Processor: Datapath and Control

• Single cycle processor
  – Datapath and Control
• Multicycle processor
  – Datapath and Control
• Microprogramming
  – Vertical and Horizontal Microcodes

Processor Design

• Processor design
  – datapath and control unit design
    » clock cycle time
    » clock cycles per instruction
• Performance of a machine is determined by
  – Instruction count
  – clock cycle time
  – clock cycles per instruction

Review: THE Performance Equation

• Our basic performance equation is then
  \[ \text{CPU time} = \frac{\text{Instruction count} \times \text{CPI} \times \text{clock cycle}}{\text{clock rate}} \]
  or
  \[ \text{CPU time} = \frac{\text{Instruction count} \times \text{CPI}}{\text{clock rate}} \]

• These equations separate the three key factors that affect performance
  – Can measure the CPU execution time by running the program
  – The clock rate is usually given in the documentation
  – Can measure instruction count by using profilers/simulators without knowing all of the implementation details
  – CPI varies by instruction type and ISA implementation for which we must know the implementation details

How to Design a Processor: step-by-step

1. Analyze instruction set => datapath requirements
   the meaning of each instruction is given by the register transfers
   datapath must include storage element for ISA registers possibly more
   datapath must support each register transfer
2. Select set of datapath components and establish clocking methodology
3. Assemble datapath meeting the requirements
4. Analyze implementation of each instruction to determine setting of control points that effects the register transfer.
5. Assemble the control logic
Ch5 

Single Cycle Processor

- Single cycle processor
  - Pros: one clock cycle per instruction
  - Cons: too long cycle time, too low clocking frequency
- Design a processor
  - analyze instruction set (the meaning of each instruction is given by the register transfers)
  - timing of each instruction
  - datapath support each register transfer
  - select datapath components and establish clocking methodology
  - analyze implementation of each instruction to determine setting of control points that affect register transfer
  - assemble control logic and datapath components

Ch5 

Clocking Methodology

- Edge-triggered clock
- setup time
- hold time
- all storage elements clocked by the same clock
- combinational logic block:
  - inputs are updated at each clock tick
  - all outputs must be stable before the next clock tick

Ch5 

Clocked Logic Timing

What is the smallest T that produces correct operation?

Worst case CL delay limits T.
Flip Flop delays:

- **clk-to-Q?** setup? hold?

CLK == 0
Sense D, but Q outputs old value.

CLK 0->1
Capture D, pass value to Q

Flip Flops have internal delays

Value of D is sampled on **positive clock edge**.
Q outputs sampled value for rest of cycle.

- **t_setup**
- **t_clk-to-Q**

Combinational Logic

ALU "time budget"

\[ T \geq \tau_{\text{clk-Q}} + \tau_{\text{CL}} + \tau_{\text{setup}} \]

\[ T \geq T_{\text{CLK}} + T_{\text{setup}} + T_{\text{clk-Q}} + \text{worst case skew.} \]
D must stay stable here

What is the intended function of this circuit?

Does flip-flop hold time affect operation of this circuit?

Under what conditions?

\[ t_{\text{clk-to-Q}} + t_{\text{inv}} > t_{\text{hold}} \]

---

**The MIPS Instruction Formats**

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

  - **R-type**
    
    | 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 |

  - **I-type**
    
    | 31 | 26 | 21 | 16 | 11 | 6 | 0 |
    |----|----|----|----|----|----|----|
    | op | rs | rt | immediate |
    
    | 6 bits | 5 bits | 5 bits | 16 bits |

  - **J-type**
    
    | 31 | 26 | 0 |
    |----|----|---|
    | op | target address |
    
    | 6 bits | 26 bits |

- The different fields are:
  - op: operation of the instruction
  - rs, rt, rd: the source and destination register specifiers
  - 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

---

**Register Transfers**

- **add** $1$, $2$, $3$; $rs = 2$, $rt = 3$, $rd = 1$
  
  \[ R[rd] \leftarrow R[rs] + R[rt] \]
  \[ PC \leftarrow PC + 4 \]

- **sub** $1$, $2$, $3$; $rs = 2$, $rt = 3$, $rd = 1$
  
  \[ R[rd] \leftarrow R[rs] - R[rt] \]
  \[ PC \leftarrow PC + 4 \]

