18-447 Lecture 6: Microprogrammed Multi-Cycle Implementation

James C. Hoe
Department of ECE
Carnegie Mellon University
Housekeeping

• Your goal today
  – understand why VAX was possible and reasonable

• Notices
  – Lab 1, Part B, due this week
  – HW1, due Wed
  – **New Office Hours: M 11~12 and F 1:30~2:30**

• Readings
  – P&H Appendix C
  – Start reading the rest of P&H Ch 4
“Single-Cycle” Datapath (MIPS)

![Datapath Diagram]

[Based on original figure from P&H CO&D, COPYRIGHT 2004 Elsevier. ALL RIGHTS RESERVED.]
Worst-Case Critical Path

[Diagram showing the worst-case critical path in a computer system, with arrows indicating the flow of instructions and data through different memory and register stages.]

[Based on original figure from P&H CO&D, COPYRIGHT 2004 Elsevier. ALL RIGHTS RESERVED.]
# Single-Cycle Datapath Analysis

- Assume (numbers from P&H)
  - memory units (read or write): 200 ps
  - ALU and adders: 100 ps
  - register file (read or write): 50 ps
  - other combinational logic: 0 ps

<table>
<thead>
<tr>
<th>steps</th>
<th>IF</th>
<th>ID</th>
<th>EX</th>
<th>MEM</th>
<th>WB</th>
<th>Delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>resources</td>
<td>mem</td>
<td>RF</td>
<td>ALU</td>
<td>mem</td>
<td>RF</td>
<td></td>
</tr>
<tr>
<td>R-type</td>
<td>200</td>
<td>50</td>
<td>100</td>
<td>50</td>
<td>400</td>
<td></td>
</tr>
<tr>
<td>I-type</td>
<td>200</td>
<td>50</td>
<td>100</td>
<td>50</td>
<td>400</td>
<td></td>
</tr>
<tr>
<td>LW</td>
<td>200</td>
<td>50</td>
<td>100</td>
<td>200</td>
<td>600</td>
<td></td>
</tr>
<tr>
<td>SW</td>
<td>200</td>
<td>50</td>
<td>100</td>
<td>200</td>
<td>550</td>
<td></td>
</tr>
<tr>
<td>Bxx/JALR</td>
<td>200</td>
<td>50</td>
<td>100</td>
<td></td>
<td>350</td>
<td></td>
</tr>
<tr>
<td>JAL</td>
<td>200</td>
<td>100</td>
<td></td>
<td>50</td>
<td>300</td>
<td></td>
</tr>
</tbody>
</table>
Single-Cycle Implementations

• Good match for the sequential and atomic semantics of ISAs
  – instantiate programmer-visible state one-for-one
  – map instructions to combinational next-state logic

• But, contrived and inefficient
  1. all instructions run as slow as slowest instruction
  2. must provide worst-case combinational resource in parallel as required by any one instruction
  3. what about CISC ISAs? polyf?

Not the fastest, cheapest or even the simplest way
Multi-cycle Implementation: Ver 1.0

• Each instruction type take only as much time as needed
  – run a 50 psec clock
  – each instruction type take as many 50-psec clock cycles as needed

• Add “MasterEnable” signal so architectural state ignores clock edges until after enough time
  – an instruction’s effect is still purely combinational from state to state
  – all other control signal unaffected
Multi-Cycle Datapath: Ver 1.0

[Based on original figure from P&H CO&D, COPYRIGHT 2004 Elsevier. ALL RIGHTS RESERVED.]
Sequential Control: Ver 1.0

- IF1 -> IF2 -> IF3 -> IF4 -> ID
- EX1 -> EX2
- WB
- MEM4 <-> MEM3 <-> MEM2 <-> MEM1
- IF1: Bxx or JAL or JALR / MasterEn=1
- EX2: LW or SW
- IF1: I-type or R-type
- IF1: LW
- IF1: SW / MasterEn=1
- IF1: JAL
- IF1: / MasterEn=1
Microsequencer: Ver 1.0

- ROM as a combinational logic lookup table

** ROM size grows as $O(2^n)$ as the number of inputs

** ROM size grows as $O(m)$ as the number of outputs

literally holds the truth table
A systematic approach to FSM sequencing/control
Microcontroller/Microsequencer

- A stripped-down “processor” for sequencing and control
  - control states are like μPC
  - μPC indexed into a μprogram ROM to select an μinstruction
  - well-formed control-flow support (branch, jump)
  - fields in the μinstruction maps to control signals

- Very elaborate μcontrollers have been built
Performance Analysis

• Iron Law:
  \[ \text{wall clock time} = (\text{inst}/\text{program}) (\text{cyc}/\text{inst}) (\text{time}/\text{cyc}) \]

• For same ISA, inst/program is the same; okay to compare

\[ \text{MIPS} = \text{IPC} \times f_{\text{clk in MHz}} \]

- million instructions per second
- instructions per cycle
- frequency in MHz
- instructions per cycle
Performance Analysis

• Single-Cycle Implementation
  
  \[ 1 \times 1,667\text{MHz} = 1667 \text{ MIPS} \]

• Multi-Cycle Implementation
  
  \[ \text{IPC}_{\text{avg}} \times 20,000 \text{ MHz} = 2178 \text{ MIPS} \]

  what is \( \text{IPC}_{\text{average}} \) ?

