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
  – HW1, past due (see Handout #6: HW 1 solutions)
  – Lab 1, Part B, due this week
  – HW2, due Mon 2/21 (Handout #5: HW 2)

• Readings
  – P&H Appendix C
  – Start reading the rest of P&H Ch 4
“Single-Cycle” Datapath: Is it any good?

Neither fast nor cheap, and not even simplest

[Based on original figure from P&H CO&D, COPYRIGHT 2004 Elsevier. ALL RIGHTS RESERVED.]
Go Fast(er)!!
Iron Law of Processor Performance

- \( \text{time/program} = (\text{inst/program}) \cdot (\text{cyc/inst}) \cdot (\text{time/cyc}) \)

- **Contributing factors**
  - \( \text{time/cyc} \): architecture and implementation
  - \( \text{cyc/inst} \): architecture, implementation, instruction mix
  - \( \text{inst/program} \): architecture, nature and quality of prgm

- **Note**: \( \text{cyc/inst} \) is a workload average potentially large instantaneous variations due to instruction type and sequence
Worst-Case Critical Path
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></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></td>
<td>50</td>
<td>400</td>
</tr>
<tr>
<td>I-type</td>
<td>200</td>
<td>50</td>
<td>100</td>
<td></td>
<td>50</td>
<td>400</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</td>
<td>200</td>
<td>50</td>
<td>100</td>
<td></td>
<td></td>
<td>350</td>
</tr>
<tr>
<td>JALR</td>
<td>200</td>
<td>50</td>
<td>100</td>
<td></td>
<td>50</td>
<td>350</td>
</tr>
<tr>
<td>JAL</td>
<td>200</td>
<td>100</td>
<td></td>
<td></td>
<td>50</td>
<td>300</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

[Diagram of multi-cycle datapath with various components and instructions]

[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
- MEM₁ → MEM₂ → MEM₃ → MEM₄

Instructions:
- LW or SW / MasterEn=1
- JAL
- Bxx or JAL or JALR / MasterEn=1
- I-type or R-type
- LW
**Performance Analysis**

- **Iron Law:**
  \[
  \text{time/program} = (\text{inst/program}) \times (\text{cyc/inst}) \times (\text{time/cyc})
  \]

- 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
Performance Analysis

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

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

• 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 \(\Rightarrow 0.115\)

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

Outputs $m$

$2^n$ rows

address $n$ / input

Inputs from instruction

State register

Next state

Combinational control logic

MasterEn
# Microcoding: Ver 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></td>
<td></td>
<td>R/I-type</td>
</tr>
<tr>
<td><strong>IF1</strong></td>
<td>next</td>
<td>-</td>
</tr>
<tr>
<td><strong>IF2</strong></td>
<td>next</td>
<td>-</td>
</tr>
<tr>
<td><strong>IF3</strong></td>
<td>next</td>
<td>-</td>
</tr>
<tr>
<td><strong>IF4</strong></td>
<td>goto</td>
<td>ID</td>
</tr>
<tr>
<td><strong>ID</strong></td>
<td>next</td>
<td>-</td>
</tr>
<tr>
<td><strong>EX1</strong></td>
<td>next</td>
<td>-</td>
</tr>
<tr>
<td><strong>EX2</strong></td>
<td>goto</td>
<td>WB</td>
</tr>
<tr>
<td><strong>MEM1</strong></td>
<td>next</td>
<td>-</td>
</tr>
<tr>
<td><strong>MEM2</strong></td>
<td>next</td>
<td>-</td>
</tr>
<tr>
<td><strong>MEM3</strong></td>
<td>next</td>
<td>-</td>
</tr>
<tr>
<td><strong>MEM4</strong></td>
<td>goto</td>
<td>-</td>
</tr>
<tr>
<td><strong>WB</strong></td>
<td>goto</td>
<td>IF1</td>
</tr>
<tr>
<td><strong>CPI</strong></td>
<td></td>
<td>8</td>
</tr>
</tbody>
</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
  - μprogram state and well-formed control-flow support (branch, jump)
  - fields in the μinstruction maps to control signals
- Very elaborate μcontrollers have been built
Go Cheap!!
(And More Capable)
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

![Diagram of instruction pipeline with instructions on memory, register reads, ALU operations, and results]

to IR or not to IR?

