Lecture 21
Performance Enhancement at Processor Level - Multicycle Implementation

Multicycle implementation
- break the major task of each instruction into a series of smaller tasks
- each task in execution will take 1 clock cycle
- allows functional unit to be used more than once per instruction as long as it is used as different clock cycles
- sharing resources reduce the amount of hardware required
- instructions are allowed to take different number of clock cycles
- can reuse functional units such as memory and ALU
- registers needed to store values between cycles
- finite state machine to control
  - because the values of the control signals now depend on which step we are performing

Figure 5.13  Page 354
- single memory unit for both instruction & data – replaced Inst. memory & Data memory
- single ALU replaced ALU & two adder
- one or more registers are added after every major functional unit to hold the output of that unit until the value is used in a subsequent clock cycle
- shared memory
- single ALU
- shared units need additional widening MUXS
- new temp register to hold data between clock cycles(A, B, ALUout, Inst. Reg Mem Data Reg)

<table>
<thead>
<tr>
<th>Initial</th>
<th>FSM</th>
<th>Microprogram</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rep</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sequential Control</td>
<td>Explicit Next</td>
<td>Micro PC + Dispatch ROMS</td>
</tr>
<tr>
<td>Logic Rep</td>
<td>Logic Equation</td>
<td>Truth Table</td>
</tr>
<tr>
<td>Implementation Tech</td>
<td>PLA</td>
<td>ROM</td>
</tr>
</tbody>
</table>
from Aluop

B from output

PC

Data Address
Memory

Inst or data

Memory data register Instruction Register

From Aluout

Data Reg# Reg# Reg# Register

B A

To Data input of mem

From PC output

ALU

To PC input

ALUOUT

To memory input address
To Data input of register
Control Links
Fig 5.32

Yellow major functional units
Green intermediate result storage registers
Red multiplexers
Pink control signals
Black data flow
Lecture 22
High level view of the FSM control

Fetch = Inst fetch + Inst decode and Register fetch

Execute

Mem FSM
FSM controller

Implemented using a block of comb. logic & register to hold the current state

To determine next state
Input from inst. Reg. Opcode field
Current State Input

Comb. control logic

Datapath control ops

State Reg

lw/sw continued
Controller

\begin{align*}
0 & \rightarrow \text{MemRead} \\
& \quad \text{AluSrcA} = 0 \\
& \quad \text{IorD} = 0 \\
& \quad \text{IR write} \\
& \quad \text{AluSrcB} = 01 \\
& \quad \text{Aluop} = 00 \\
& \quad \text{PCwrite} \\
& \quad \text{PCSource} = 00 \\
1 & \rightarrow \text{AluSrcA} = 0 \\
& \quad \text{AluSrcB} = 11 \\
& \quad \text{Aluop} = 00 \\
3 & \rightarrow \text{MemRead} \\
& \quad \text{IorD} = 1 \\
4 & \rightarrow \text{MemWrite} \\
& \quad \text{IorD} = 1 \\
5 & \rightarrow \text{RegDst} = 0 \\
& \quad \text{RegWrite} \\
& \quad \text{MemtoReg} = 1
\end{align*}
MemRead = State 0 + State 3
AluSrcA = State 2
IorD = State 3 + State 4
IRwrite = State 0
AluSrcB = State 0 + State 1
PCwrite = State 0
AluSrcB = State 1 + State 2
MemWrite = State 5
Regwrite = State 5
MemtoReg = State 5

NS Bits
Next State 0 = State 4 + State 5
Next State 1 = State 0
Next State 2 = State 1(lw + sw)
Next State 3 = State 2.lw
Next State 4 = State 3
Next State 5 = State 2(sw)

<table>
<thead>
<tr>
<th>NS2</th>
<th>NS1</th>
<th>NS0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

NS0 = 1 + 3 + 5
NS1 = 2 + 3
NS2 = 4 + 5

NS0 = State 0 + State 2.lw + State 2.sw
NS1 = State 1(lw + sw) + State 2.lw
NS2 = State 3 + State 2(sw)
FSM Controller for lw/sw

6 states

NS0 should be active whenever NS0 = 1 is true for
Next State 1 + Next State 3 + Next State 5

Next State 1 = State 0 = S2S1S0
Next State 3 = State 2(lw) + State 3
   = S2S1S0 (op5op4op3op2op1op0)

NS0 = 1 + 3 + 5
   = State 0 + (State 2 & lw) + (State 2 & sw) + (State 3)

NS1 = 2 + 3
   = (State 1 & lw + State 1 & sw)

NS2 = 4 + 5
   = (State 2 & sw) + State 3

NS0
000
001
010
011
100
101
Activity 23 Design PLA controllers to execute for R, Beq, J instructions. Then design the PLA to execute all LW+SW+R+BEQ+J instructions.