- **ori** $1$, $2$, $20$; $rs = 2$, $rt = 1$
  
  \[ R[rt] \leftarrow R[rs] + \text{zero_ext}(\text{imm16}) \]
  \[ PC \leftarrow PC + 4 \]

- **lw** $1$, $200($2$)$; $rs = 2$, $rt = 1$
  
  \[ R[rt] \leftarrow \text{MEM}(R[rs] + \text{sign_ext}(\text{imm16})) \]
  \[ PC \leftarrow PC + 4 \]

- **sw** $1$, $200($2$)$; $rs = 2$, $rt = 1$
  
  \[ \text{MEM}(R[rs] + \text{sign_ext}(\text{imm16})) \leftarrow R[rt] \]
  \[ PC \leftarrow PC + 4 \]

---

**Components**

Memory: hold instruction and data

Registers: 32 32-bit registers
  - read rs
  - read rt
  - write rd
  - write rt

Program counter

Extender

Add and Sub registers or extended immediates

Add 4 to PC or Add extended immediate to PC (jump inst)
Combinational Logic Elements

- **Adder** (to add values)
  - A, B, CarryIn → Sum, Carry

- **MUX (multi-plexor)** (to choose between values)
  - A, B → Y

- **ALU** (to do add, subtract, or)
  - A, B → Result

Storage Element: Register (Basic Building Block)

- **Register**
  - Similar to the D Flip Flop except:
    - N-bit input and output
    - Write Enable input
  - Write Enable:
    - negated (0): Data Out will not change
    - asserted (1): Data Out will become Data In

Sequential Logic Elements

- Registers: n-bit input and output, D F/F, write enable
- rs, rt, rd: register specifiers

Fetching Instructions

- Fetching instructions involves:
  - reading the instruction from the Instruction Memory
  - updating the PC to hold the address of the next instruction
  - PC is updated every cycle, so it does not need an explicit write control signal
  - Instruction Memory is read every cycle, so it doesn’t need an explicit read control signal
Decoding Instructions

- Decoding instructions involves
  - sending the fetched instruction’s opcode and function field bits to the control unit
  - reading two values from the Register File
    » Register File addresses are contained in the instruction

Executing R Format Operations

- R format operations (add, sub, slt, and, or)
  - perform the (op and funct) operation on values in rs and rt
  - store the result back into the Register File (into location rd)

Executing Load and Store Operations

- Load and store operations involves
  - compute memory address by adding the base register (read from the Register File during decode) to the 16-bit signed-extended offset field in the instruction
  - store value (read from the Register File during decode) written to the Data Memory
  - load value, read from the Data Memory, written to the Register File

Executing Branch Operations

- Branch operations involves
  - compare the operands read from the Register File during decode for equality (zero ALU output)
  - compute the branch target address by adding the updated PC to the 16-bit signed-extended offset field in the instruction
Executin Jump Operations

- Jump operation involves
  - replace the lower 28 bits of the PC with the lower 26 bits of the fetched instruction shifted left by 2 bits

Creating a Single Datapath from the Parts

- Assemble the datapath segments and add control lines and multiplexors as needed
- Single cycle design – fetch, decode and execute each instructions in one clock cycle
  - no datapath resource can be used more than once per instruction, so some must be duplicated (e.g., separate Instruction Memory and Data Memory, several adders)
  - multiplexors needed at the input of shared elements with control lines to do the selection
  - write signals to control writing to the Register File and Data Memory
- Cycle time is determined by length of the longest path

Adding the Control

- Selecting the operations to perform (ALU, Register File and Memory read/write)
- Controlling the flow of data (multiplexor inputs)

Observations

- op field always in bits 31-26
- addr of registers to be read are always specified by the rs field (bits 25-21) and rt field (bits 20-16); for lw and sw rs is the base register
- addr of register to be written is in one of two places – in rt (bits 20-16) for lw; in rd (bits 15-11) for R-type instructions
- offset for beq, lw, and sw always in bits 15-0
Adding the Jump Operation

