

Chapter 5 The Processor: Datapath and Control

©2004 Morgan Kaufmann Publishers 1

Introduction

- Performance determined by
 - # of instructions
 - Cycle per instruction (CPI)
 - Clock time
- Implementation of Processor?
- Implementation of processor
 - Datapath and control
 - Simple version: MIPS instruction set
 - Complex version: IA-32 instruction set

©2004 Morgan Kaufmann Publishers 2

MIPS Implementation

- Key principle
 - Create datapath
 - Create control
- Core MIPS instruction set
 - Arithmetic—logical instructions: **add sub and or slt**
 - Memory reference instructions: **lw sw**
 - Branch and jump: **beq j**
- Basic idea can be applied to other instructions/systems
 - Integer instructions: shift, multiply, and divide
 - Floating-point instructions
 - Other computer systems

©2004 Morgan Kaufmann Publishers 3

The Processor: Datapath & Control

- Generic Implementation
 - Use the program counter (PC) to supply instruction address
 - Get the instruction from memory
 - Read registers
 - Use the instruction to decide exactly what to do
- All instructions use the ALU after reading the registers
 - Why?
 - memory-reference?
 - arithmetic?
 - control flow?

©2004 Morgan Kaufmann Publishers 4

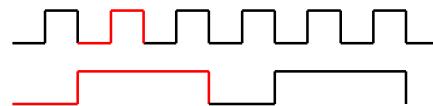
Logic Design Conventions

- Combinational logic
 - Operate on data values
 - No internal storage
 - Lose power?
 - Outputs always depends on inputs
 - Same inputs → Same outputs
 - e.g. Arithmetic logic Unit (ALU)
- Sequential logic
 - Operate on **state elements**
 - Internal storage – contain **state**
 - Lose power?
 - Outputs depends on inputs and contents of internal states
 - Can be read at any time
 - e.g. Registers and memories

©2004 Morgan Kaufmann Publishers 5

Clocking Methodology

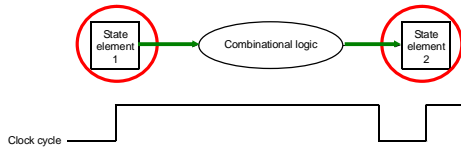
- Signal asserted
 - Mean signal is logical high
- Signal deasserted
 - Mean signal is logical low
- Edge-triggered clocking
 - Any values stored in sequential logic can be updated only on a clock edge



©2004 Morgan Kaufmann Publishers 6

Clocking Methodology

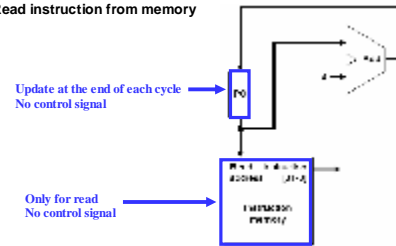
- Only state elements can store a value
- Typical execution:
 - read contents of some state elements,
 - send values through some combinational logic
 - write results to one or more state elements
 - **Must be done in a single cycle** → Decide clock cycle



©2004 Morgan Kaufmann Publishers 7

Building a Datapath

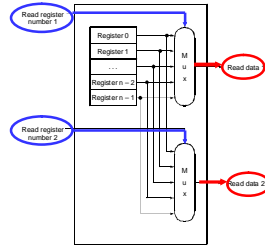
- Fetch instruction
 - Instruction address ← Program counter (PC)
 - Next sequential instruction address ← PC+4
 - Read instruction from memory



©2004 Morgan Kaufmann Publishers 8

Building a Datapath

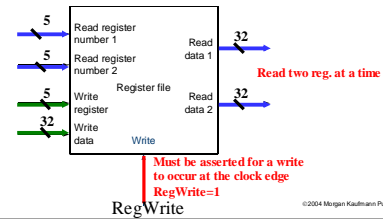
- Register file
 - Collection of registers
 - Registers can be read or written by specifying register #
 - Contain register state
 - Read two registers at a time



©2004 Morgan Kaufmann Publishers 9

Building a Datapath

- Arithmetic-logical instruction (R-type)
 - Add sub and or slt
 - Typical instruction: add \$t1, \$t2, \$t3
 - Read: \$t2 and \$t3
 - Write: \$t1
 - Need register file and ALU



