CPE300: Digital System Architecture and Design

Fall 2011
MW 17:30-18:45 CBC C316

1-Bus Architecture and Datapath
10242011

http://www.egr.unlv.edu/~b1morris/cpe300/
Outline

- Midterm
- 1-Bus Microarchitecture
- Datapath
- 1-Bus Logic Design
Midterm Scores

- Max = 65.5, min = 28
- Mean = 48.6, std = 10.8
  - 48+ should feel pretty good
  - Mean + std should feel very good

![Histogram of midterm scores with mean and standard deviation indicated.]
Midterm Discussion

• Go through solutions on board
Chapter 4 - Processor Design

1. The Design Process
2. A 1-bus Microarchitecture for SRC
3. Data Path Implementation
4. Logic Design for the 1-bus SRC
5. The Control Unit
6. The 2- and 3-bus Processor Designs
7. The Machine Reset Process
8. Machine Exceptions
Register Transfer Descriptions

- Abstract RTN
  - Defines “what” not the “how” (Chapter 2)
    - Overall effect of instructions on programmer-visible registers
  - Implementation independent
    - Registers and operations

- Concrete RTN
  - Detailed register transfer steps in datapath to produce overall effect
    - Dependent on implementation details
  - Steps correspond to processor clock pulses
The (SRC) Design Process

1. Informal description – abstract RTN (Chapter 2)
2. Propose several block diagram architectures to support the abstract RTN
   ▫ Develop RTN steps consistent with architecture
   ▫ Track demands made by concrete RTN on hardware
3. Design datapath hardware and identify needed control signals
4. Design control unit to generate control signals
1-Bus SRC Microarchitecture

- 5 classic components of computer
  - Memory, Input, Output
  - CPU – Control and Datapath
High-Level View of 1-Bus SRC Design

Copyright © 2004 Pearson Prentice Hall, Inc.
Microarchitecture Constraints

- One bus connecting registers
  - Only single register transfer at a time
- Memory address must be copied into memory address (MA) register by CPU
- Memory data written from or read into memory data (MD) register
- ALU operation
  - First operand always registered in A
  - Second operand always comes from bus
  - Result registered in C
- Information into IR and MA only from bus
  - Decoder (not shown) interprets contents of IR
  - MA supplies address to memory not CPU bus
RTN for \textbf{ADD} Instruction

- Develop steps to execute instruction
- Abstract RTN
  - (IR\rightleftharpoons M[PC]: PC\rightleftharpoons PC+4; instruction\_execution);
  - Instruction\_execution := (... 
    \textbullet add(:=op=12) \rightarrow R[ra] \leftarrow R[rb]+R[rc];
    \textbullet ...);

- Concrete RTN
  - 3 concrete RT (T3, T4, T5)
  - 2 RT in T0
  - 6 total clock cycles

<table>
<thead>
<tr>
<th>Step</th>
<th>RTN</th>
</tr>
</thead>
<tbody>
<tr>
<td>T0</td>
<td>MA\rightleftharpoons PC: C\rightleftharpoons PC+4;</td>
</tr>
<tr>
<td>T1</td>
<td>MD\rightleftharpoons M[MA]: PC\leftarrow C</td>
</tr>
<tr>
<td>T2</td>
<td>IR\leftarrow MD</td>
</tr>
<tr>
<td>T3</td>
<td>A\leftarrow R[rb]</td>
</tr>
<tr>
<td>T4</td>
<td>C\leftarrow A+R[rc];</td>
</tr>
<tr>
<td>T5</td>
<td>R[ra]\leftarrow C</td>
</tr>
</tbody>
</table>

Fetch execution
Concrete RTN to Subunits

• ALU must be able to add two 32-bit values \((\text{ADD})\)
• ALU must be able to increment B input by 4
  ▫ \(C \leftarrow PC + 4\);
• Memory read must use address from MA and return data to MD
  ▫ \(MD \leftarrow M[MA], \ IR \leftarrow MD\)
• RT separated by colon (:) are operations in the same clock cycle
• Steps T0-T2 make up instruction fetch and is the same for all instructions
RTN for **ADDI** Instruction

