrEn Adr
Data In
32
Data Memory
Clk
32
imm16
16
Extender
32
ALUSrc = ?
ExtOp = ?

©GK&DR Fall 1999

The Single Cycle Datapath during Load

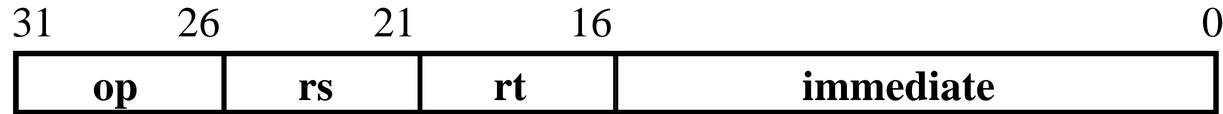


- R[rt] <- Data Memory {R[rs] + SignExt[imm16]}

Branch = 0
Jump = 0
Clk
Instruction Fetch Unit
Instruction<31:0>
<21:25>
<16:20>
<11:15>
<0:15>
Rt
Rs
Rd
Imm16
RegDst = 0
Rd
Rt
1 Mux 0
RegWr = 1
5
5
Rs
5
Rt
ALUctr = Add
Zero
MemWr = 0
MemtoReg = 1
busW
32
Clk
Rw Ra Rb
32 32-bit Registers
busA
32
busB
32
0
Mux
1
ALU
32
0
Mux
1
WrEn Adr
Data In
32
Data Memory
Clk
32
imm16
16
Extender
32
ALUSrc = 1
ExtOp = 1

©GK&DR Fall 1999

The Single Cycle Datapath during Store

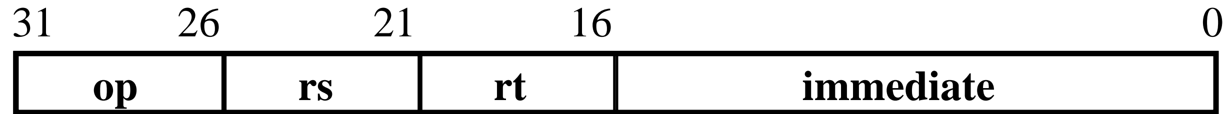


° Data Memory {R[rs] + SignExt[imm16]} <- R[rt]

Instruction<31:0>
Branch =
Jump =
Instruction Fetch Unit
Clk
<21:25>
<16:20>
<11:15>
<0:15>
Rt
Rs
Rd
Imm16
RegDst =
Rd
Rt
1 Mux 0
RegWr =
Rs
Rt
5
5
5
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
WrEn Adr
Data In 32
Data Memory
Clk
imm16
16
Extender
32
ALUSrc =
ExtOp =

©GK&DR Fall 1999

The Single Cycle Datapath during Store

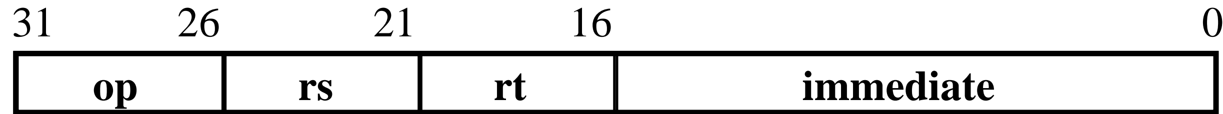


- Data Memory {R[rs] + SignExt[imm16]} <- R[rt]

Instruction<31:0>
Branch = 0
Jump =0
Clk
Instruction Fetch Unit
<21:25>
<16:20>
<11:15>
<0:15>
Rt
Rs
Rd
Imm16
RegDst =x
Rd
Rt
1 Mux 0
RegWr = 0
5
5
5
Rs
Rt
ALUctr
MemtoReg =x
busA
Zero
MemWr =1
busW
Rw Ra Rb
32 32-bit Registers
32
32
Clk
busB
32
0 Mux 1
ALU
32
0 Mux 1
32
WrEn Adr
Data In
32
Data Memory
Clk
imm16
16
Extender
32
ALUSrc =1
ExtOp = 1