©2004 Morgan Kaufmann Publishers 10

Building a Datapath

- Arithmetic-logical instruction (R-type)
 - Add sub and or slt
 - ALU function
 - ALUOp: 4-bit control signal (6 input combinations are used)

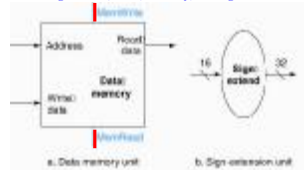
ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	Set less than
1100	NOR



©2004 Morgan Kaufmann Publishers 11

Building a Datapath

- Memory reference instruction (I-type)
 - lw and sw
 - Typical instruction:
 - lw \$t1, offset_value(\$t2) or sw \$t1, offset_value(\$t2)
 - Load: Read \$t2; calculate address; read memory
 - Store: Read \$t1 and \$t2; calculate address; write memory
 - Computer memory address: \$t2- 16-bit signed offset
 - Need register file, ALU, memory, and sign extension



©2004 Morgan Kaufmann Publishers 12

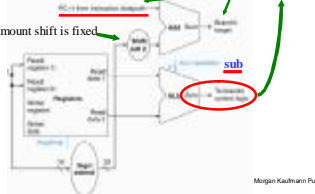
Building a Datapath

- Branch instruction

- beq
- Typical instruction:
 - Beq \$t1, \$t2, offset
 - Read \$t1 and \$t2; compare whether equal (branch taken/branch not taken)
 - Calculate branch address (word address)
 - 16-bit offset: 16-bit sign-extension, shifted left 2 bits
 - PC+4+offset

Not actual shift hardware since amount shift is fixed
Add 00 to lower order end

instruction



Morgan Kaufmann Publishers 13

Building a Datapath

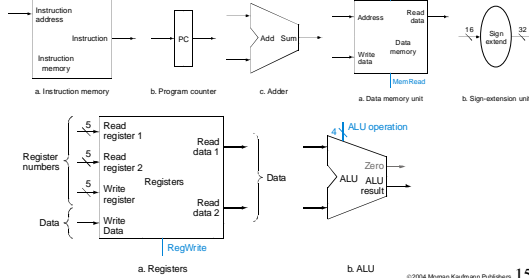
- Jump instruction

- j
- Typical instruction:
 - j target-address
 - Calculate jump address (word address)
 - 26-bit offset: shifted left 2 bits (add 00 to lower order end)
 - PC+4+offset

©2004 Morgan Kaufmann Publishers 14

Simple Implementation

- Include the functional units we need for each instruction



©2004 Morgan Kaufmann Publishers 15

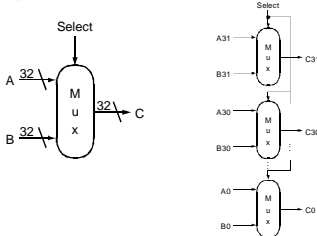
Creating a Single Datapath

- We have examined datapath components
- Create a single datapath
 - How do we put components together with control?
 - Many components shared by different instructions
 - Might need multiple connections to input of elements
 - Use multiplexor and control signal to select among multiple inputs
- Main components
 - PC
 - Memory
 - Register file
 - ALU
 - Sign-extension
 - Control unit

©2004 Morgan Kaufmann Publishers 16

Multiplexor

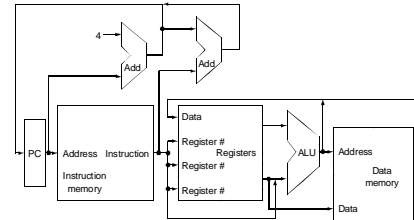
- Make sure you understand the abstractions!
- Sometimes it is easy to think you do, when you don't



©2004 Morgan Kaufmann Publishers 17

More Implementation Details

- Abstract / Simplified View:

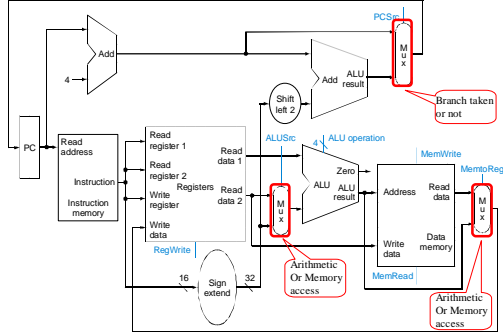


Two types of functional units:

