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

- 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

- wall clock time = \(\text{inst/program}) \times (\text{cyc/inst}) \times (\text{time/cyc})\)

- Contributing factors
  - time/cyc: architecture and implementation
  - cyc/inst: architecture, implementation, instruction mix
  - inst/program: architecture, nature and quality of prgm

- **Note**: cyc/inst is a workload average
  potentially large instantaneous variations
due to instruction type and sequence
Worst-Case Critical Path

[Diagram showing the critical path with highlighted paths and controls.]

[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 resources</th>
<th>ID</th>
<th>EX</th>
<th>MEM</th>
<th>WB</th>
<th>Delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>R-type</td>
<td>200 mem</td>
<td>50</td>
<td>100</td>
<td>50</td>
<td>400</td>
<td></td>
</tr>
<tr>
<td>I-type</td>
<td>200 mem</td>
<td>50</td>
<td>100</td>
<td>50</td>
<td>400</td>
<td></td>
</tr>
<tr>
<td>LW</td>
<td>200 mem</td>
<td>50</td>
<td>100</td>
<td>200</td>
<td>50</td>
<td>600</td>
</tr>
<tr>
<td>SW</td>
<td>200 mem</td>
<td>50</td>
<td>100</td>
<td>200</td>
<td>50</td>
<td>550</td>
</tr>
<tr>
<td>Bxx/JALR</td>
<td>200 mem</td>
<td>50</td>
<td>100</td>
<td>50</td>
<td>350</td>
<td></td>
</tr>
<tr>
<td>JAL</td>
<td>200 mem</td>
<td>100</td>
<td>50</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

IF$_1$ → IF$_2$ → IF$_3$ → IF$_4$ → ID

IF$_2$ → IF$_1$
IF$_3$ → IF$_2$
IF$_4$ → IF$_3$
ID → IF$_4$

EX$_1$ → ID
JAL

EX$_2$ → MEM$_4$
Bxx or JAL or JALR / MasterEn=1

MEM$_4$ → MEM$_3$ → MEM$_2$ → MEM$_1$
LW or SW

MEM$_1$ → MEM$_2$
MEM$_2$ → MEM$_3$
MEM$_3$ → MEM$_4$
MEM$_4$ → WB

WB → IF$_1$

LW

SW / MasterEn=1

I-type or R-type

MasterEn=1
Performance Analysis

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

• 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
  \[\text{IPC}_{\text{avg}} \times 20,000 \text{ MHz} = 2178 \text{ MIPS}\]
  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 \(\Rightarrow 0.115\)

\[\text{MIPS} = \text{IPC} \times f_{\text{clk}}\]
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
## Microcoding: Ver 0
(note: this is only about counting clock ticks)

<table>
<thead>
<tr>
<th>state label</th>
<th>cntrl flow</th>
<th>R/I-type</th>
<th>LW</th>
<th>SW</th>
<th>Bxx</th>
<th>JALR</th>
<th>JAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF&lt;sub&gt;1&lt;/sub&gt;</td>
<td>next</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>IF&lt;sub&gt;2&lt;/sub&gt;</td>
<td>next</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>IF&lt;sub&gt;3&lt;/sub&gt;</td>
<td>next</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>IF&lt;sub&gt;4&lt;/sub&gt;</td>
<td>goto</td>
<td>ID</td>
<td>ID</td>
<td>ID</td>
<td>ID</td>
<td>ID</td>
<td>EX&lt;sub&gt;1&lt;/sub&gt;</td>
</tr>
<tr>
<td>ID</td>
<td>next</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>EX&lt;sub&gt;1&lt;/sub&gt;</td>
<td>next</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>EX&lt;sub&gt;2&lt;/sub&gt;</td>
<td>goto</td>
<td>WB</td>
<td>MEM&lt;sub&gt;1&lt;/sub&gt;</td>
<td>MEM&lt;sub&gt;1&lt;/sub&gt;</td>
<td>IF&lt;sub&gt;1&lt;/sub&gt;</td>
<td>IF&lt;sub&gt;1&lt;/sub&gt;</td>
<td>IF&lt;sub&gt;1&lt;/sub&gt;</td>
</tr>
<tr>
<td>MEM&lt;sub&gt;1&lt;/sub&gt;</td>
<td>next</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>MEM&lt;sub&gt;2&lt;/sub&gt;</td>
<td>next</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>MEM&lt;sub&gt;3&lt;/sub&gt;</td>
<td>next</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>MEM&lt;sub&gt;4&lt;/sub&gt;</td>
<td>goto</td>
<td>-</td>
<td>WB</td>
<td>IF&lt;sub&gt;1&lt;/sub&gt;</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>WB</td>
<td>goto</td>
<td>IF&lt;sub&gt;1&lt;/sub&gt;</td>
<td>IF&lt;sub&gt;1&lt;/sub&gt;</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td><strong>CPI</strong></td>
<td></td>
<td><strong>8</strong></td>
<td><strong>12</strong></td>
<td><strong>11</strong></td>
<td><strong>7</strong></td>
<td><strong>7</strong></td>
<td><strong>6</strong></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

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”

• 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)

- PC ← PC + 4
- PC ← PC + immediate_{SB-type,U-type}(IR)
- PC ← A + immediate_{SB-type}(IR)
- IR ← MEM[ PC ]
- A ← RF[ rs1(IR) ]
- B ← RF[ rs2(IR) ]
- ALUOut ← A + B
- ALUOut ← A + immediate_{l-type,S-type}(IR)
- ALUOut ← PC + 4
- MDR ← MEM[ ALUOut ]
- MEM[ ALUOut ] ← B
- RF[ rd(IR) ] ← ALUOut
- RF[ rd(IR) ] ← MDR
RT Sequencing: R-Type ALU

- **IF**
  \[ \text{IR} \leftarrow \text{MEM}[\text{PC}] \] \hspace{1cm} \text{step 1} \\
- **ID**
  \[ \text{A} \leftarrow \text{RF}[\text{rs1}(\text{IR})] \] \hspace{1cm} \text{step 2} \\
  \[ \text{B} \leftarrow \text{RF}[\text{rs2}(\text{IR})] \] \hspace{1cm} \text{step 3} \\
- **EX**
  \[ \text{ALUOut} \leftarrow \text{A} + \text{B} \] \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} \text{ 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**
  \[ IR \leftarrow MEM[ PC ] \]
- **ID**
  \[ A \leftarrow RF[ rs1( IR ) ] \]
  \[ B \leftarrow RF[ rs2( IR ) ] \]
- **EX**
  \[ ALUOut \leftarrow A + \text{imm}_{l-type}( IR ) \]
- **MEM**
  \[ MDR \leftarrow MEM[ ALUOut ] \]
- **WB**
  \[ RF[ rd( IR ) ] \leftarrow MDR \]
  \[ PC \leftarrow PC + 4 \]

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

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

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

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

### common steps

- ALUOut ← A+B
- ALUOut ← A+imm(IR)
- ALUOut ← A+imm(IR)
- PC ← PC + 4
- PC ← PC + 4
- RF[rd(IR)] ← ALUOut
- PC ← PC + 4
- RF[rd(IR)] ← M[ALUOut]
- PC ← PC + 4
- M[ALUOut] ← B
- PC ← PC + 4
- cond?( A, B )
- RF[rd(IR)] ← MDR
- PC ← ALUOut
- PC ← 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 ]”
“B ← RF[ IR[ 20:16 ] ]”

…...

m-bit input

k-bit output

Still more elaborate behaviors can be sequenced as μsubroutines
μProgrammed Implementation

[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
  – 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
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 1\textsuperscript{st} 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
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?