©GK&DR Fall 1999

The Single Cycle Datapath during Branch

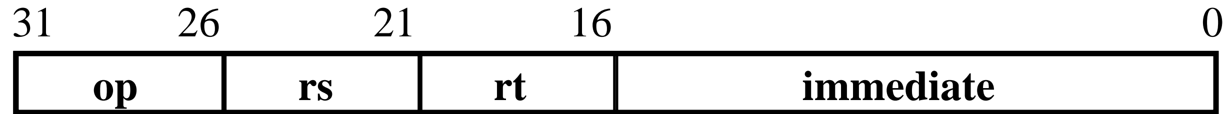


- if (R[rs] - R[rt] == 0) then Zero <- 1 ; else Zero <- 0

Branch = ?
Jump = ?
Instruction<31:0>
Instruction Fetch Unit
Clk
<21:25>
<16:20>
<11:15>
<0:15>
Rt
Rs
Rd
Imm16
RegDst = ?
Rd
Rt
1 Mux 0
ALUctr =?
MemtoReg = ?
RegWr = ?
5
5
5
Rs
Rt
Zero
MemWr = ?
Rw Ra Rb
busA
32
busW
32 32-bit Registers
32
ALU
32
0
Mux
1
Clk
busB
32
0
Mux
1
WrEn Adr
32
Data In
32
Data Memory
imm16
16
Extender
32
ALUSrc = ?
Clk
ExtOp = ?

©GK&DR Fall 1999

The Single Cycle Datapath during Branch



° if (R[rs] - R[rt] == 0) then Zero <- 1 ; else Zero <- 0

Branch = 1
Jump = 0
Clk
Instruction Fetch Unit
Instruction<31:0>
<21:25>
<16:20>
<11:15>
<0:15>
Rt
Rs
Rd
Imm16
RegDst = x
Rd
Rt
1 Mux 0
RegWr = 0
5
5
5
Rs
Rt
ALUctr = Subtract
Zero
MemWr = 0
MemtoReg = x
busA
32
busW
32
Clk
Rw Ra Rb
32 32-bit Registers
busB
32
0 Mux 1
ALU
32
0 Mux 1
32
WrEn Adr
Data In
32
Data Memory
Clk
imm16
16
Extender
32
ALUSrc = 0
ExtOp = x

©GK&DR Fall 1999

Instruction Fetch Unit at the End of Branch

31	26	21	16	0
op	rs	rt	immediate	

- if (Zero == 1) then PC = PC + 4 + SignExt[imm16]*4 ; else PC = PC + 4

30
30
PC<31:28>
4
Target
Instruction<25:0>
26
30
Addr<31:2>
"00"
Addr<1:0>
Instruction Memory
1
Mux
0
PC
30
Adder
"1"
Clk
30
0
Mux
1
Adder
30
Jump = ?
32
Instruction<31:0>
imm16
Instruction<15:0>
16
SignExt
30
Assume Zero = 1 to see the interesting case.
Branch = ?
Zero = 1

©GK&DR Fall 1999

Instruction Fetch Unit at the End of Branch

31	26	21	16	0
op	rs	rt	immediate	

- if (Zero == 1) then PC = PC + 4 + SignExt[imm16]*4 ; else PC = PC + 4

30
30
PC<31:28>
Addr<31:2>
Addr<1:0>
"00"
Instruction Memory
4
Target Instruction<25:0>
30
26
1
Mux
0
PC
30
Adder
0
30
Mux
"1"
Adder
1
Jump = 0
32
Instruction<31:0>
Clk
30
SignExt
30
imm16
Instruction<15:0>
16

Assume Zero = 1 to see the interesting case.

Branch = 1
Zero = 1

©GK&DR Fall 1999

The Single Cycle Datapath during Jump

- Nothing to do! Make sure control signals are set correctly!

Branch = 0
Jump = 1
Clk
Instruction Fetch Unit
Instruction<31:0>
<21:25>
<16:20>
<11:15>
<0:15>
Rt
Rs
Rd
Imm16
RegDst = x
Rd
Rt
1 Mux 0
RegWr = 0
5
5
5
Rs
Rt
ALUctr = x
Zero
MemWr = 0
MemtoReg = x
busA
32
Rw Ra Rb
32 32-bit Registers
busW
32
Clk
busB
32
0 Mux 1
ALU
32
0 Mux 1
32
Data In
32
WrEn Adr
Data Memory
Clk
imm16
16
Extender
32
ALUSrc = x
ExtOp = x

