

Control & DataPath Part 1

Ref: Chapter 5

Datapath

- The *datapath* is the interconnection of the components that make up the processor.
- The datapath must provide connections for moving bits between memory, registers and the ALU.

Control

- The control is a collection of signals that enable/disable the inputs/outputs of the various components.
- You can think of the control as the brain, and the datapath as the body.
 - the datapath does only what the brain tells it to do.

Processor Design

The *sequencing* and *execution* of instructions

- We already know about many of the individual components that are necessary:
 - ALU, Multiplexors, Decoders, Flip-Flops
- We need to discuss how to use a *clock*
- We need to think about registers and memory.

CompOrg Fall 2000 - Control & DataPath Part 1

4

The Clock

The *clock* generates a never-ending sequence of alternating 1s and 0s.



All operations are synchronized to the clock.

CompOrg Fall 2000 - Control & DataPath Part 1

5

Clocking Methodology

- Determines when (relative to the clock) a *signal* can be read and written.
 - Read: signal value is used by some component.
 - Written: a signal value is generated by some component.

CompOrg Fall 2000 - Control & DataPath Part 1

6

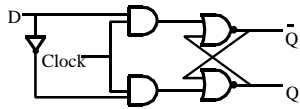
Simple Example: Enabled AND

- We want an AND gate that holds its output value constant until the clock switches from 0 (lo) to 1 (hi).
- We can use a flip-flop to hold the inputs to the AND gate constant during the time we want the output constant.
- We use a clocked flip-flop to make things happen when the clock changes.

CompOrg Fall 2000 - Control & DataPath Part 1

7

D Flip-Flop Reminder

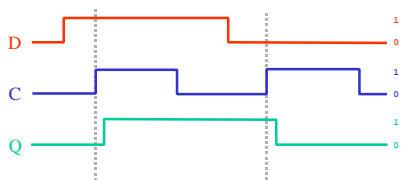


The output (Q) changes to reflect D only when the Clock is a 1.

CompOrg Fall 2000 - Control & DataPath Part 1

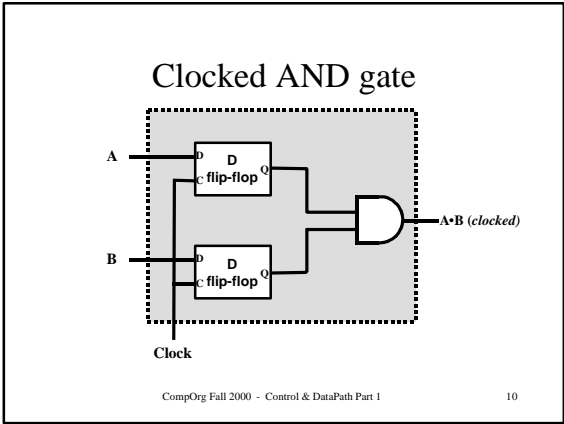
8

D Flip-Flop Timing



CompOrg Fall 2000 - Control & DataPath Part 1

9



Edge-triggered Clocking

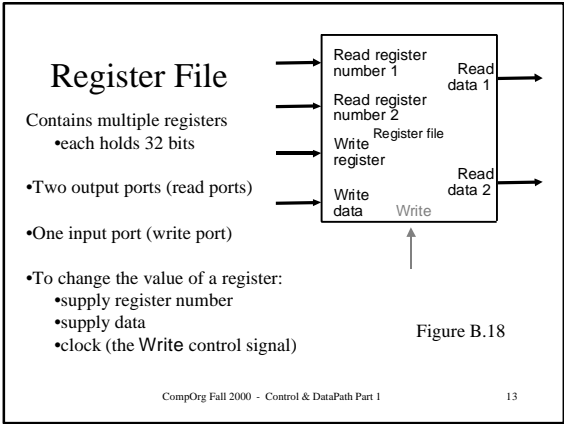
- Values stored are updated (can change) only on a clock edge.
 - When the clock switches from 0 to 1 everybody allows signals in.
 - *everybody* means *state elements*
 - combinational elements always do the same thing, they don't care about the clock (that's why we added the flip-flops to our AND gate).

