18-447 Lecture 6: Microcontrolled Multi-Cycle Implementation

James C. Hoe
Department of ECE
Carnegie Mellon University

Housekeeping

- Your goal today
  - understand why VAX was possible
- Notices
  - Lab 1, Part B, due this week
  - HW1, due Wed
- Readings
  - P&H Appendix D
  - Start reading the rest of P&H Ch 4
"Single-Cycle" Datapath (MIPS)

Worst-Case Critical Path
Single-Cycle Datapath Analysis

- Assume (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>50</td>
<td>600</td>
</tr>
<tr>
<td>SW</td>
<td>200</td>
<td>50</td>
<td>100</td>
<td>200</td>
<td></td>
<td>550</td>
</tr>
<tr>
<td>Bxx/JALR</td>
<td>200</td>
<td>50</td>
<td>100</td>
<td>50</td>
<td>350</td>
<td></td>
</tr>
<tr>
<td>JAL</td>
<td>200</td>
<td></td>
<td>100</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
  - all instructions run as slow as the slowest instruction
  - must provide worst-case combinational resource in parallel as required by any one instruction

- Not necessarily the simplest way to implement an ISA

Would you build a single-cycle implementation of VAX?
Multi-cycle Implementation: Ver 1.0

- Why not let each instruction type take only as much time as it needs
- Idea
  - run a 50 psec clock
  - let each instruction type take as many clock cycles as needed
  - add a "MasterEnable" so program-visible state (PVS) can ignores clock edges until after enough time
  - an instruction’s effect is still purely combinational from PVS to PVS
  - all other control signal also unaffected

Multi-Cycle Datapath: Ver 1.0

![Multi-Cycle Datapath Diagram]
Sequential Control: Ver 1.0

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

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

(Note: this is only about counting clock ticks)

<table>
<thead>
<tr>
<th>state label</th>
<th>cntrl flow</th>
<th>conditional targets</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>IF</strong>&lt;sub&gt;1&lt;/sub&gt;</td>
<td>next</td>
<td>R/l-type</td>
</tr>
<tr>
<td><strong>IF</strong>&lt;sub&gt;2&lt;/sub&gt;</td>
<td>next</td>
<td>-</td>
</tr>
<tr>
<td><strong>IF</strong>&lt;sub&gt;3&lt;/sub&gt;</td>
<td>next</td>
<td>-</td>
</tr>
<tr>
<td><strong>IF</strong>&lt;sub&gt;4&lt;/sub&gt;</td>
<td>branch</td>
<td>ID</td>
</tr>
<tr>
<td><strong>ID</strong></td>
<td>next</td>
<td>-</td>
</tr>
<tr>
<td><strong>EX</strong>&lt;sub&gt;1&lt;/sub&gt;</td>
<td>next</td>
<td>-</td>
</tr>
<tr>
<td><strong>EX</strong>&lt;sub&gt;2&lt;/sub&gt;</td>
<td>branch</td>
<td>WB</td>
</tr>
<tr>
<td><strong>MEM</strong>&lt;sub&gt;1&lt;/sub&gt;</td>
<td>next</td>
<td>-</td>
</tr>
<tr>
<td><strong>MEM</strong>&lt;sub&gt;2&lt;/sub&gt;</td>
<td>next</td>
<td>-</td>
</tr>
<tr>
<td><strong>MEM</strong>&lt;sub&gt;3&lt;/sub&gt;</td>
<td>next</td>
<td>-</td>
</tr>
<tr>
<td><strong>MEM</strong>&lt;sub&gt;4&lt;/sub&gt;</td>
<td>branch</td>
<td>-</td>
</tr>
<tr>
<td><strong>WB</strong></td>
<td>branch</td>
<td>IF&lt;sub&gt;1&lt;/sub&gt;</td>
</tr>
</tbody>
</table>

**CPI**

| 8 | 12 | 11 | 7 | 7 | 6 |

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 microprogram ROM to select a μinstruction
  - Well-formed control-flow support (branch, jump)
  - Fields in the μinstruction maps to control signals

- Very elaborate μcontrollers have been built

![Microprogram counter](https://example.com/microprogram-counter.png)