©GK&DR Fall 1999

Instruction Fetch Unit at the End of Jump

31	26	0
op	target address	

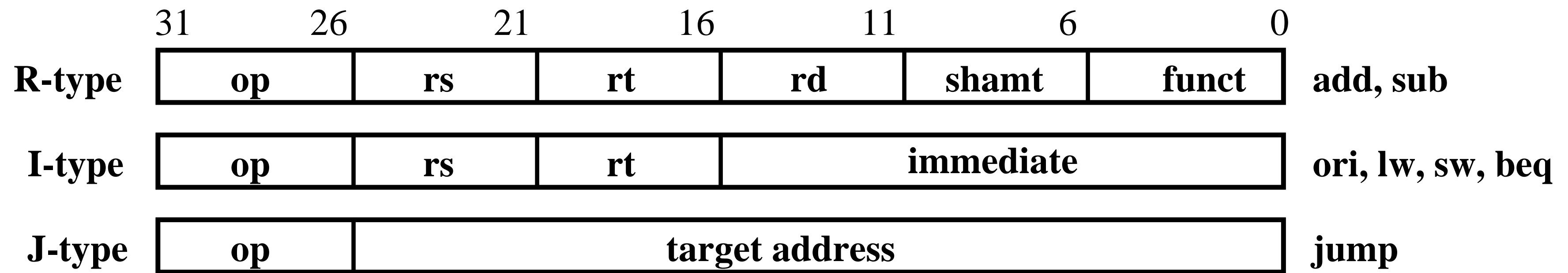
- PC <- PC<31:28> concat target<25:0> concat "00"

©GK&DR Fall 1999

A Summary of the Control Signals

See Appendix A

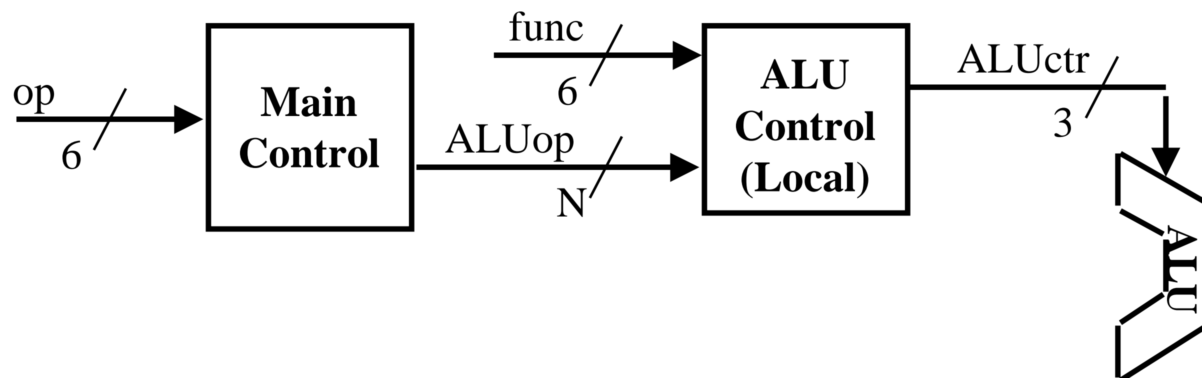
func	10 0000	10 0010	(We Don't Care :-)				
op	00 0000	00 0000	00 1101	10 0011	10 1011	00 0100	00 0010
	add	**sub**	**ori**	**lw**	**sw**	**beq**	**jump**
RegDst	1	1	0	0	x	x	x
ALUSrc	0	0	1	1	1	0	x
MemtoReg	0	0	0	1	x	x	x
RegWrite	1	1	1	1	0	0	0
MemWrite	0	0	0	0	1	0	0
Branch	0	0	0	0	0	1	x
Jump	0	0	0	0	0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



©GK&DR Fall 1999

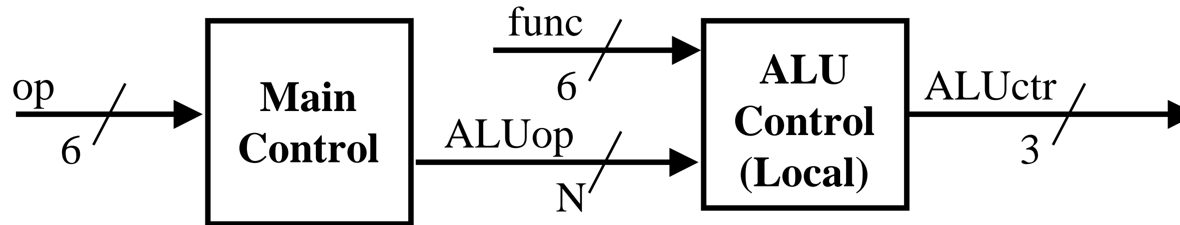
The Concept of Local Decoding

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	x
Jump	0	0	0	0	0	1
ExtOp	x	0	1	1	x	x
ALUop<N:0>	"R-type"	Or	Add	Add	Subtract	xxx