R-type instruction continue

Instruction Fetch

Instruction decode
Reg. Fetch

Execution
AluSrcA = 1
AluSrcB = 00
AluOp = 10

R-type completion
RegDst = 1
RegWrite
MemtoReg = 0

R-type

NS1
00
01
10
11
Branch

Instruction Fetch

Inst. decode Reg. Fetch

Branch completion

AluSrcA = 1
AluSrcB = 00
AluOp = 01
PCWrite
PCSource = 01
Branch

\[
\begin{array}{ccc}
\text{NS1} & \text{NS0} \\
0 & 0 \quad & \quad 0 & 1 \\
1 & 0 \\
\end{array}
\]

NS0 = State 1
NS1 = State 2

NS0 = 1 = Current State 0
NS1 = 2 = Current State 1 & \( \text{op} = J \)
\[ = S_1S_0(\text{op}_5\text{op}_4\text{op}_3\text{op}_2\text{op}_1\text{op}_0) \]
Jump

Inst Fetch

Inst decode
Reg. Fetch

Jump completion

PCWrite
PCSource = 10

Jump

op0~op5
S0~S3

Controller

Signals NS0~NS3
Put all FSM together
MemRead = State 0 + State 3 = 0000 + 0011 = S3'S2'S1'S0' + S3'S2'S1S0
AluSrcA = State 2 + State 6 + State 8 = 0010 + 0110 + 1000 = S3'S2'S1'S0' + S3'S2'S1S0 + S3S2'S1'S0'
IorD = State 3 + State 5 = 0011 + 0101 = S3'S2'S1S0 + S3'S2S1'S0
IRwrite = State 0 = 0000
AluSrcB0 = State 0 + State 1 = 0000 + 0001
AluOp0 = State 8 = 1000
AluOp1 = State 6 = 0110
PCwrite = State 0 + State 9 = 0000 + 1001
PCSource0 = State 8 = 1000
PCSource1 = State 9 = 1001
AluSrcB1 = State 1 + State 2 = 0001 + 0010
MemWrite = State 5 = 0101
PCWriteCond = State 8 = 1000
MemtoReg = State 4 = 0100
RegWrite = State 4 + State 7 = 0100 + 0111
RegDst = State 7 = 0111

Next State 0 = State 4 + State 5 + State 7 + State 8 + State 9
Next State 1 = State 0 = S3S2S1S0
Next State 2 = State 1(lw + sw) = S3S2S1S0(lw + sw)
Next State 3 = State 2(lw) = S3S2S1S0(lw)
Next State 4 = State 3
Next State 5 = State 2(sw)
Next State 6 = State 1(R-type)
Next State 7 = State 6
Next State 8 = State 1(beq)
Next State 9 = State (jump)
\[ NS0 = 1 + 3 + 5 + 7 + 9 \]
\[ NS1 = 2 + 3 + 6 + 7 \]
\[ NS2 = 4 + 5 + 6 + 7 \]
\[ NS3 = 8 + 9 \]

<table>
<thead>
<tr>
<th>NS3</th>
<th>NS2</th>
<th>NS1</th>
<th>NS0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Lecture 24

Instruction fetch step
IR = memory (PC)
PC = PC + 4

Instruction decode / Register fetch
A = Reg [IR[25-12]]
B = Reg[IR[20-16]]
Aluout = PC + sign extend(IR[15-0] << 2)

Execution address computation
Branch, Jump completion

Execution
Aluout = A OP B

Address computation
Aluout = A + Sign extended (IR[150])

Branch
If (A = B) then
PC = Aluout

Jumps
PC = PC [31-28] || (IR[250] << 2)

Memory access – or R-type completion
R-type completion
Reg [IR[15-11]] = Aluout

Memory access
Load MDR = memory [Aluout]
or
Store = memory[Aluout] = B

Memory Read Completion
Active for Memory Reference Instruction
Load : Reg[IR[20-16]] = MDR
Lecture 25
Microprogramming
- FSM can contain thousands of states
  - graphical representation difficult – good for small
  - specifying complex control function as equation without making any mistakes is difficult
- use micro instruction
  - define the set of datapath control signals that must be asserted in a given state
  - executing a micro instruction has the effect of asserting control signals specified by the micro inst.
- should specify sequentials
  - sometimes next state
  - sometimes depend on the ip
micro inst format

<table>
<thead>
<tr>
<th>label</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
</table>

label any string
Alucontrol
- specify the operation – add, sub, func code
- result in Aluout

SRC1
  First Aluop – PC, Reg A

SRC2
  Second OP – B, 4, op of sign ext unit, ext shift(op of shift by two unit)

Register Control
  Specify read or write for the register file & the course of the value of a write
  1. Read 2 regs rs rt of IR put data A & B
  2. WriteALU Reg
  3. Write MDR – write the content of MDR using rt of IR

Memory
  Specify Read or Write & the course for the memory
  For Read – specify the destination register
  1. Read PC – reading memory using PC as address
  2. Read ALU – Reading memory using ALUout as address write result intoMDR
  3. Write ALU – Write memory using ALUout as address contests of ???

