

Lecture 6: Designing a Single Cycle Datapath



 The Five Classic Components of a Computer

**Processor**

**Input**

**Control**

**Memory**

**Datapath**

****

**Output**

****

 Performance of a machine is determined by: ◦ Instruction count

◦ Clock cycle time

**CPI**

◦ Clock cycles per instruction

**Inst. Count Cycle Time**

 Processor design (datapath and control) will determine: ◦ Clock cycle time

◦ Clock cycles per instruction

 Single cycle processor - one clock cycle per instruction ◦ Advantages: Simple design, low CPI

◦ Disadvantages: Long cycle time, which is limited by the slowest instruction.



 Analyze instruction set => datapath requirements ◦ the meaning of each instruction is given by register transfers R[rd] <– R[rs] + R[rt];

◦ datapath must include storage element for ISA registers ◦ datapath must support each register transfer

 Select set of datapath components and establish clocking methodology

 Design datapath to meet the requirements

 Analyze implementation of each instruction to determine setting of control points that effects the register transfer.

 Design the control logic



 All MIPS instructions are 32 bits long. The three instruction

formats are:**R-type**

***6 bits* op**

***5 bits* rs**

***5 bits* rt**

***5 bits* rd**

***5 bits 6 bits* shamt funct**

**I-type**

***6 bits 5 bits 5 bits 16 bits* op rs rt immediate**

***6 bits 26 bits***

**J-type op target address**

The different fields are:

op : basic operation of the instruction (opcode) rs, rt, rd : source and destination register specifier

shamt : shift amount

funct : selects the variant of the operation in the “op” field immediate : address offset or immediate value

target address : target address of the jump instruction



 R-Type:

◦ add rd, rs, rt

◦ sub rd, rs, rt

◦ and rd, rs, rt

◦ or rd, rs, rt

◦ slt rd, rs, rt

 LOAD and STORE: ◦ lw rt, rs, imm16 ◦ sw rt, rs, imm16

 BRANCH:

◦ beq rs, rt, imm16

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

31 26 21 16 0 **op rs rt immediate**

6 bits 5 bits 5 bits 16 bits

31 26 21 16 0 **op rs rt immediate**

6 bits 5 bits 5 bits 16 bits



 RTL gives the meaning of the instructions  All instructions start by fetching the instruction

**op | rs | rt | rd | shamt | funct = MEM[ PC ]**

**op | rs | rt | Imm16 = MEM[ PC ]**

**inst Register Transfers**

**add R[rd]** 🡨 **R[rs] + R[rt]; PC** 🡨 **PC + 4 sub R[rd]** 🡨 **R[rs] – R[rt]; PC** 🡨 **PC + 4 load R[rt]** 🡨 **MEM[ R[rs] + sign\_ext(imm16)]; PC** 🡨 **PC + 4 store MEM[ R[rs] + sign\_ext(imm16) ]** 🡨 **R[rt]; PC** 🡨 **PC + 4**

**beq if ( R[rs] == R[rt] ) then PC** 🡨 **PC + 4 + sign\_ext(imm16)] || 00 else PC** 🡨 **PC + 4**

****

**** Memory

◦ instruction & data

 Registers (32 x 32)

◦ read rs

◦ read rt

◦ write rt or rd

 PC

 Sign extender

 Add and sub register or extended immediate

 Add 4 and/or shifted extended immediate to PC





 Adder

**A**

**32**

**CarryIn**

**A**

**d**

**B**

 MUX

**32**

**d**

**e**

**r**

**Select**

**32 Sum Carry**

 ALU

**32 A B32**

**M**

**U**

**X**

**OP**

**Y 32 3**

Combinational Logic: Does not use a clock

**A B**

**32 32**

**A**

**LU**

**32 Result**

**Write Enable**

 Register

◦ Similar to the D Flip Flop except

**Data In**

**Data Out**

 N-bit input and output

 Write enable input

◦ Write Enable:

 negated (0): Data Out will not change

**N N Clk**

 asserted (1): Data Out will become Data In on the falling edge of the clock



 Register File consists of

**RW RA RB**

32 registers:

◦ Two 32-bit output busses: busA and busB

◦ One 32-bit input bus: busW  Register is selected by:

**Write Enable**

**busW**

**32**

**Clk**

**5 5 5**

**32 32-bit Registers**

**busA 32**

**busB 32**

◦ RA (number) selects the register to put on busA (data) ◦ RB (number) selects the register to put on busB (data) ◦ RW (number) selects the register to be written

via busW (data) when Write Enable is 1

 Clock input (CLK)

◦ The CLK input is a factor ONLY during write operation ◦ During read operation, behaves as a combinational logic block:

 RA or RB valid => busA or busB valid after “access time.” 