- elements that operate on data values (combinational)
- elements that contain state (sequential)

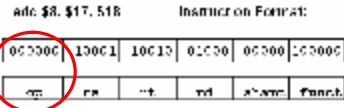
©2004 Morgan Kaufmann Publishers 18

Building the Datapath – use multiplexor



Control

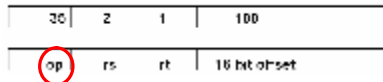
- Selecting the operations to perform (ALU, read/write, etc.)
- Controlling the flow of data (multiplexor inputs)
- Information comes from the 32 bits of the instruction
- Example:



Control

- ALU's operation based on instruction type and function code
- e.g., what should the ALU do with this instruction

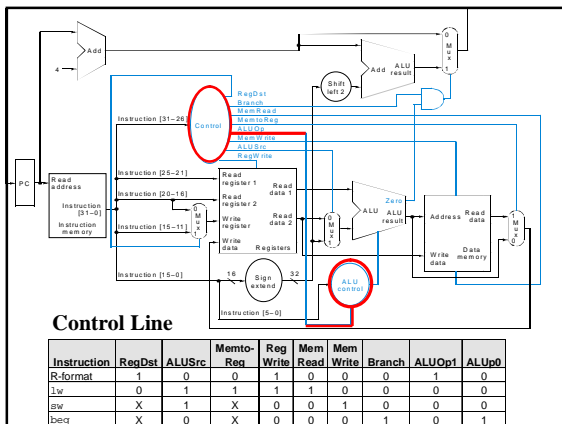
Example: lw \$1, 100(\$2)



ALU Control

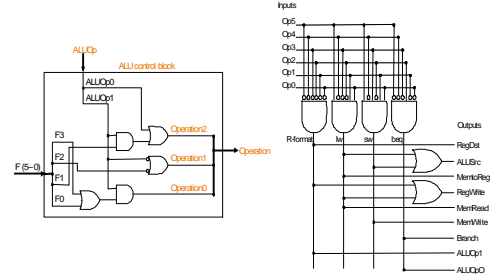
- Must describe hardware to compute 4-bit ALU control input
 - given instruction type
 - 00 = lw, sw
 - 01 = beq
 - 10 = arithmetic
 - function code for arithmetic
- Describe it using a truth table (can turn into gates):

Lw/sw beq add subtract and or slt	ALUOp		Function Code						Operation	
	ALUOp4	ALUOp3	F6	F5	F4	F3	F2	F1		
00	0	0	0	0	0	0	0	0	0	MEM
01	0	1	0	0	0	0	0	0	0	MEM
10	0	0	0	0	0	0	0	0	0	MEM
11	0	0	0	0	0	0	0	0	0	MEM
10	0	0	0	0	0	0	0	0	0	MEM
11	0	0	0	0	0	0	0	0	0	MEM
10	0	0	0	0	0	0	0	0	0	MEM
11	0	0	0	0	0	0	0	0	0	MEM



Control

- Simple combinational logic (truth tables)



Datapath for Typical Instructions

- **Opcode**
 - Op[5:0]
 - Always bits 31:26
- **Register rs and rt fields**
 - Always bits 25:21 and 20:16 for **R-type, branch equal**
- **Base register**
 - Always 25:21 (rs) for **load and store**
- **16-bit offset**
 - Always bits 15:0 for **branch equal, load, and store**
- **Destination register**
 - Always bits 15:11 (rd) for **R-type instruction**
 - Always bits 20:16 for **load**

©2004 Morgan Kaufmann Publishers 25

Datapath for R-Type Instructions

- **add \$t1, \$t2, \$t3**
 - 1. Fetch instruction & update PC
 - 2. Read \$t2 and \$t3 & control unit computes setting of control lines
 - 3. ALU operates on data read from register file using function code (bit 5:0)
 - 4. Results from ALU written into register file \$t1
- [Figure 519.pdf](#)