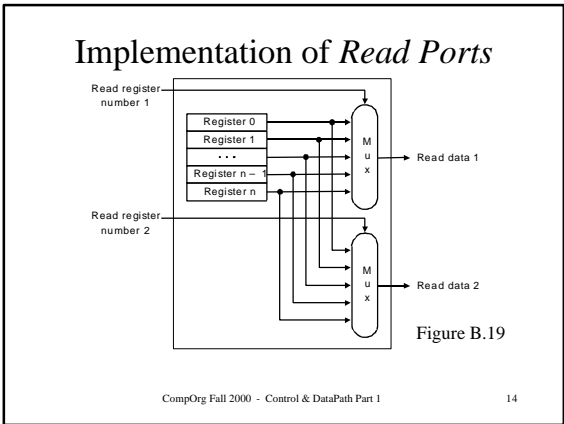
CompOrg Fall 2000 - Control & DataPath Part 1 11

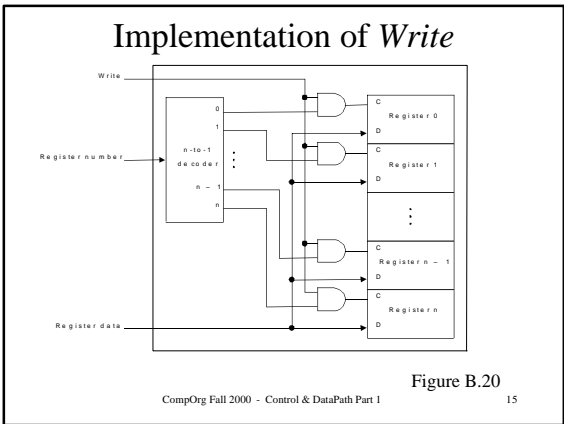
State Elements

- Any component that *stores* one or more values is a *state element*.
 - The entire processor can be viewed as a circuit that moves from one state (collection of all the state elements) to another state.
 - At time i a component uses values generated at time $i-1$.

CompOrg Fall 2000 - Control & DataPath Part 1 12



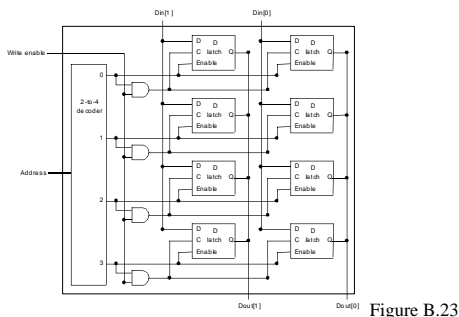




Memory

- Memory is similar to a very large register file:
 - single read port (output)
 - *chip select* input signal
 - *output enable* input signal
 - *write enable* input signal
 - address lines (determine which memory element)
 - data input lines (used to write a memory element)

4 x 2 Memory (SRAM)



Memory Usage

- For now, we treat memory as a single component that supports 2 operations:
 - write (we change the value stored in a memory location)
 - read (we get the value currently stored in a memory location).
- We can only do one operation at a time!

Instruction & Data Memory

- It is useful to treat the memory that holds instructions as a separate component.
 - instruction memory is *read-only*
- Typically there is really one memory that holds both instructions and data.
 - as we will see when we talk more about memory, the processor often has two interfaces to the memory, one for instructions and one for data!

CompOrg Fall 2000 - Control & DataPath Part 1

19

Designing a Datapath for MIPS

- We start by looking at the datapaths needed to support a simple subset of MIPS instructions:
 - a few arithmetic and logical instructions
 - load and store word
 - **beq** and **j** instructions

CompOrg Fall 2000 - Control & DataPath Part 1

20

Functions needed for MIPS Instructions

- We can generalize the functions we need to:
 - using the PC register as the address, read a value from the memory (read the instruction)
 - Read one or two register values (depends on the specific instruction).
 - ALU Operation, Memory read or write, ...
 - Possibly change the value of a register.

