CS305
Computer Architecture

Control Logic for the Multi-Cycle MIPS Implementation

Bhaskaran Raman
Room 406, KR Building
Department of CSE, IIT Bombay

http://www.cse.iitb.ac.in/~br

Summary of Control Lines

Single Cycle

RegWr: 0=disable, 1=enable

RegDst: 0=Rd, 1=Rt

Mem2Reg: 0=ALUOut, 1=MemOut

ALUSrc: 0=Reg2Val, 1=SgnExtImm

ALUOp2: 00=add, 01=sub, 10=reg-reg

MemRd, MemWr: 0=disable, 1=enable

Branch: 0 for other instrns, 1 for beq

Multi-Cycle

RegWr: 0=disable, 1=enable

RegDst: 0=Rd, 1=Rt

Mem2Reg: 0=ALUOut, 1=MDR

Reg File

ALUIp1Ctl: 0=Reg1Val, 1=PC

ALUIp2Ctl2: 00=Reg2Val, 01=4, 10=SgnExtImm, 10=SgnExtImmShft2

ALUOp2: 00=add, 01=sub, 10=reg-reg

ALU

MemRd, MemWr: 0=disable, 1=enable

IorD: 0=PC, 1=ALUOut

IRWr: 0=disable, 1=enable

MDRWr: 0=disable, 1=enable (not needed)

Mem

PCSrc: 0=OutputOfALU, 1=ALUOut

PCWr: 0=disable, 1=enable

PC write

Generating Control Lines

- Generate control lines as per the execution plan
- Control lines are a function of what happens in each stage in each instruction (i.e. the execution plan)

Control Lines in Stage-1

Stage-1:
(a) Instruction fetch
IR = Mem[PC]
(b) PC = PC+4

RegWr: **0=disable**, 1=enable

RegDst: 0=Rd, 1=Rt

Mem2Reg: 0=ALUOut, 1=MDR

*ALUIp1Ctl: 0=Reg1Val, **1=PC***

*ALUIp2Ctl2: 00=Reg2Val, **01=4**, 10=SgnExtImm, 10=SgnExtImmShft2*

ALUOp2: **00=add**, 01=sub, 10=reg-reg

MemRd: 0=disable, **1=enable**

MemWr: **0=disable**, 1=enable

*IorD: **0=PC**, 1=ALUOut*

*IRWr: 0=disable, **1=enable***

*PCSrc: **0=OutputOfALU**, 1=ALUOut*

*PCWr: 0=disable, **1=enable***

Control Lines in Stage-2

Stage-2:

(a) Register read
Reg1T = Reg[Rs]
Reg2T = Reg[Rt]