 Memory

◦ One input bus: Data In

**Write Enable Data In**

**Address**

**32**

**DataOut**

◦ One output bus: Data Out  Memory word is selected by:

**32 32 Clk**

◦ Address selects the word to put on Data Out

◦ Write Enable = 1: address selects the memory

word to be written via the Data In bus

 Clock input (CLK)

◦ The CLK input is a factor ONLY during write operation ◦ During read operation, memory behaves as a combinational logic block:

 Address valid => Data Out valid after “access time.” 



 Register Transfer Requirements –> Datapath Design ◦ Instruction Fetch

◦ Decode instructions and Read Operands

◦ Execute Operation

◦ Write back the result



 The common RTL operations

◦ Fetch the Instruction: mem[PC]

◦ Update the program counter:

 Sequential Code: PC 🡨 PC + 4

 Branch and Jump: PC 🡨 “something else”

Clk **PC**

**Next Address**

**Logic**

Address

**Instruction**

**Memory**

****

Instruction Word 32



 R[rd] 🡨 R[rs] op R[rt] Example: add rd, rs, rt ◦ Ra, Rb, and Rw come from instruction’s rs, rt, and rd fields ◦ ALUctr and RegWr: control logic after decoding the 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

RegWr

busW

32

Rd Rs Rt 5 5 5

Rw Ra Rb **32 32-bit**

**Registers**

busA 32

ALUctr 3

**A**

**L**

**U**

Result

32

Clk



busB 32



 R[rt] 🡨 Mem[R[rs] + SignExt[imm16] Example: lw rt, rs, imm16 31 26 21 16 0

11

**op rs rt immediate**

RegDst

Rd Rt

**Mux**

Rs

6 bits 5 bits 5 bits 16 bits **rd**

Rt

ALUctr

busW 32

RegWr

5 5 5

Rw Ra Rb **32 32-bit**

**Registers**

busA 32

3

**A**

**L**

**U**

32

MemtoReg

Clk

busB 32

**Mu**

MemWr

**M**

**u**

**E**

**x**

**t**

**S**

**x**

Data In

WrEn

Adr

32

**x**

imm16

32

16

**en**

**ig**

**n**

32

**Data**

**d**

**e**

**r**

****

ALUSrc Clk

**Memory** MemRd

 Mem[R[rs]+SignExt[imm16]] 🡨 R[rt] Example: sw rt, rs, imm16

31 26 21 16 0

**op rs rt immediate**

6 bits 5 bits 5 bits 16 bits

RegDst

Rd

**Mux**

RegWr

Rt

Rt

Rs

ALUctr 3

MemWr

MemtoReg

busW

32

Clk

5 5 5

Rw Ra Rb **32 32-bit**

**Registers**

busA

32

busB

32

**E**

**x**

**A**

**L**

**U**

**M**

**u**

**x**

32

WrEn

Adr

**M**

**u**

**x**

imm16

**te**

**S**

**ig**

Data In

32

**Data**

32

32 16

**n**

**n**

**d**

**e**

**r**

****

ALUSrc

Clk

**Memory**

MemRd

31 26 21 16 0

**op rs rt immediate**

6 bits 5 bits 5 bits 16 bits

 beq rs, rt, imm16

◦ mem[PC] Fetch the instruction from memory ◦ Equal 🡨 R[rs] == R[rt] Calculate the branch condition

◦ if (Equal && Branch Instr.) Calculate the next instruction’s address

 PC 🡨 PC + 4 + ( SignExt(imm16) x 4 )

else

 PC 🡨 PC + 4



 beq rs, rt, imm16

31 26 21 16 0 **op rs rt immediate**

6 bits 5 bits 5 bits 16 bits

**Inst Address**

Bran ch

**4** nPC\_sel **A**

**d**

**d**

RegWr

32

Rs Rt

5 5 5 Rw Ra Rb

busA

imm16

**e**

**r**

**P**

**C**

**E**

**x**

**t**

**A**

**d**

**d**

**e**

**r**

0 **M**

**u**

**x**

1

**0 0**

**P**

**C**

Clk

busW Clk

**32 32-bit Registers**

32

busB 32

Eq.

**ALU**

**Inst**

**Memory** Adr

<

21

:

2

5>

<

16

:

2

0>

<

11

:

1

5>

<

0

:

1

5>

Instruction<31:0>

Rs Rt Rd Imm16

nPC\_sel

RegDst

Rd

Rt

**Equal**

ALUctr

MemWr

MemtoReg

**4**

RegWr

1 0

Rs

Rt

3

**A**

**d**

**d**

**e**

**r**

**A**

**M**

**u**

**x**

**0 0**

**P**

**C**

busW 32

Clk

5 5 5

Rw Ra Rb **32 32-bit**

**Registers**

busA

busB 32

32

0

**M**

**u**

**=**

**A**

**L**

**U**

32

0

**M**

**PC**

**dd**

**e**

**r**

Clk

**E**

**x**

**t**

**Si**

**x** 1

Data InWrEn 32 Adr **Data**

**ux**

1

imm16

32 16

6 1

m

m

i

**E**

**x**

**t**

**r**

**e**

**n**

**de**

**gn**

ALUSrc

Clk

**Memory** MemRd



0

M

u

x

Add ALU

1

result

Add Shift

4

Instruction [31-26]