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

• 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 (by dest)

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

- **IF**
  
  $\text{IR} \leftarrow \text{MEM}[\text{PC}]$  \hspace{1cm} \text{step 1}

- **ID**
  
  $A \leftarrow \text{RF}[\text{rs1} (\text{IR})]$  \hspace{1cm} \text{step 2}
  
  $B \leftarrow \text{RF}[\text{rs2} (\text{IR})]$  \hspace{1cm} \text{step 3}

- **EX**
  
  $\text{ALUOut} \leftarrow A + B$  \hspace{1cm} \text{step 4}

- **MEM**

- **WB**
  
  $\text{RF}[\text{rd} (\text{IR})] \leftarrow \text{ALUOut}$  \hspace{1cm} \text{step 5}
  
  $\text{PC} \leftarrow \text{PC} + 4$  \hspace{1cm} \text{step 6}

if $\text{MEM}[\text{PC}] == \text{ADD} \text{ rd} \text{ rs1 rs2}$

$\text{GPR}[\text{rd}] \leftarrow \text{GPR}[\text{rs1}] + \text{GPR}[\text{rs2}]$

$\text{PC} \leftarrow \text{PC} + 4$
RT Datapath Conflicts

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

1. IR ← MEM[ PC ]
2. A ← RF[ rs1(IR) ]
   B ← RF[ rs2(IR) ]
3. ALUOut ← A + B
4. RF[ rd(IR) ] ← ALUOut
   PC ← PC+4
RT Sequencing: LW

- **IF**
  \[ \text{IR} \leftarrow \text{MEM}[\text{PC}] \]

- **ID**
  \[ \text{A} \leftarrow \text{RF}[\text{rs}_1(\text{IR})] \]
  \[ \text{B} \leftarrow \text{RF}[\text{rs}_2(\text{IR})] \]

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

- **MEM**
  \[ \text{MDR} \leftarrow \text{MEM}[\text{ALUOut}] \]

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

---

if \(\text{MEM}[\text{PC}] == \text{LW \ rd \ offset(base)}\)

\[ \text{EA} = \text{sign-extend}(\text{offset}) + \text{GPR}[\text{base}] \]

\[ \text{GPR}[\text{rd}] \leftarrow \text{MEM}[\text{EA}] \]

\[ \text{PC} \leftarrow \text{PC} + 4 \]
### 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>IR « MEM[ PC ]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A « RF[ rs1(IR) ]</td>
<td>B « RF[ rs2(IR) ]</td>
<td>ALUOut « PC+imm(IR)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**opcode dependent steps**

- ALUOut &laquo; A+B  
- RF[rd(IR)] &laquo; ALUOut  
- PC &laquo; PC+4

- ALUOut &laquo; A+imm(IR)  
- MDR &laquo; M[ALUOut]  
- PC &laquo; PC+4

- ALUOut &laquo; A+imm(IR)  
- M[ALUOut] &laquo; B  
- cond?( A , B )  
- PC &laquo; PC+4

- RF[rd(IR)] &laquo; MDR  
- PC &laquo; PC+4

- PC &laquo; ALUOut  

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

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

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]]"

Still more elaborate behaviors can be sequenced as μsubroutines
μProgrammed Implementation

[Diagram showing the flow of data from instruction register to datapath with various components labeled: Combining control logic, Instruction register, Memory data register, ALU, ALUOut, etc.]

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

• Can we extend last slide
  – to support a new instruction?
  – to support a complex instruction, e.g. polyf?
• Yes, very simple datapath do very complicated things easily but with a slowdown
  – Turing complete

  *With enough uOp’s, can sequence arbitrary complex instructions and even whole programs*
  – will need some μISA state (e.g. loop counters) for more elaborate μprograms
  – more elaborate μISA features also make life easier
Single-Bus Microarchitecture

[8086 Family User’s Manual]

Figure 4-3. 8086 Elementary Block Diagram
Evolution of ISAs

• Why were the earlier ISAs so simple? e.g., EDSAC
  – technology
  – precedence

• Why did it get so complicated later? e.g., VAX11
  – assembly programming
  – lack of memory size and performance
  – microprogrammed implementation

• Why did it become simple again? e.g., RISC
  – memory size and speed (cache!)
  – compilers

• Why is x86 still so popular?
  – technical merit vs. {SW base, psychology, deep pocket}

Why has ARM thrived while other RISC ISAs vanished

Why RISC-V now?
1980’s CISC vs RISC Debate

- time/program = (inst/program) (cyc/inst) (time/cyc)
- “Performance from architecture: comparing a RISC and a CISC with similar hardware organization”, Bhandarkar&Clark, 1991
  - time/cyc on par (MIPS R2000 vs VAX 8700)
  - RISC increases inst/program by ~2
  - CISC increases cyc/inst by ~6

**RISC factor: 2.7 savings in cyc/program**
End of RISC/CISC Debate

CISC won or RISC won?
High Performance CISC Today

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

```
16 bytes of x86 instructions

uOp ROM: play-back a uOp sequence for more complicated instructions
```

```
primary decoder
```

```
decoder
decode 1st x86 into 1~4 uOp’s
```

```
decoder
decode up to 2 more simple x86 that each map to 1 uOp
```

```
uOp stream executes on a RISC internal machine
```

Compilers helps by avoiding bad insts