CompOrg Fall 2000 - Control & DataPath Part 1

21

Fetching the next instruction

- PC Register holds the address
- Memory holds the instruction
 - we need to *read* from memory.
- Need to update the PC
 - add 4 to current value

Instruction Fetch DataPath

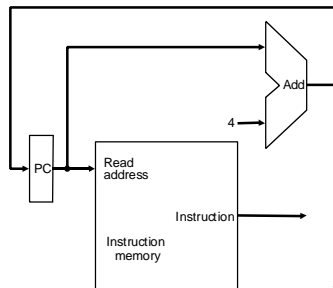
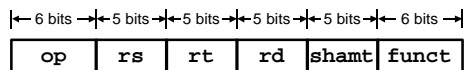


Figure 5.5

Supporting R-format instructions



Includes **add**, **sub**, **slt**, **and** & **or** instructions.

Generalization:

- read 2 registers and send to ALU.
- perform ALU operation
- store result in a register

MIPS Registers

- MIPS has 32 general purpose registers.
- Register File holds all 32 registers
 - need 5 bits to select a register
 - **rs**, **rt** & **rd** fields in R-format instructions.
- MIPS Register File has 2 *read ports*.
 - can get at both *source* registers at the same time.

Datapath for R-format Instructions

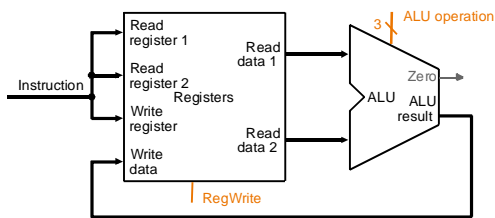


Figure 5.7

Load and Store Instructions

- Need to compute the address
- offset (part of the instruction)
 - base (stored in a register).

For Load:

- read from memory
- store in a register

For Store:

- read from register
- write to memory

Computing the address

- 16 bit signed offset is part of the instruction.
- We have a 32 bit ALU.
 - need to sign extend the offset (to 32 bits).
- Feed the 32 bit offset and the contents of a register to the ALU
- Tell the ALU to “add”.

Load/Store Datapath

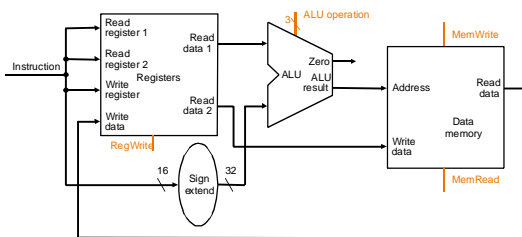


Figure 5.9

Supporting `beq`

- 2 registers compared for equality
- 16 bit offset used to compute target address.
 - signed offset is relative to the PC
 - offset is in *words* not in bytes!
- Might branch, might not (need to decide).

Computing target address

- Recall that the offset is actually relative to the address of the next instruction.
 - we *always* add 4 to the PC, we must make sure we use this value as the base.
- Word vs. Byte offset
 - we just need to shift the 16 bit offset 2 bits to the right (fill with 2 zeros).

CompOrg Fall 2000 - Control & DataPath Part 1

31

Branch Datapath

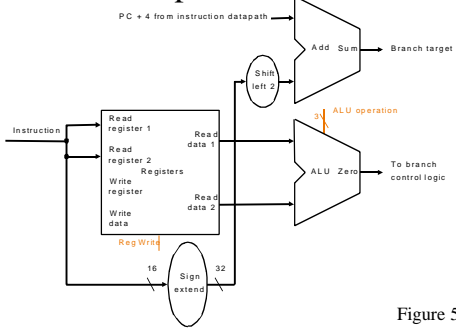


Figure 5.10

CompOrg Fall 2000 - Control & DataPath Part 1

32