©2004 Morgan Kaufmann Publishers 26

Datapath for I-Type Instructions

- **lw \$t1, offset(\$t2)**
 - 1. Fetch instruction & update PC
 - 2. Read \$t2 & control unit computes setting of control lines
 - 3. ALU addition
 - Sum = Data read from register file + sign-extended 16-bit offset
 - 4. Sum used to access data memory
 - 5. Data read from memory is written into register file
 - Register # give by bits 20:16 of the instruction
- [Figure 520.pdf](#)

©2004 Morgan Kaufmann Publishers 27

Datapath for Branch instruction

- **beq \$t1, \$t2, offset**
 - 1. Fetch instruction & update PC
 - 2. Read \$t1 & \$t2 & control unit computes setting of control lines
 - 3. Branch target address
 - PC + 4 + sign-extended 16-bit offset shifted left by two
 - ALU subtract
 - \$t1 - \$t2
 - 4. ALU zero result used to decide branch or not
- [Figure 521.pdf](#)

©2004 Morgan Kaufmann Publishers 28

Datapath for Jump instruction

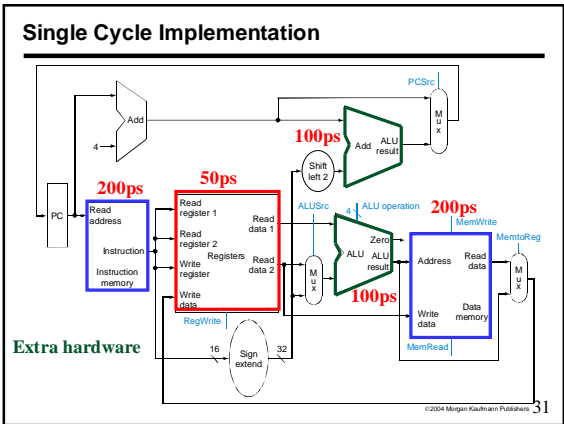
- **Jump instruction**
 - 1. Fetch instruction & update PC
 - 2. Upper 4 bits of PC+4 concatenated with
 - 3. 26-bit immediate field of jump instruction left shifted by 2
- [Figure 524.pdf](#)

©2004 Morgan Kaufmann Publishers 29

Single Cycle Implementation

- **Calculate cycle time assuming negligible delays except:**
 - memory (200ps)
 - ALU and adders (100ps)
 - Register file access (50ps)

©2004 Morgan Kaufmann Publishers 30



Worst case – cycle time

- Assume
 - memory (200ps)
 - ALU and adders (100ps)
 - Register file access (50ps)

Cycle time
– worst case of instruction execution

Load	Add
Fetch Instr. (200ps)	Fetch Instr. (200ps)
Read register (50ps)	Read register (50ps)
ALU – address (100ps)	ALU – addition (100ps)
Read data (200ps)	Write register (50ps)
Write register (50ps)	
Total: 600ps	Total: 400ps

©2004 Morgan Kaufmann Publishers 32

Where we are headed

- Single Cycle Problems:
 - what if we had a more complicated instruction?
 - wasteful of area
- One Solution:
 - use a “smaller” cycle time
 - have different instructions take different numbers of cycles
 - a “multicycle” datapath:

©2004 Morgan Kaufmann Publishers 33

Multicycle Approach

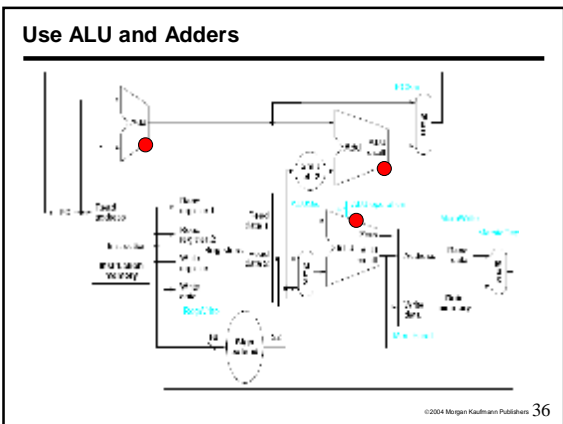
- Reuse functional units
 - ALU used to compute address and to increment PC
 - Memory used for instruction and data
- Our control signals will not be determined directly by instruction
 - e.g., what should the ALU do for a “subtract” instruction?

