Why a Multiple Clock Cycle CPU?

- the problem =>
- the solution => break up execution into , each task taking a , different instructions requiring different numbers of cycles or tasks
- other advantages => reuse of functional units (e.g., alu, memory)
- performance = instructions * cpi * cycle time

Breaking Execution Into Clock Cycles

- We will have five execution steps (not all instructions use all five)
- We will use Register-Transfer-Language (RTL) to describe these steps
Breaking Execution Into Clock Cycles

- Introduces extra registers when:
  - signal is computed in one clock cycle and used in another, AND
  - the inputs to the functional block that outputs this signal can change before the signal is written into a state element.
- Significantly complicates control. Why?
- The goal is to balance the done each cycle.

1. Fetch

\[
IR = \text{Mem}[PC]  \\
PC = PC + 4  \\
(\text{may not be final value of PC})
\]

2. Instruction Decode and Register Fetch

\[
A = \text{Reg}[IR[25-21]]  \\
B = \text{Reg}[IR[20-16]]  \\
\text{ALUOut} = PC + (\text{sign-extend} (IR[15-0]) << 2)
\]

- compute target before we know if it will be used (may not be branch, branch may not be taken)
- target is a new state element (temp register)
- everything up to this point must be Instruction-independent, because we still haven’t decoded the instruction.
- everything Inst-dependent from here on.
3. Execution, memory address computation, or branch completion

• Memory reference (load or store)
  \[ \text{ALUOut} = A + \text{sign-extend}(IR[15-0]) \]

• R-type
  \[ \text{ALUOut} = A \text{ op } B \]

• Branch
  if (A == B) \( PC = \text{ALUOut} \)

At this point, Branch is complete, and we start over; others require more cycles.

CSE 141  Dean Tullsen

4. Memory access or R-type completion

• Memory reference
  - load
    \[ \text{MDR} = \text{Mem}[\text{ALUout}] \]
  - store
    \[ \text{Mem}[\text{ALUout}] = B \]

• R-type
  \[ \text{Reg}[IR[15-11]] = \text{ALUout} \]

R-type is complete

CSE 141  Dean Tullsen

5. Memory Write-Back

\[ \text{Reg}[IR[20-16]] = \text{MDR} \]

memory instruction is complete

CSE 141  Dean Tullsen

<table>
<thead>
<tr>
<th>Step</th>
<th>R-type</th>
<th>Memory</th>
<th>Branch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instruction Fetch</td>
<td>( IR = \text{Mem}[PC] )</td>
<td></td>
<td>Branch</td>
</tr>
<tr>
<td></td>
<td>( PC = PC + 4 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Instruction Decode/</td>
<td>( A = \text{Reg}[IR[25-21]] )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>register fetch</td>
<td>( B = \text{Reg}[IR[20-16]] )</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>( \text{ALUout} = PC + (\text{sign-extend}(IR[15-0])) \ll 2 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Execution, address</td>
<td>( \text{ALUout} = A \text{ op } B )</td>
<td>( \text{ALUout} = A + \text{sign-extend}(IR[15-0]) )</td>
<td>if (A==B) then (PC=ALUout)</td>
</tr>
<tr>
<td>computation, branch</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memory access or R-type</td>
<td>( \text{Reg}[IR[15-11]] = \text{ALUout} )</td>
<td>\text{memory-data} = \text{Mem}[\text{ALUout}] or \text{Mem}[\text{ALUout}] = B</td>
<td></td>
</tr>
<tr>
<td>completion</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Write-back</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

CSE 141  Dean Tullsen

Summary of execution steps

CSE 141  Dean Tullsen
Complete Multicycle Datapath

Instruction Fetch

Instruction Decode and Reg Fetch

3. Execution (R-type)

A = Register[IR[25-21]]
B = Register[IR[20-16]]
Target = PC + (sign-extend (IR[15-0]) << 2)

ALUout = A op B

IR = Memory[PC]
PC = PC + 4

(notice logic for jump instruction now included)
4. R-type Completion

Reg [IR[15-11]] = ALUout

3. Branch Completion

if (A == B) PC = Target

3. Memory Address Computation

ALUout = A + sign-extend(IR[15-0])

4. Memory Access

memory-data = Memory[ALUout], or Memory[ALUout] = B
5. Write-back

Reg[IR[20-16]] = memory-data

3. JMP Completion

PC = PC[31-28] | (IR[25-0] <<2)

Multicycle Control

- Single-cycle control used logic
- Multi-cycle control uses ??
- FSM defines a succession of states, transitions between states (based on inputs), and outputs (based on state)
- First two states same for every instruction, next state depends on opcode

Multicycle Control FSM

Instruction fetch

Decode and Register Fetch

Memory instructions

R-type instructions

Branch instructions

Jump instruction
First two states of the FSM

Instruction Fetch, state 0
Instruction Decode/Register Fetch, state 1

Start

MemRead
ALUSrcA = 0
IorD = 0
IRWrite
ALUSrcB = 01
ALUOp = 00
PCWrite
PCSource = 00

? Opcode = LW or SW
Opcode = BEQ
Opcode = JMP

Memory Inst
R-type Inst
Branch Inst
Jump Inst
FSM

CSE 141 Dean Tullsen

R-type Instructions

from state 1

ALUSrcA = 1
ALUSrcB = 00
ALUOp = 10

Execution
Completion
To state 0

CSE 141 Dean Tullsen

BEQ Instruction

from state 1

ALUSrcA = 1
ALUSrcB = 00
ALUOp = 01
PCWriteCond
PCSource = 01

To state 0

CSE 141 Dean Tullsen

Memory Instructions

from state 1

Address Computation

MemRead
IorD = 1
Memory
Access

MemWrite
IorD = 1

write-back
MemRead
MemtoReg = 1
RegDst = 0

To state 0

CSE 141 Dean Tullsen
Simple Questions

• How many cycles will it take to execute this code?
  lw $t2, 0($t3)
  lw $t3, 4($t3)
  beq $t2, $t3, Label  #assume not taken
  add $t5, $t2, $t3
  sw $t5, 8($t3)
  Label:  ...

• What is going on during the 8th cycle of execution?
• In what cycle does the actual addition of $t2 and $t3 take place?

• Assume 20% loads, 10% stores, 50% R-type, 20% branches, what is the CPI?
ROM Implementation

- ROM = "Read Only Memory"
  - values of memory locations are fixed ahead of time
- A ROM can be used to implement a truth table
  - if the address is \( m \)-bits, we can address \( 2^m \) entries in the ROM.
  - our outputs are the bits of data that the address points to.

\[
\begin{array}{c|c}
\text{m} & \text{n} \\
\hline
0000011 & 0011100 \\
01011000111000 & 1000000 \\
1010001 & 1100110 \\
1110111 \\
\end{array}
\]

\( 2^m \) is the "height", and \( n \) is the "width"

- How many inputs are there?
  - 6 bits for \( m \), 4 bits for \( n \) = 10 address lines
  - (i.e., \( 2^{10} = 1024 \) different addresses)
- How many outputs are there?
  - 16 outputs, 4 \( = 20 \) outputs
- ROM is \( 2^{10} \times 20 = 20K \) bits (and a rather unusual size)
- Rather wasteful, since for lots of the entries, the outputs are the same
  — i.e., opcode is often ignored

Multicycle CPU Key Points

- Performance gain achieved from variable-length instructions
- \( ET = \#\text{instrs} \times CPI \times \text{cycle time} \)
- Required very few new state elements
- More, and more complex, control signals
- Control requires FSM