(b) Compute BrTgt
ALUOut =
(PC+4) + SgnExtImmShift2

RegWr: **0=disable**, 1=enable

RegDst: 0=Rd, 1=Rt

Mem2Reg: 0=ALUOut, 1=MDR

*ALUIp1Ctl: 0=Reg1Val, **1=PC***

*ALUIp2Ctl2: 00=Reg2Val, 01=4, 10=SgnExtImm, **10=SgnExtImmShft2***

ALUOp2: **00=add**, 01=sub, 10=reg-reg

MemRd: **0=disable**, 1=enable

MemWr: **0=disable**, 1=enable

IorD: 0=PC, 1=ALUOut

*IRWr: **0=disable**, 1=enable*

PCSrc: 0=OutputOfALU, 1=ALUOut

*PCWr: **0=disable**, 1=enable*

Control Lines in Stage-3: reg-reg

Stage-3: reg-reg
(a) ALU operation
ALUOut = Reg1T op Reg2T

RegWr: **0=disable**, 1=enable

RegDst: 0=Rd, 1=Rt

Mem2Reg: 0=ALUOut, 1=MDR

*ALUIp1Ctl: **0=Reg1Val**, 1=PC*

*ALUIp2Ctl2: **00=Reg2Val**, 01=4, 10=SgnExtImm, 10=SgnExtImmShft2*

ALUOp2: 00=add, 01=sub, **10=reg-reg**

MemRd: **0=disable**, 1=enable

MemWr: **0=disable**, 1=enable

IorD: 0=PC, 1=ALUOut

*IRWr: **0=disable**, 1=enable*

PCSrc: 0=OutputOfALU, 1=ALUOut

*PCWr: **0=disable**, 1=enable*

Control Lines in Stage-3: `lw, sw`

Stage-3: lw, sw

(a) Mem addr computation

ALUOut = Reg1T + SgnExtImm

RegWr: **0=disable**, 1=enable

RegDst: 0=Rd, 1=Rt

Mem2Reg: 0=ALUOut, 1=MDR

*ALUIp1Ctl: **0=Reg1Val**, 1=PC*

*ALUIp2Ctl2: 00=Reg2Val, 01=4, **10=SgnExtImm**, 10=SgnExtImmShft2*

ALUOp2: **00=add**, 01=sub, 10=reg-reg

MemRd: **0=disable**, 1=enable

MemWr: **0=disable**, 1=enable

IorD: 0=PC, 1=ALUOut

*IRWr: **0=disable**, 1=enable*

PCSrc: 0=OutputOfALU, 1=ALUOut

*PCWr: **0=disable**, 1=enable*

Control Lines in Stage-3: beq

Stage-3: beq

(a) Branch condition check
ALUOut = Reg1T – Reg2T
zero? = (ALUOut != 0)

(b) If zero?, PC = BrTgt (ALUOut)

RegWr: **0=disable**, 1=enable

RegDst: 0=Rd, 1=Rt

Mem2Reg: 0=ALUOut, 1=MDR

*ALUIp1Ctl: **0=Reg1Val**, 1=PC*

*ALUIp2Ctl2: **00=Reg2Val**, 01=4, 10=SgnExtImm, 10=SgnExtImmShft2*

ALUOp2: 00=add, **01=sub**, 10=reg-reg

MemRd: **0=disable**, 1=enable

MemWr: **0=disable**, 1=enable

IorD: 0=PC, 1=ALUOut

*IRWr: **0=disable**, 1=enable*

*PCSrc: 0=OutputOfALU, **1=ALUOut***

PCWr: 0=disable, 1=enable ***zero?***

Control Lines in Stage-4: reg-reg

Stage-4: reg-reg
(a) Write back ALUOut to RegFile

RegWr: 0=disable, **1=enable**

RegDst: **0=Rd**, 1=Rt

Mem2Reg: **0=ALUOut**, 1=MDR

ALUIp1Ctl: 0=Reg1Val, 1=PC

ALUIp2Ctl2: 00=Reg2Val, 01=4, 10=SgnExtImm, 10=SgnExtImmShft2

ALUOp2: 00=add, 01=sub, 10=reg-reg

MemRd: **0=disable**, 1=enable

MemWr: **0=disable**, 1=enable

IorD: 0=PC, 1=ALUOut

*IRWr: **0=disable**, 1=enable*

PCSrc: 0=OutputOfALU, 1=ALUOut

*PCWr: **0=disable**, 1=enable*

Control Lines in Stage-4: `lw`

Stage-4: lw
(a) Read memory
MDR = Mem[ALUOut]