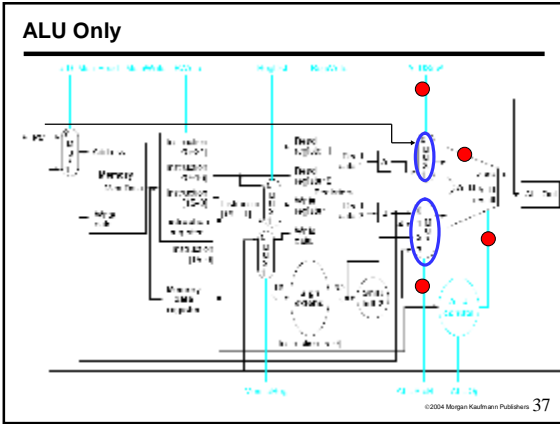
©2004 Morgan Kaufmann Publishers 34

ALU and adders

- Single cycle datapath
 - ALU and two adders
- ALU
 - Operation on two registers for arithmetic instructions
 - Add a register to a sign-extended constant to compute effective addresses for lw and sw instructions
- One adder
 - Compute PC + 4
- One adder
 - Comput branch targets
 - Add a sign-extended shifted offset to (PC + 4)

©2004 Morgan Kaufmann Publishers 35

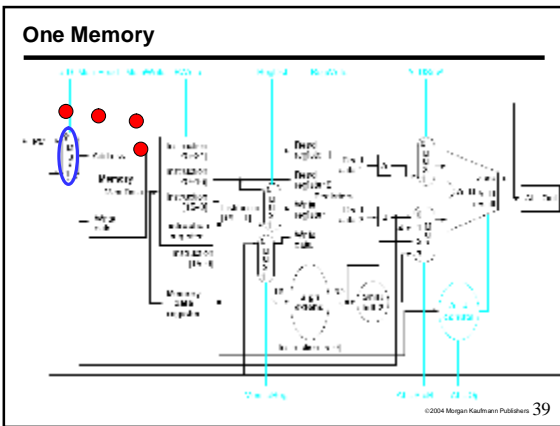




ALU Only

- 2-to-1 mux ALUSrcA
 - First ALU input
 - PC
 - register
- 4-to-1 mux ALUSrcB
 - Second ALU input
 - Register file
 - Constant 4
 - Sign-extended constant
 - Sign-extended and shifted constant
- Single ALU
 - Arithmetic/logical operations
 - PC+4
 - Memory address for lw/sw
 - Branch address

©2004 Morgan Kaufmann Publishers 38



Multicycle Approach

- Single data path
 - Fetch instruction & PC increment
 - Access register
 - ALU operation
 - Access memory
 - Access register
- Break up the instructions into steps, each step takes a cycle
 - balance the amount of work to be done
 - restrict each cycle to use only one major functional unit
 - Memory
 - Register
 - ALU

©2004 Morgan Kaufmann Publishers 40

Multicycle Approach

- At the end of a cycle
 - store values for use in later cycles (easiest thing to do)
 - introduce additional "internal" registers
 - Instruction register, IR
 - Hold instruction read from memory
 - Memory data register, MDR
 - Hold data read from memory
 - A and B register
 - Hold values read from register file
 - ALUOut register
 - Hold output of ALU

©2004 Morgan Kaufmann Publishers 41

Instructions from ISA perspective

- Consider each instruction from perspective of ISA.
- Example: add
 - The add instruction changes a register.
 - Register specified by bits 15:11 of instruction.
 - Instruction specified by the PC.
 - New value is the sum ("op") of two registers.
 - Registers specified by bits 25:21 and 20:16 of the instruction
$$\text{Reg[Memory[PC][15:11]]} \leftarrow \text{Reg[Memory[PC][25:21]] op Reg[Memory[PC][20:16]}$$
 - In order to accomplish this we must break up the instruction. (kind of like introducing variables when programming)