*Based on original figure from P&H CO&D, COPYRIGHT 2004 Elsevier. All Rights Reserved.*
Performance Analysis

- **Iron Law:**
  \[ \text{wall clock time} = \left( \frac{\text{inst}}{\text{program}} \right) \left( \frac{\text{cyc}}{\text{inst}} \right) \left( \frac{\text{time}}{\text{cyc}} \right) \]

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

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

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

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

  **Assume:** 25% LW, 15% SW, 40% ALU, 13.3% Branch, and 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 one adder for both additions in the same instruction

Previous example of reuse by mutually exclusive conditions

Reducing Datapath by Sequential Reuse

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!

Synchronous Register Transfers

- Synchronous state with latch enables
  - PC, IR, RF, MEM, A, B, ALUOut, MDR
- One can enumerate all possible “register transfers” in the datapath
- 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

Not all feasible RTs are meaningful
Useful Register Transfers

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

RT Sequencing: R-Type ALU

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

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

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

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

\[ \text{if MEM[PC]==LW } \text{rd offset(base)} \]
\[ \text{EA = sign-extend(offset) + GPR[base]} \]
\[ \text{GPR[rd] ← MEM[ EA ]} \]
\[ \text{PC ← PC + 4} \]

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>IR ← MEM[ PC ]</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **common steps**
  - A ← RF[ rs1(IR) ]
  - B ← RF[ rs2(IR) ]
  - ALUOut ← PC + imm\text{l-type}(IR)

- **opcode dependent steps**
  - ALUOut ← A + B
  - ALUOut ← A + imm\text{l-type}(IR)
  - PC ← PC + 4
  - PC ← PC + imm\text{l-type}(IR)

RTs in each state corresponds to some setting of the control signals
Microcoding for CISC

- Can we extend the μcontroller and datapath?
  - to support a new instruction I haven’t thought of yet
  - to support a complex instruction, e.g. polyf
- Yes, and probably more
  - if I can sequence an arbitrary RISC instruction then I can sequence an arbitrary “RISC program” as a μprogram sequence
  - will need some μISA state (e.g. loop counters) for more elaborate μprograms
  - more elaborate μISA features also make life easier
- μcoding allows very simple datapath do very powerful computation
  - a datapath as simple as a Turning machine is universal
  - μcode enables a minimal datapath to emulate any ISA you like (with a commensurate slow down)

Single-Bus Microarchitecture

[8086 Family User’s Manual]
High Performance CISC Today

- High-perf x86s translate CISC inst’s to RISC-like uOPs
  think of resulting uOPs as an equivalent program
- 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
  
  decode up to 2 more simple x86 that each map to 1 uOP
  
  uOP stream executes on a RISC internal machine

Terminology: Horizontal Microcode

[Based on original figure from P&H CO&D, COPYRIGHT 2004 Elsevier. ALL RIGHTS RESERVED.]
Terminology: Vertical Microcode

- Inputs from instruction register opcode field
- 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]]"

1-bit signal means do this RT

If done right (i.e., m<<n, and m<<k), two ROMs together (2^n×m+2^m×k bit) should be smaller than horizontal μcode ROM (2^n×k bit)

Terminology: Nanocode and Millicode

- Nanocode: a level below μcode
  - μprogrammed control for sub-systems (e.g., a complicated floating-point module) that acts as a slave in a μcontrolled datapath
  - e.g., the polyf sequence may be generated by a separate nanocontroller in the FPU
- Millicode: a level above μcode
  - ISA-level subroutines hardcoded into a ROM that can be called by the μcontroller to handle really complicated operations
  - e.g., to add polyf to MIPS, one may code up polyf as a software routine (in ROM invisible to user) to be called by the μcontroller when the polyf opcode is decoded
- In both cases, we avoid complicating the main μcontroller with polyf support
Nanocode Concept Illustrated

We refer to this as "nanocode" when a μcoded subsystem is embedded in a μcoded system.

a "μcoded" processor implementation

ROM
μPC

processor datapath

a "μcoded" FPU implementation

ROM
μPC

arithmetic datapath