• Assume: 25% LW, 15% SW, 40% ALU, 13.3% Branch, 6.7% Jumps [Agerwala and Cocke, 1987]
  
  – weighted arithmetic mean of CPI \( \Rightarrow 9.18 \)
  – weighted harmonic mean of IPC \( \Rightarrow 0.109 \)
  – weighted arithmetic mean of IPC \( \Rightarrow 0.115 \)

\[ \text{MIPS} = \text{IPC} \times f_{\text{clk}} \]
Reducing Datapath by Resource Reuse

How to reuse same adder for two additions in one instruction

“Single-cycle” reused same adder for different instructions
Reducing Datapath by Sequential Reuse

A LU cont rol

ALU control

3
4
Zero

RegWrite

to IR or not to IR?
Removing Redundancies

- Latch Enables: PC, IR, MDR, A, B, ALUOut, RegWr, MemWr
- Steering: ALUSrc1{RF, PC}, ALUSrc2{RF, immed}, MAddrSrc{PC, ALUOut}, RFDataSrc{ALUOut, MDR}

Could also reduce down to a single register read-write port!

[Based on original figure from P&H CO&D, COPYRIGHT 2004 Elsevier. ALL RIGHTS RESERVED.]
Synchronous Register Transfers

• Synchronous state with latch enables
  – PC, IR, RF, MEM, A, B, ALUOut, MDR
• One can enumerate all possible “register transfers”
• For example starting from PC
  – IR ← MEM[ PC ]
  – MDR ← MEM[ PC ]
  – PC ← PC ⊕ 4
  – PC ← PC ⊕ B
  – PC ← PC ⊕ immediate(IR)
  – ALUOut ← PC ⊕ 4
  – ALUOut ← PC ⊕ immediate(IR)
  – ALUOut ← PC ⊕ B
Useful Register Transfers (by dest)

- $PC \leftarrow PC + 4$
- $PC \leftarrow PC + \text{immediate}_{SB\text{-type},U\text{-type}}(IR)$
- $PC \leftarrow A + \text{immediate}_{SB\text{-type}}(IR)$
- $IR \leftarrow \text{MEM}[PC]$  
- $A \leftarrow \text{RF}[rs1(IR)]$
- $B \leftarrow \text{RF}[rs2(IR)]$
- $\text{ALUOut} \leftarrow A + B$
- $\text{ALUOut} \leftarrow A + \text{immediate}_{I\text{-type},S\text{-type}}(IR)$
- $\text{ALUOut} \leftarrow PC + 4$
- $\text{MDR} \leftarrow \text{MEM}[\text{ALUOut}]$
- $\text{MEM}[\text{ALUOut}] \leftarrow B$
- $\text{RF}[rd(IR)] \leftarrow \text{ALUOut}$
- $\text{RF}[rd(IR)] \leftarrow \text{MDR}$
RT Sequencing: R-Type ALU

- **IF**
  \[ IR \leftarrow \text{MEM[PC]} \]
- **ID**
  \[ A \leftarrow \text{RF[rs1(IR)]} \]
  \[ B \leftarrow \text{RF[rs2(IR)]} \]
- **EX**
  \[ \text{ALUOut} \leftarrow A + B \]
- **MEM**
- **WB**
  \[ \text{RF[rd(IR)]} \leftarrow \text{ALUOut} \]
  \[ \text{PC} \leftarrow \text{PC+4} \]

if MEM[PC] == ADD rd rs1 rs2
  GPR[rd] ← GPR[rs1] + GPR[rs2]
  PC ← PC + 4
RT Datapath Conflicts

Can utilize each resource only once per control step (cycle)
RT Sequencing: R-Type ALU

1. $\text{IR} \leftarrow \text{MEM[PC]}$
2. $\text{A} \leftarrow \text{RF[rs1(IR)]}$
   $\text{B} \leftarrow \text{RF[rs2(IR)]}$
3. $\text{ALUOut} \leftarrow \text{A + B}$
4. $\text{RF[rd(IR)]} \leftarrow \text{ALUOut}$
   $\text{PC} \leftarrow \text{PC+4}$
RT Sequencing: LW

- **IF**
  \[ IR \leftarrow MEM[\ PC\ ] \]

- **ID**
  \[ A \leftarrow RF[ \ rs1(IR)\ ] \]
  \[ B \leftarrow RF[ \ rs2(IR)\ ] \]