Single Cycle Control Unit: ALU control
On page 301, ALU control lines = Operation

<table>
<thead>
<tr>
<th>ALU control lines</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>AND</td>
</tr>
<tr>
<td>0001</td>
<td>OR</td>
</tr>
<tr>
<td>0010</td>
<td>add</td>
</tr>
<tr>
<td>0110</td>
<td>subtract</td>
</tr>
<tr>
<td>0111</td>
<td>set on less than</td>
</tr>
<tr>
<td>1100</td>
<td>NOR</td>
</tr>
</tbody>
</table>

ALU Control Implementation

ALU control lines function tables:

<table>
<thead>
<tr>
<th>ALUOp1</th>
<th>ALUOp0</th>
<th>F5</th>
<th>F4</th>
<th>F3</th>
<th>F2</th>
<th>F1</th>
<th>F0</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

a. The truth table for Operation2 = 1 (this table corresponds to the left bit of the Operation field in Figure C2.1)

<table>
<thead>
<tr>
<th>ALUOp1</th>
<th>ALUOp0</th>
<th>F5</th>
<th>F4</th>
<th>F3</th>
<th>F2</th>
<th>F1</th>
<th>F0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>O</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

b. The truth table for Operation1 = 1

<table>
<thead>
<tr>
<th>ALUOp1</th>
<th>ALUOp0</th>
<th>F5</th>
<th>F4</th>
<th>F3</th>
<th>F2</th>
<th>F1</th>
<th>F0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

c. The truth table for Operation0 = 1

Instruction opcode | ALUOP | Instruction operation | Funkt field | Desired ALU action | ALU control input, i.e., Operation |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>LW 00</td>
<td>load word</td>
<td>xxxxxxx</td>
<td>add</td>
<td>0010</td>
<td></td>
</tr>
<tr>
<td>SW 00</td>
<td>store word</td>
<td>xxxxxxx</td>
<td>add</td>
<td>0010</td>
<td></td>
</tr>
<tr>
<td>Branch equal 01</td>
<td>branch equal</td>
<td>xxxxxxx</td>
<td>subtract</td>
<td>0110</td>
<td></td>
</tr>
<tr>
<td>R type 10</td>
<td>add</td>
<td>100000</td>
<td>add</td>
<td>0010</td>
<td></td>
</tr>
<tr>
<td>R type 10</td>
<td>subtract</td>
<td>10010</td>
<td>subtract</td>
<td>0110</td>
<td></td>
</tr>
<tr>
<td>R type 10</td>
<td>AND</td>
<td>100100</td>
<td>and</td>
<td>0000</td>
<td></td>
</tr>
<tr>
<td>R type 10</td>
<td>OR</td>
<td>100101</td>
<td>or</td>
<td>0001</td>
<td></td>
</tr>
<tr>
<td>R type 10</td>
<td>set on less than</td>
<td>101010</td>
<td>set on less than</td>
<td>0111</td>
<td></td>
</tr>
</tbody>
</table>
Setting of the control signals

<table>
<thead>
<tr>
<th>Instruction</th>
<th>RegDst</th>
<th>ALUSrc</th>
<th>MemtoReg</th>
<th>RegWrite</th>
<th>MemRead</th>
<th>MemWrite</th>
<th>Branch</th>
<th>ALUOp1</th>
<th>ALUOp0</th>
</tr>
</thead>
<tbody>
<tr>
<td>R type</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>lw</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>sw</td>
<td>x</td>
<td>1</td>
<td>x</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>beq</td>
<td>x</td>
<td>0</td>
<td>x</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Control Unit PLA Implementation

<table>
<thead>
<tr>
<th>Control</th>
<th>Signal name</th>
<th>R-format</th>
<th>lw</th>
<th>sw</th>
<th>beq</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inputs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Op0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Op4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Op2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Op1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Op3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Outputs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RegWrite</td>
<td>1</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>MemtoReg</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>MemRead</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>MemWrite</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Branch</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>ALUOp1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>ALUOp0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

FIGURE C.2.4 The control function for the simple one-clock implementation is completely specified by this truth table. This table is the same as that shown in Figure 5.22.