- **Abstract RTN**
  - (IRASI M[PC]: PCASI PC+4; instruction_execution);
  - Instruction_execution := (... 
    · addi(:=op=13)ASI R[ra]ASI R[rb]+c2<16..0> {two’s complement, sign-extend};
    · ...);

- **Concrete RTN**
  - Same as **ADD** except step T4
  - Requires hardware for sign extension

<table>
<thead>
<tr>
<th>Step</th>
<th>RTN</th>
</tr>
</thead>
<tbody>
<tr>
<td>T0-T2</td>
<td>Instruction Fetch</td>
</tr>
<tr>
<td>T3</td>
<td>AASI R[rb]</td>
</tr>
<tr>
<td>T4</td>
<td>CASI A+c2 {sign-extend};</td>
</tr>
<tr>
<td>T5</td>
<td>R[ra]ASI C</td>
</tr>
</tbody>
</table>
More Complete View of 1-Bus SRC Design

- Concrete RTN adds detail to the datapath

Condition bit flip-flop

IR register logic and data paths

Shift counter register
RTN for Load/Store Instruction

• Abstract RTN
  - ld(:=op=1) → R[ra] ← M[disp]:
  - st(:=op=3) → M[disp] ← R[ra]:
    - disp<31..0> := ((rb=0) → c2<16..0> {sign-extend}:
      (rb≠0) → R[rb] + c2<16..0>{sign-ext, 2’s comp}

• Concrete RTN

<table>
<thead>
<tr>
<th>Step</th>
<th>RTN ld</th>
<th>RTN st</th>
</tr>
</thead>
<tbody>
<tr>
<td>T0-T2</td>
<td>Instruction Fetch</td>
<td></td>
</tr>
<tr>
<td>T3</td>
<td>A ← (rb=0 → 0: rb≠0 → R[rb]);</td>
<td></td>
</tr>
<tr>
<td>T4</td>
<td>C ← A+(16@IR&lt;16&gt;#IR&lt;15..0&gt;);</td>
<td></td>
</tr>
<tr>
<td>T5</td>
<td>MA ← C;</td>
<td></td>
</tr>
<tr>
<td>T6</td>
<td>MD ← M[MA];</td>
<td>MD ← R[ra]</td>
</tr>
<tr>
<td>T7</td>
<td>R[ra] ← MD;</td>
<td>M[MA] ← MD;</td>
</tr>
</tbody>
</table>

T3, T4 are effective address arithmetic calculation
Notes for Load/Store RTN

- T0-T2 are same as for add (all instructions)
- T3-T5 are same for ld and st – calculate disp
- Need way to use 0 for R[rb] when rb=0
- 15-bit sign extension of IR<16..0> is needed
- Memory read into MD at T6 of ld
- Write of MD into memory at T7 of st
RTN for Conditional Branch

- **Abstract RTN**
  - \( \text{br} (:= \text{op}=8) \rightarrow (\text{cond} \rightarrow \text{PC} \leftarrow R[rb]) : \)
    - \( \text{cond} := ( \)
      - \( c3<2..0>=0 \rightarrow 0 : \text{never} \)
      - \( c3<2..0>=1 \rightarrow 1 : \text{always} \)
      - \( c3<2..0>=2 \rightarrow R[rc]=0 : \text{if register is zero} \)
      - \( c3<2..0>=3 \rightarrow R[rc] \neq 0 : \text{if register is nonzero} \)
      - \( c3<2..0>=4 \rightarrow R[rc]<31>=0 : \text{if register is positive or zero} \)
      - \( c3<2..0>=5 \rightarrow R[rc]<31>=1 ) : \text{if register is negative} \)