PCWrite Control
  Specify the writing of the PC
  1. ALU
  2. ALUout-control
  3. Jump address
Specify how to choose the next micro instruction to the execution

1. Seq
2. Fetch
3. Dispatch

Alu Control – add – Aluop 00
   – sub – Aluop 01
   – func code – Aluop 10

SRC1 – PC  AluSrcA = 0
   – A  AluSrc A = 1

SRC2 – B  AluSrc B = 00
   – 4  AluSrc B = 01
      – Extend AluSrc B = 10
      – Extshft AluSrc B = 11

Reg control  write Alu  RegWrite
            RegDst = 1
            MemtoReg = 0

Write MDR  RegWrite
           RegDst = 0
           MemtoReg = 1

Memory  Read PC  MemRead
         Iord = 1

WriteAlu  MemWrite
         Iord = 1

PC Write Control  Alu  PCSource = 00
                   PCWrite

   Aluout  PCSource = 01
   Cond   PCWrite Cond

   Jump  PCSource = 10
           PCWrite

Sequence  Seq  Add 11
          Fetch  00
          Dis 1  01
          Dis 2  10
- Microinstruction placed in a ROM or a PLA

- two dispatch tables
- one to dispatch from state 1
- second one to dispatch from state 2

<table>
<thead>
<tr>
<th>Label</th>
<th>AluControl</th>
<th>SRC1</th>
<th>SRC2</th>
<th>Reg Control</th>
<th>Memory</th>
<th>Pcwrlte</th>
<th>Seq</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Fetch</td>
<td>Add</td>
<td>PC</td>
<td>4</td>
<td>Read PC</td>
<td>ALU</td>
<td>Seq</td>
</tr>
<tr>
<td>1</td>
<td>Add</td>
<td>PC</td>
<td>ExtShift</td>
<td>Read</td>
<td>Dispatch 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Mem1</td>
<td>Add</td>
<td>A</td>
<td>Extend</td>
<td>Dispatch 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Lw2</td>
<td>Add</td>
<td>A</td>
<td>Extend</td>
<td>Read ALU</td>
<td>Seq</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Write MDR</td>
<td>Fetch</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Sw2</td>
<td></td>
<td></td>
<td></td>
<td>Write ALU</td>
<td>Fetch</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Rformat1</td>
<td>Func Code</td>
<td>A</td>
<td>B</td>
<td>Write ALU</td>
<td>Seq</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Fetch</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Beq1</td>
<td>Subt</td>
<td>A</td>
<td>B</td>
<td>ALU-out cond</td>
<td>Fetch</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Jump1</td>
<td></td>
<td></td>
<td></td>
<td>Jump address</td>
<td>Fetch</td>
<td></td>
</tr>
</tbody>
</table>

Microprogram controller
Address Select logic    IP from IR opcode field

Dispatch ROM 1             Dispatch ROM 2

0

Add Ctrl

MUX 3

State

To PLA

Adder

- Data used by subsequent instruction stored in
  Reg file  User Visible
  PC or     State
  Memory    Element

- Data used by the same instruction stored into one of the additional Regs
- In multicycle design
  - assume that the clock cycle can accommodate at most one of the following operation
  - memory access
  - Reg file access (two reads or one write)
  - ALU OP

Any data produced by one of these 3 units must be saved into temp regs for use later cycle.

- Multicycle datapath for MIPS that handle Inst
  Fig 5.31 page 380

- need MUX for memory add

```
Aluout   PC
\downarrow \downarrow
1 \hspace{1em} 0
MUX
\downarrow \quad \text{Address}
```

- need MUX for top ALU IP

```
Aluout   PC output
\downarrow \downarrow
2 \hspace{1em} 0
MUX
\downarrow \quad \text{ALU 1}
```

- expand MUX on bottom ALU to 4 way

```
SL2 \hspace{1em} \text{Sign extended Or 15 inst bits} \quad 4 \quad \text{from B}
\downarrow \downarrow \downarrow \downarrow
3 \hspace{1em} 2 \hspace{1em} 1 \hspace{1em} 0
\downarrow \quad \text{ALU2}
```
### Dispatch ROM

<table>
<thead>
<tr>
<th>OP</th>
<th>Opcode name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>000000</td>
<td>R-type</td>
<td>0110</td>
</tr>
<tr>
<td>000010</td>
<td>Jump</td>
<td>1001</td>
</tr>
<tr>
<td>000100</td>
<td>Beq</td>
<td>1000</td>
</tr>
<tr>
<td>100011</td>
<td>Lw</td>
<td>0010</td>
</tr>
<tr>
<td>101011</td>
<td>Sw</td>
<td>0010</td>
</tr>
</tbody>
</table>