RegWr: **0=disable**, 1=enable

RegDst: 0=Rd, 1=Rt

Mem2Reg: 0=ALUOut, 1=MDR

ALUIp1Ctl: 0=Reg1Val, 1=PC

ALUIp2Ctl2: 00=Reg2Val, 01=4, 10=SgnExtImm, 10=SgnExtImmShft2

ALUOp2: 00=add, 01=sub, 10=reg-reg

MemRd: 0=disable, **1=enable**

MemWr: **0=disable**, 1=enable

*IorD: 0=PC, **1=ALUOut***

*IRWr: **0=disable**, 1=enable*

PCSrc: 0=OutputOfALU, 1=ALUOut

*PCWr: **0=disable**, 1=enable*

Control Lines in Stage-4: sw

Stage-4: sw
(a) Write memory
Mem[ALUOut] = Reg2T

RegWr: **0=disable**, 1=enable

RegDst: 0=Rd, 1=Rt

Mem2Reg: 0=ALUOut, 1=MDR

ALUIp1Ctl: 0=Reg1Val, 1=PC

ALUIp2Ctl2: 00=Reg2Val, 01=4, 10=SgnExtImm, 10=SgnExtImmShft2

ALUOp2: 00=add, 01=sub, 10=reg-reg

MemRd: **0=disable**, 1=enable

MemWr: 0=disable, **1=enable**

*IorD: 0=PC, **1=ALUOut***

*IRWr: **0=disable**, 1=enable*

PCSrc: 0=OutputOfALU, 1=ALUOut

*PCWr: **0=disable**, 1=enable*

Control Lines in Stage-5: lw

Stage-5: lw

(a) Write back to RegFile from MDR

RegWr: 0=disable, **1=enable**