Single Cycle Disadvantages & Advantages

- Uses the clock cycle inefficiently – the clock cycle must be timed to accommodate the slowest instruction
- especially problematic for more complex instructions like floating point multiply

- May be wasteful of area since some functional units (e.g., adders) must be duplicated since they can not be shared during a clock cycle
- Is simple and easy to understand
Multicycle Datapath Approach

- Let an instruction take more than 1 clock cycle to complete
  - Break up instructions into steps where each step takes a cycle while trying to
    » balance the amount of work to be done in each step
    » restrict each cycle to use only one major functional unit
  - Not every instruction takes the same number of clock cycles

- In addition to faster clock rates, multicycle allows functional units that can be used more than once per instruction as long as they are used on different clock cycles, as a result
  - only need one memory – but only one memory access per cycle
  - need only one ALU/adder – but only one ALU operation per cycle

At the end of a cycle

- Store values needed in a later cycle by the current instruction in an internal register (not visible to the programmer). All (except IR) hold data only between a pair of adjacent clock cycles (no write control signal needed)

IR – Instruction Register
MDR – Memory Data Register
A, B – regfile read data registers
ALUout – ALU output register

- Data used by subsequent instructions are stored in programmer visible registers (i.e., register file, PC, or memory)
Multicycle Control Unit

- Multicycle datapath control signals are not determined solely by the bits in the instruction
  - e.g., op code bits tell what operation the ALU should be doing, but not what instruction cycle is to be done next
- Must use a finite state machine (FSM) for control
  - a set of states (current state stored in State Register)
  - next state function (determined by current state and the input)
  - output function (determined by current state and the input)

Multicycle Control Unit

- Combinational control logic
- Datapath control points
- Inst Opcode
- State Reg
- Next State

The Five Steps of the Load Instruction

- IFetch: Instruction Fetch and Update PC
- Dec: Instruction Decode, Register Read, Sign Extend Offset
- Exec: Execute R-type; Calculate Memory Address; Branch Comparison; Branch and Jump Completion
- Mem: Memory Read; Memory Write Completion; R-type Completion (RegFile write)
- WB: Memory Read Completion (RegFile write)

INSTRUCTIONS TAKE FROM 3 - 5 CYCLES!
Logic Equations for Next-State Outputs

NextState1 = State0 = S3 • S2 • S1 • S0
NextState3 = State2 • (Op[5:0] = lw)
  = S3 • S2 • S1 • S0 • Op5 • Op4 • Op3 • Op2 • Op1 • Op0
NextState5 = State2 • (Op[5:0] = sw)
  = S3 • S2 • S1 • S0 • Op5 • Op4 • Op3 • Op2 • Op1 • Op0
NextState7 = State6 = S3 • S2 • S1 • S0
NextState9 = State1 • (Op[5:0] = jmp)
  = S3 • S2 • S1 • S0 • Op5 • Op4 • Op3 • Op2 • Op1 • Op0

NS0 is the logical sum of all these terms.
Implementing the Next-State Function with a Seuencc

Control unit

FIGURE 5.4.1

1
2
3

Inputs

State

Add/OI

Address select logic

Instruction register opcode field

Add/OI value

Action

0
1
2
3

Set state to 0
Dispatch with ROM 1
Dispatch with ROM 2
Use the incremented state

Dispatch ROM 1

Op
00000
00010
00100
01000
10000
10100
11000
11100

Op code name
Format
Jump
Branch
Load
Save
Terminate
Execute
Reset

Value
010
011
100
101
110
111
000
001

Dispatch ROM 2

Op
00011
00111
10111

Op code name
In
Out

Value
010
011
001

FIGURE 5.4.4 The values of the address/control lines are set in the control word that corresponds to the state.

State number

Address/control action

Value of Add/OI

0
1
2
3
4
5
6
7
8
9

Use incremented state
Use dispatch ROM 1
Use dispatch ROM 2
Dispatch with ROM 1
Dispatch with ROM 2
Replace state number by 0
Replace state number by 0
Use incremented state
Replace state number by 0
Replace state number by 0

Translating a Microprogram to Hardware

Control unit

Microcode memory

Input

Output

Datapath

Instruction register opcode field

Address select logic

Microprogram counter