©GK&DR Fall 1999

The Encoding of ALUop



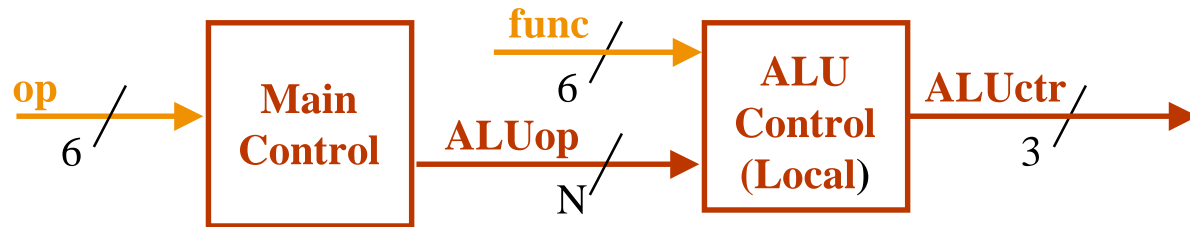
- In this exercise, ALUop has to be 2 bits wide to represent:
  - (1) “R-type” instructions
  - “I-type” instructions that require the ALU to perform:
    - (2) Or, (3) Add, and (4) Subtract

- To implement the full MIPS ISA, ALUop hat to be 3 bits to represent:
  - (1) “R-type” instructions
  - “I-type” instructions that require the ALU to perform:
    - (2) Or, (3) Add, (4) Subtract, and (5) And (Example: andi)

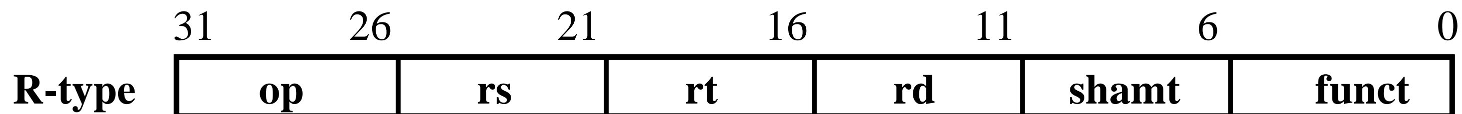
	R-type	ori	lw	sw	beq	jump
ALUop (Symbolic)	“R-type”	Or	Add	Add	Subtract	xxx
ALUop<2:0>	1 00	0 10	0 00	0 00	0 01	xxx

©GK&DR Fall 1999

The Decoding of the “func” Field

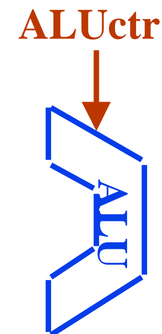


	R-type	ori	lw	sw	beq	jump
ALUop (Symbolic)	“R-type”	Or	Add	Add	Subtract	xxx
ALUop<2:0>	1 00	0 10	0 00	0 00	0 01	xxx



funct<5:0>	Instruction Operation
10 0000	add
10 0010	subtract
10 0100	and
10 0101	or
10 1010	set-on-less-than

ALUctr



ALUctr<2:0>	ALU Operation
000	Add
001	Subtract
010	And
110	Or
111	Set-on-less-than

©GK&DR Fall 1999

The Truth Table for ALUctr

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

ALUop (Symbolic)	R-type	ori	lw	sw	beq
	"R-type"	Or	Add	Add	Subtract
ALUop<2:0>	1 00	0 10	0 00	0 00	0 01

ALUop			func				ALU Operation	ALUctr		
bit<2>	bit<1>	bit<0>	bit<3>	bit<2>	bit<1>	bit<0>		bit<2>	bit<1>	bit<0>
0	0	0	x	x	x	x	Add	0	1	0
0	x	1	x	x	x	x	Subtract	1	1	0
0	1	x	x	x	x	x	Or	0	0	1
1	x	x	0	0	0	0	Add	0	1	0
1	x	x	0	0	1	0	Subtract	1	1	0
1	x	x	0	1	0	0	And	0	0	0
1	x	x	0	1	0	1	Or	0	0	1
1	x	x	1	0	1	0	Set on <	1	1	1