©2004 Morgan Kaufmann Publishers 42

Breaking down an instruction

- ISA definition of arithmetic:
`Reg[Memory[PC][15:11]] <=`
`Reg[Memory[PC][25:21]] op`
`Reg[Memory[PC][20:16]]`
- Could break down to:
 - `IR <= Memory[PC]`
 - `A <= Reg[IR[25:21]]`
 - `B <= Reg[IR[20:16]]`
 - `ALUOut <= A op B`
 - `Reg[IR[20:16]] <= ALUOut`
- We forgot an important part of the definition of arithmetic!
 - `PC <= PC + 4`

©2004 Morgan Kaufmann Publishers 43

Idea behind multicycle approach

- We define each instruction from the ISA perspective (do this!)
- Break it down into steps following our rule that data flows through at most one major functional unit (e.g., balance work across steps)
- Introduce new registers as needed (e.g., A, B, ALUOut, MDR, etc.)
- Finally try and pack as much work into each step (avoid unnecessary cycles) while also trying to share steps where possible (minimizes control, helps to simplify solution)

Result: Multicycle Implementation!

©2004 Morgan Kaufmann Publishers 44

Five Execution Steps

- Instruction Fetch
- Instruction Decode and Register Fetch
- Execution, Memory Address Computation, or Branch Completion
- Memory Access or R-type instruction completion
- Write-back step
INSTRUCTIONS TAKE FROM 3 - 5 CYCLES!

©2004 Morgan Kaufmann Publishers 45

Step 1: Instruction Fetch

- Use PC to get instruction and put it in the Instruction Register.
- Increment the PC by 4 and put the result back in the PC.
- Can be described succinctly using RTL "Register-Transfer Language"

```
IR <= Memory[PC];  
PC <= PC + 4;
```

©2004 Morgan Kaufmann Publishers 46

Step 2: Instruction Decode and Register Fetch

- Read registers rs and rt in case we need them
- Compute the branch address in case the instruction is a branch
- ```
A <= Reg[IR[25:21]];
B <= Reg[IR[20:16]];
ALUOut <= PC + (sign-extend(IR[15:0])
<< 2);
```

©2004 Morgan Kaufmann Publishers 47

## Step 3 (instruction dependent)

- ALU is performing one of three functions, based on instruction type
- Memory Reference:

```
ALUOut <= A + sign-extend(IR[15:0]);
```
- R-type:

```
ALUOut <= A op B;
```
- Branch:

```
if (A==B) PC <= ALUOut;
```

©2004 Morgan Kaufmann Publishers 48

### Step 4 (R-type or memory-access)

- Loads and stores access memory  
 $MDR \leftarrow Memory[ALUOut];$   
 OR  
 $Memory[ALUOut] \leftarrow B;$
- R-type instructions finish  
 $Reg[IR[15:11]] \leftarrow ALUOut;$

*The write actually takes place at the end of the cycle on the edge*

### Write-back step

- $Reg[IR[20:16]] \leftarrow MDR;$

### Summary:

| Step                 | Function                           | Control Signals            | Write Enable           | Write to Register |
|----------------------|------------------------------------|----------------------------|------------------------|-------------------|
| Instruction Memory   |                                    | $PC \leftarrow PC + 4$     |                        |                   |
| Instruction Register |                                    | $PC \leftarrow PC + 4$     |                        |                   |
| ALU                  | $ALUOut \leftarrow A \pm B$        | $ALUOp \leftarrow IR[6:5]$ |                        |                   |
| Memory               | $MDR \leftarrow Memory[ALUOut]$    | $MEMOp \leftarrow IR[6:5]$ |                        |                   |
| Register File        | $Reg[IR[15:11]] \leftarrow ALUOut$ |                            | $WE \leftarrow IR[15]$ | $Reg[IR[15:11]]$  |
| Register File        | $Reg[IR[20:16]] \leftarrow MDR$    |                            | $WE \leftarrow IR[20]$ | $Reg[IR[20:16]]$  |

FIGURE 5.11 Summary of the 5-stage pipeline. The pipeline is divided into five stages: Instruction Memory, Instruction Register, ALU, Memory, and Register File. Each stage performs a specific function and is controlled by signals from the instruction register. The Register File stage is divided into two parts: one for R-type instructions and one for memory-access instructions.