RegDst: 0=Rd, **1=Rt**

Mem2Reg: 0=ALUOut, **1=MDR**

ALUIp1Ctl: 0=Reg1Val, 1=PC

ALUIp2Ctl2: 00=Reg2Val, 01=4, 10=SgnExtImm, 10=SgnExtImmShft2

ALUOp2: 00=add, 01=sub, 10=reg-reg

MemRd: **0=disable**, 1=enable

MemWr: **0=disable**, 1=enable

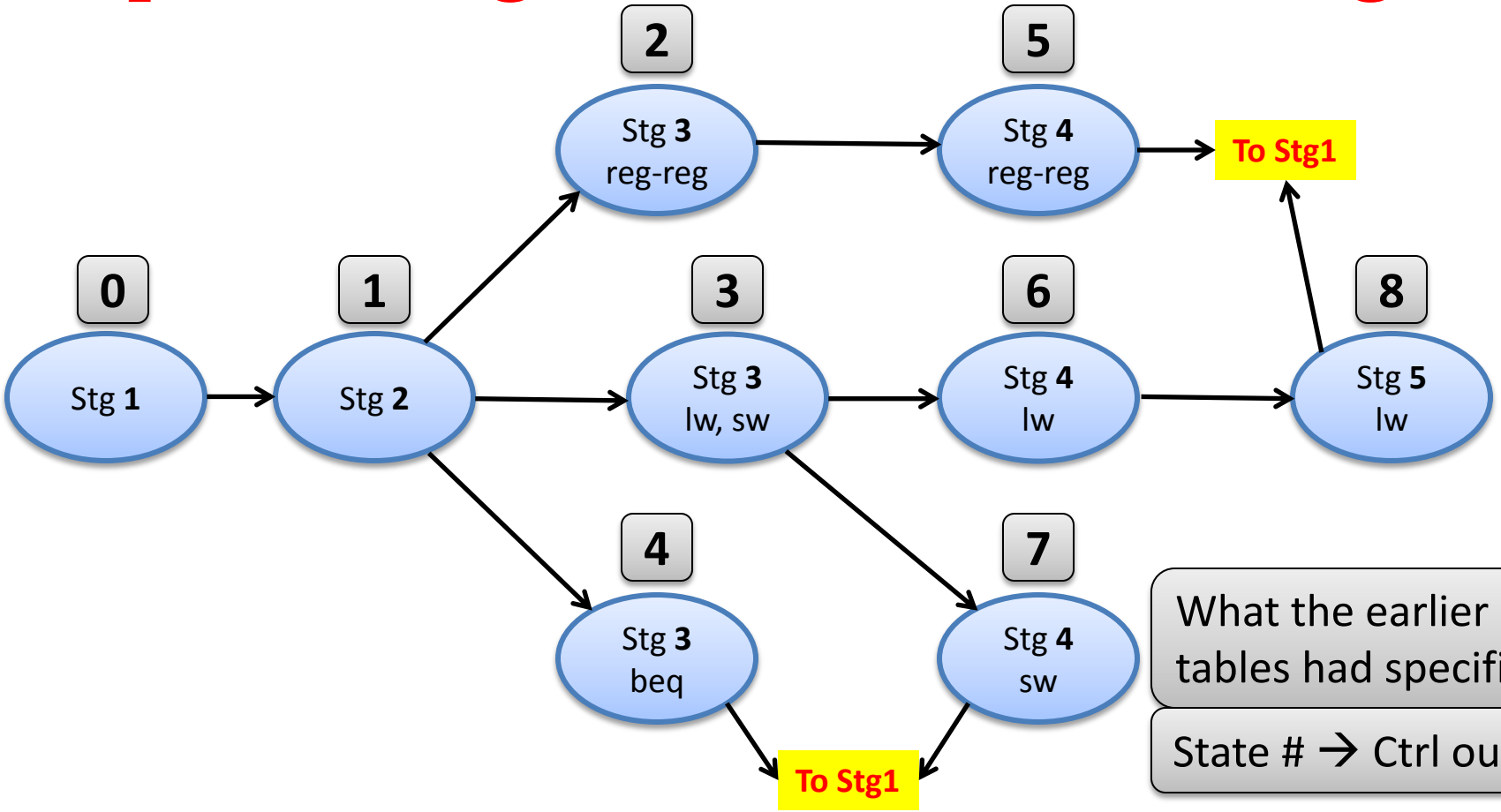
IorD: 0=PC, 1=ALUOut

*IRWr: **0=disable**, 1=enable*

PCSrc: 0=OutputOfALU, 1=ALUOut

*PCWr: **0=disable**, 1=enable*

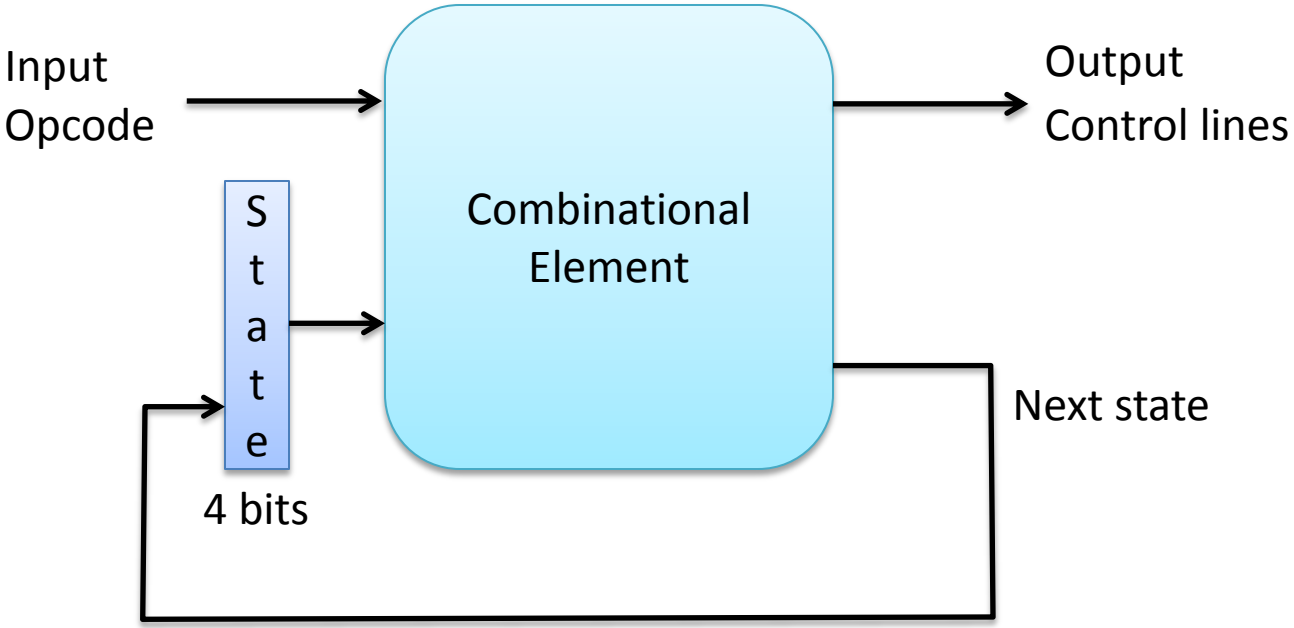
Implementing the Control: State Diagram



What the earlier tables had specified:

State # → Ctrl output

State Machine



We have: a Moore machine, but for beq

Moore machine, or Mealy machine

Moore machine: output depends on current state

Mealy machine: output depends on current state & input

In both cases: next state depends on current state and input

Summary

- Execution plan determines control lines
- Control logic: Moore or Mealy machine
  - More complex than single cycle case (combinational logic)