©GK&DR Fall 1999

The Logic Equation for ALUctr<2>

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

©GK&DR Fall 1999

The Logic Equation for ALUctr<1>

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

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

©GK&DR Fall 1999

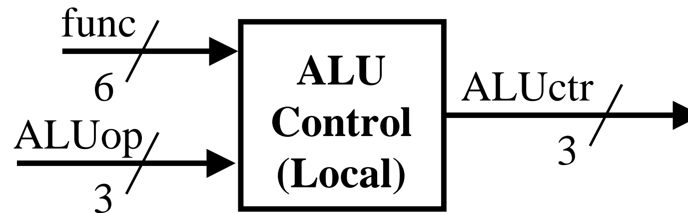
The Logic Equation for ALUctr<0>

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

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

©GK&DR Fall 1999

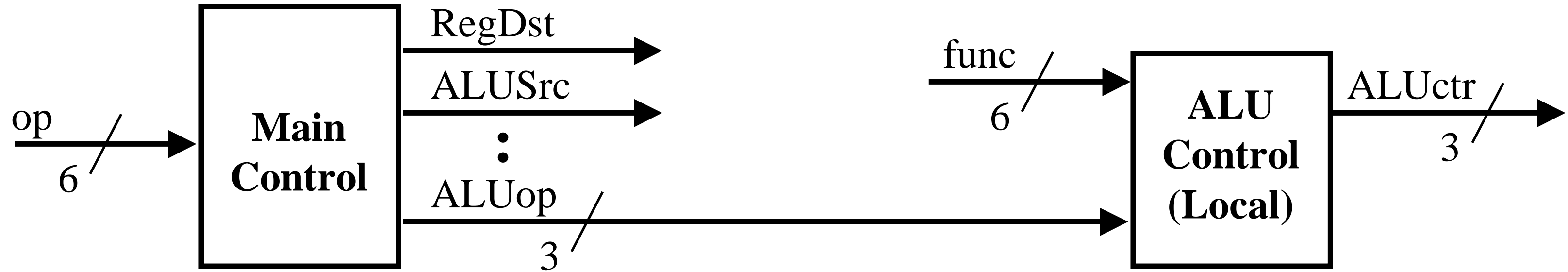
The ALU Control Block



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

©GK&DR Fall 1999

The “Truth Table” for the 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	x
Jump	0	0	0	0	0	1
ExtOp	x	0	1	1	x	x
ALUop (Symbolic)	“R-type”	Or	Add	Add	Subtract	xxx
ALUop <2>	1	0	0	0	0	x
ALUop <1>	0	1	0	0	0	x
ALUop <0>	0	0	0	0	1	x

©GK&DR Fall 1999

The "Truth Table" for RegWrite

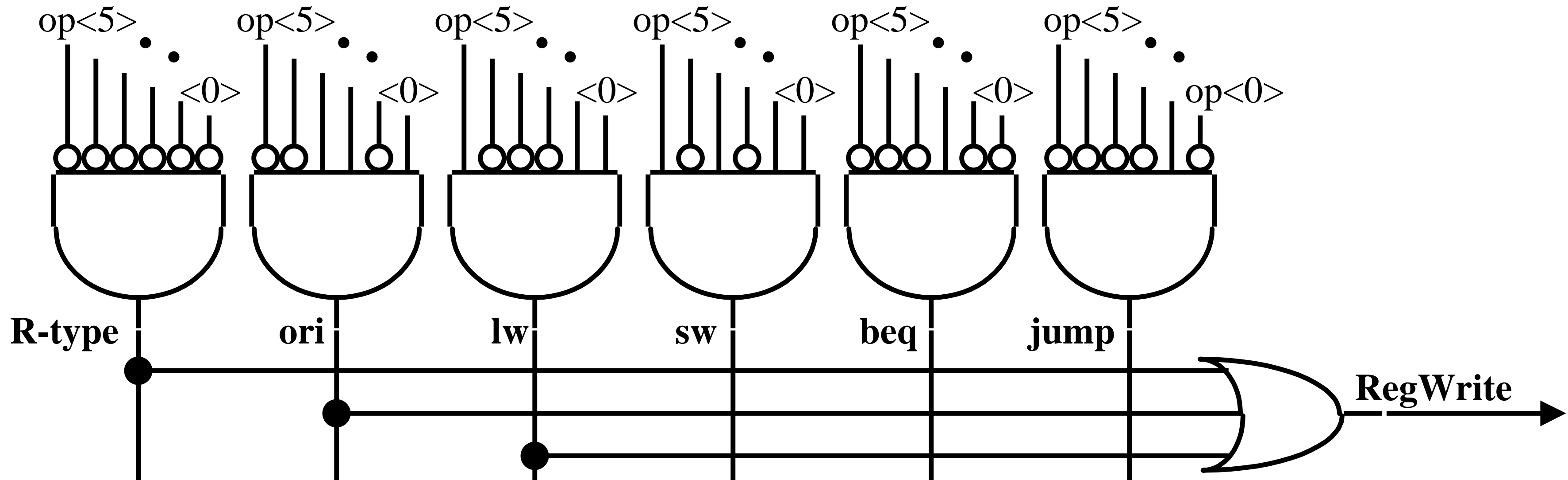
op	00 0000	00 1101	10 0011	10 1011	00 0100	00 0010
	R-type	ori	lw	sw	beq	jump
RegWrite	1	1	1	0	0	0