Control

RegDst

Branch

MemRead MemtoReg ALUOp

MemWrite ALUSrc

RegWrite

left 2

PC

Read 

Address

Instruction

[31-0]

Instruction [25-21] Instruction [20-16]

0

M

Read reg 1

Read reg 2

Write

Read

data 1

Registers

Read

ALU

zero

ALU

result

Address

Read

1

Instruction memory

Instruction [15-11]

u x 1

register

Write

data

data 2

0

M u x 1

Write data

data 

Data

memory

M u x 0

Instruction [15-0] 

16 32

Sign

extend

Instruction [5-0]

ALU control

Instruction<31:0>

**Inst**

**Memory** Adr

Op

<

26

:

31

>

<

0

:

5

>

Fun

**Control**

nPC\_sel

RegDst ALUSrc ALUctr MemWr MemtoReg **Equal** RegWr MemRd

**DATA PATH**

****

**** Rs, Rt, Rd and Imm16 hardwired into datapath  nPC\_sel: 0 => PC 🡨 PC + 4;

1 => PC 🡨 PC + 4 + SignExt(Im16) || 00 nPC\_sel

**4**

**A**

**d**

6 1

m

m

i

**PC**

**E**

**x**

**t**

**d**

**e**

**r**

**A**

**dd**

**e**

**r**

0

**M**

**u**

**x**

1

**0**

**0**

**P**

**C**

Clk

**Inst**

**Memory** Adr



 **ALUsrc: 0 => regB; 1 => immed**  **MemRd: read memory**

 **MemWr: write memory**

■ **RegWr: write dest register** ■ **MemtoReg: 0 => ALU; 1 => Mem** ■ **RegDst: 0 => “rt”; 1 => “rd”**

 **ALUctr: “add”, “sub”, “and”, “or”, “set less than”**

RegDst

Rd

Rt

**Equal**

ALUctr

MemWr

MemtoReg

RegWr

1 0

Rs

Rt

3

busW 32

Clk

5 5 5

Rw Ra Rb **32 32-bit**

**Registers**

busA

busB 32

**=**

32

0

**M**

**u**

**A**

**LU**

32

0

**Mu**

**E**

**x**

**te**

**Si**

**x** 1

Data InWrEn 32 Adr **Data**

**x** 1

imm16

ALUSrc

32 16

**g**

**n**

**n**

**d**

**e**

**r**

Clk

**Memory** MemRd

**Inst Memory**

Adr

<

21

:

2

5>

<

16

:

2

0>

<

11

:

1

5>

<

0

:

1

5>

Instruction<31:0>

Rs Rt Rd Imm16

nPC\_sel

RegDst

Rd

Rt

**Equal**

ALUctr

MemWr

MemtoReg

**+4 rt add** 1 0

**4**

**A**

**d**

RegWr

Rs

5 5 5

Rt

busA

**d**

**e**

**r**

**A**

0 **M**

**u**

**x**

1

**0 0**

**P**

**C**

busW 32

Clk

Rw Ra Rb **32 32-bit**

**Registers**

32

busB

32

**=**

0

**M**

**u**

**A**

**LU**

32

0

**M**

**PC**

**dd**

**e**

**r**

Clk

imm16

**E**

**x**

**te**

**n**

**S**

**ig**

**x** 1

Data InWrEn 32 Adr **Data**

**ux**

1

6 1

m

m

i

**E**

**x**

**t**

32 16

**n**

**d**

**e**

**r**

**ext**

ALUSrc

Clk

**Memory**

MemRd



**Ideal**

**Instruction Memory**

Instruction

**Control**

**Control Signals Conditions**

Instruction Address

Rd 5

Rs 5

Rt

5

A

Data

**s s**

**e**

**r**

**d**

**d**

**A**

**t**

**x**

**e**

**N**

**C**

**P**

k

l

C

Rw Ra Rb

32 32 **32 32-bit**

**Registers**

B

Clk

32

**A**

**L**

**U**

32

Address **Ideal Data**

Data

**Memory**

In

Clk

Data Out

**Datapath**

****

**** 5 steps to design a processor

◦ 1. Analyze instruction set => datapath requirements ◦ 2. Select set of datapath components & establish clock methodology

◦ 3. Design datapath meeting the requirements

◦ 4. Analyze implementation of each instruction to determine setting of control points that effects the register transfer. ◦ 5. Design the control logic

 MIPS makes it easier

◦ Instructions same size

◦ Source registers always in same place

◦ Immediates same size, location

◦ Operations always on registers/immediates

 Single cycle datapath => CPI=1, CCT => long