- **EX**
  \[ ALUOut \leftarrow A + \text{imm}_{l\text{-type}}(IR) \]

- **MEM**
  \[ MDR \leftarrow MEM[\ ALUOut\ ] \]

- **WB**
  \[ RF[\ rd(IR)\ ] \leftarrow MDR \]
  \[ PC \leftarrow PC + 4 \]

\[
\text{if MEM[PC]==LW rd offset(base)} \\
\quad \text{EA} = \text{sign-extend(offset)} + \text{GPR[base]} \\
\quad \text{GPR[rd]} \leftarrow \text{MEM[ EA]} \\
\quad \text{PC} \leftarrow \text{PC} + 4
\]
## Combined RT Sequencing

<table>
<thead>
<tr>
<th>R-Type</th>
<th>LW</th>
<th>SW</th>
<th>Branch</th>
<th>Jump</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>start:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>ALUOut</strong> &amp; $$\text{ALUOut} \leftarrow \text{A+B}$$</td>
<td><strong>ALUOut</strong> &amp; $$\text{ALUOut} \leftarrow \text{A+imm(IR)}$$</td>
<td><strong>ALUOut</strong> &amp; $$\text{ALUOut} \leftarrow \text{A+imm(IR)}$$</td>
<td><strong>PC</strong> &amp; $$\text{PC} \leftarrow \text{PC + 4}$$</td>
<td><strong>PC</strong> &amp; $$\text{PC} \leftarrow \text{PC+imm(IR)}$$</td>
</tr>
<tr>
<td><strong>RF[rd(IR)]</strong> &amp; $$\text{RF[rd(IR)]} \leftarrow \text{ALUOut}$$</td>
<td><strong>MDR</strong> &amp; $$\text{MDR} \leftarrow \text{M[ALUOut]}$$</td>
<td><strong>M[ALUOut]</strong> &amp; $$\text{M[ALUOut]} \leftarrow \text{B}$$</td>
<td><strong>PC</strong> &amp; $$\text{PC} \leftarrow \text{PC+4}$$</td>
<td><strong>PC</strong> &amp; $$\text{PC} \leftarrow \text{PC+imm(IR)}$$</td>
</tr>
<tr>
<td><strong>RF[rd(IR)]</strong> &amp; $$\text{RF[rd(IR)]} \leftarrow \text{MDR}$$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

RTs in each state corresponds to some setting of the control signals
Horizontal Microcode

Control Store: $2^n \times k$ bit (not including sequencing)

[Based on original figure from P&H CO&D, COPYRIGHT 2004 Elsevier. ALL RIGHTS RESERVED.]
18-447-S18-L06-S25, James C. Hoe, CMU/ECE/CALCM, ©2018
Vertical Microcode

![Diagram of microcode and datapath control](https://example.com/vertical-microcode-diagram.png)

1-bit signal means do this RT:

- "PC ← PC+4"
- "PC ← ALUOut"
- "PC ← PC[ 31:28 ],IR[ 25:0 ],2'b00"
- "IR ← MEM[ PC]"
- "A ← RF[ IR[ 25:21 ]]"
- "B ← RF[ IR[ 20:16 ]]"

Inputs from instruction register opcode field:

- Microcode storage
- Microprogram counter
- Address select logic

Outputs:

- Sequencing control

Datapath control outputs:

- "PC ← PC+4"
- "PC ← ALUOut"
- "PC ← PC[ 31:28 ],IR[ 25:0 ],2'b00"
- "IR ← MEM[ PC]"
- "A ← RF[ IR[ 25:21 ]]"
- "B ← RF[ IR[ 20:16 ]]"

Outputs:

- ROM
- k-bit output

[Based on original figure from P&H CO&D, COPYRIGHT 2004 Elsevier. ALL RIGHTS RESERVED.]
Microcoding for CISC

• Can we extend \( \mu \)controller and datapath?
  – to support a new instruction
  – to support a complex instruction, e.g. polyf

• Yes, ws very simple datapath do very complicated things easily but with a slowdown
  – if I can sequence an arbitrary RISC instruction then I can sequence an arbitrary “RISC program” as a \( \mu \)program sequence
  – will need some \( \mu \)ISA state (e.g. loop counters) for more elaborate \( \mu \)programs
  – more elaborate \( \mu \)ISA features also make life easier
Single-Bus Microarchitecture

[8086 Family User’s Manual]

Figure 4-3. 8086 Elementary Block Diagram
High Performance CISC Today

- High-perf x86s translate CISC inst’s to RISC uOPs
- Pentium-Pro decoding example:

16 bytes of x86 instructions

uop ROM: play-back a uOP sequence for more complicated instructions

primary decoder

decode 1st x86 into 1~4 uOPs

decoder

byte offset

decode up to 2 more simple x86 that each map to 1 uOP

uOP stream executes on a RISC internal machine