- RegWrite = R-type + ori + lw

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

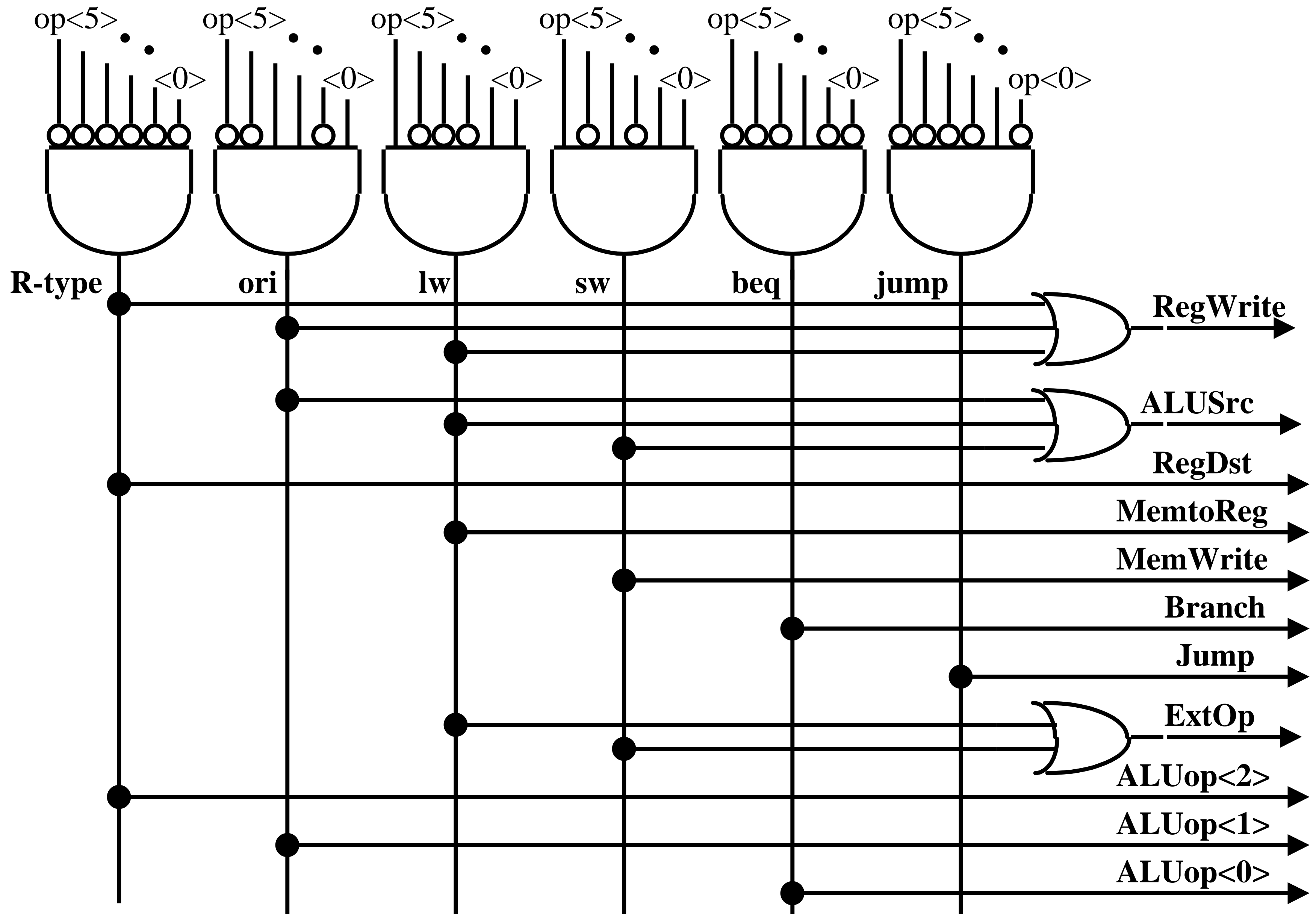
  \+ !op<5> & !op<4> & op<3> & op<2> & !op<1> & op<0> (ori)

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



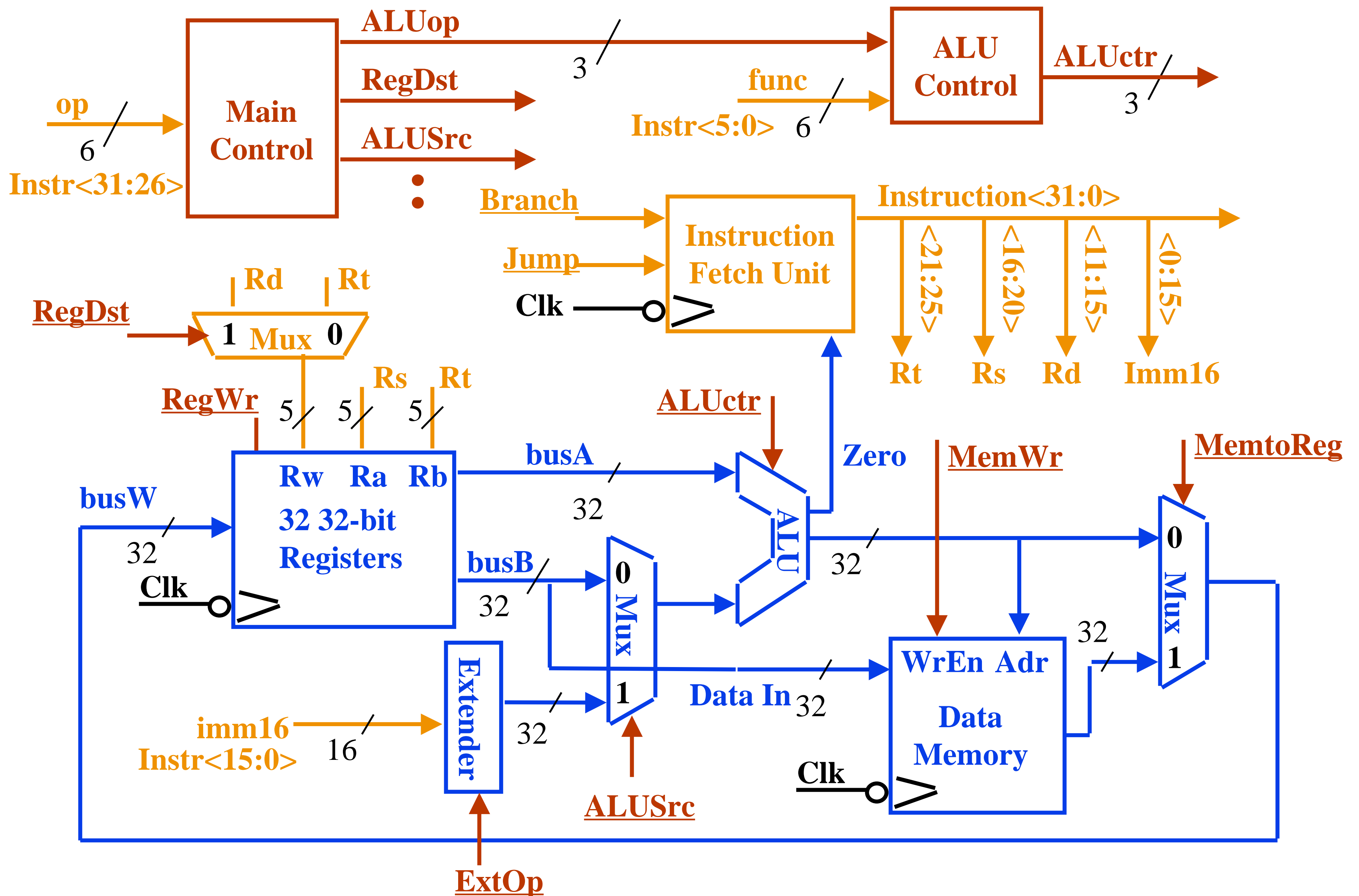
©GK&DR Fall 1999

PLA Implementation of the Main Control



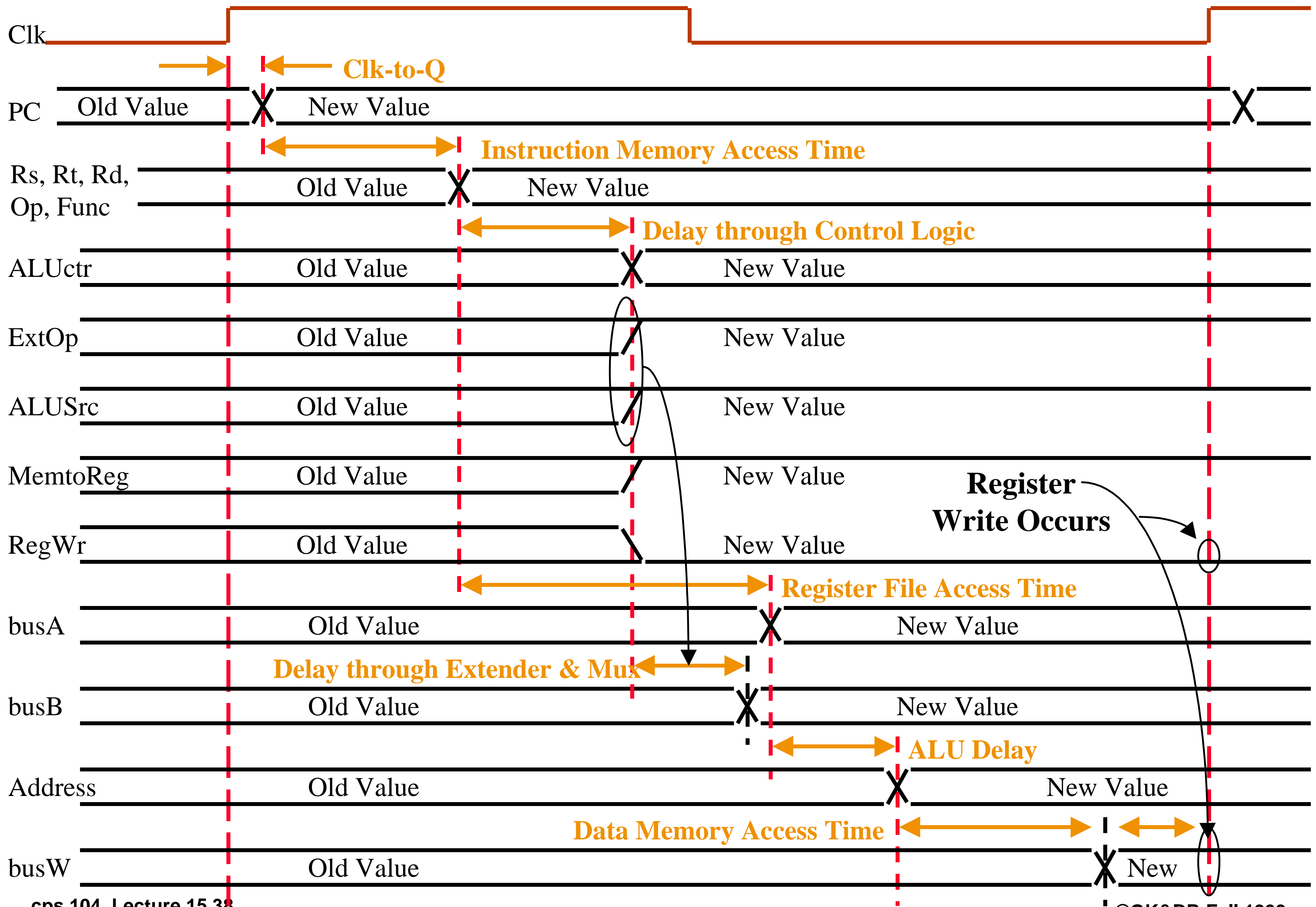
©GK&DR Fall 1999

Putting it All Together: A Single Cycle Processor



©GK&DR Fall 1999

Worst Case Timing: lw $1, $2(offset)



©GK&DR Fall 1999

Drawback of this Single Cycle Processor

- Long cycle time:
  - Cycle time must be long enough for the load instruction:

    PC's Clock -to-Q +

    Instruction Memory Access Time +

    Register File Access Time +

    ALU Delay (address calculation) +

    Data Memory Access Time +

    Register File Setup Time +

    Clock Skew

- Cycle time is much longer than needed for all other instructions

©GK&DR Fall 1999