- **Concrete RTN**

<table>
<thead>
<tr>
<th>Step</th>
<th>RTN</th>
</tr>
</thead>
<tbody>
<tr>
<td>T0-T2</td>
<td>Instruction Fetch</td>
</tr>
<tr>
<td>T3</td>
<td>( \text{CON} \leftarrow \text{cond}(R[rc]) ; )</td>
</tr>
<tr>
<td>T4</td>
<td>( \text{CON} \rightarrow \text{PC} \leftarrow R[rb] ; )</td>
</tr>
</tbody>
</table>

\( \text{CON} \) is 1-bit register that is set based on condition logic: the contents of \( c<2..0> \) and \( R[rc] \)
Notes on Conditional Branch RTN

- $c_3^{<2..0>}$ are just 3 low order bits of IR
- $\text{cond}()$ is evaluated by combinational logic circuit having inputs $R[rc]$ and $c_3^{<2..0>}$
- One bit $\text{CON}$ register is not accessible to the programmer
  - Holds intermediate output of combinational logic for the condition
- If branch succeeds
  - PC is replaced by contents of a general register
RTN for SRC Shift Right

- **Abstract RTN**
  - \( \text{shr(:=op=26) } \rightarrow R[ra]<31..0> \leftarrow (n@0) \# R[rb]<31..n> \):
    - \( n := ((c3<4..0>=0) \rightarrow R[rc]<4..0>:\)
        - \( (c3<4..0)\neq 0 \rightarrow c3<4..0> :) \);  
      ;shift count in reg.
    - \( (c3<4..0)=0 \rightarrow R[rc]<4..0> :) \);  
      ;shift cnt const. field

- **Concrete RTN**

<table>
<thead>
<tr>
<th>Step</th>
<th>RTN</th>
</tr>
</thead>
<tbody>
<tr>
<td>T0–T2</td>
<td>Instruction Fetch</td>
</tr>
<tr>
<td>T3</td>
<td>( n \leftarrow IR&lt;4..0&gt; )</td>
</tr>
<tr>
<td>T4</td>
<td>( (n=0) \rightarrow (n \leftarrow R[rc]&lt;4..0&gt;) );</td>
</tr>
<tr>
<td>T5</td>
<td>( C \leftarrow R[rb] )</td>
</tr>
<tr>
<td>T6</td>
<td>( \text{Shr(:=n}\neq 0 \rightarrow (C&lt;31..0) \leftarrow 0#C&lt;31..1&gt; : n \leftarrow n-1; \text{ Shrt}) );</td>
</tr>
<tr>
<td>T7</td>
<td>( R[ra] \leftarrow C )</td>
</tr>
</tbody>
</table>

\( T6 \) is repeated \( n \) times
Notes on Shift RTN

- Abstract RTN defines n with :=
- Concrete RTN has n as a physical register
- n is not only the shift count but used as a counter in step T6
  - T6 is repeated n times through recursive Shr call
  - Will require more complicated control, described later
Datapath/Control Unit Separation

- Interface between datapath and control consists of gate and strobe signals
  - Gate – selects one of several values to apply to a common point (e.g. bus)
  - Strobe – changes the contents of a register (flip-flops) to match new inputs
- Type of flip-flop used in a register has significant impact on control and limited impact on datapath
  - Latch – simpler hardware but more complex timing
  - Edge triggered – simpler timing but approximately 2x hardware
Latch/Edge-Triggered Operation

- Latch output follows input while strobe is high

- Edge-triggering samples input at the edge time
More Complete View of 1-Bus SRC Design

- Add control signals and gate-level logic
Register File and Control Signals

- Register selection
  - IR decode of register fields
  - Grx signal to gate register rx by decoder
- R_out gates selected register onto the bus
- R_in strobes selected register from the bus
- Base address out BA_out gates zero signal when R[0] is selected
Extracting Constants/op from IR

- 3D blocks distinguish multi-bit elements
  - Register flip-flops
  - Tri-state bus drivers
- Sign bits fanned out from one to several bits and gated onto bus
  - $\text{IR}<21>$ is sign bit of $c_1$ and must be sign extended
  - $\text{IR}<16>$ is sign bit of $c_2$ and must be sign extended
Memory Interface

- **MD** is loaded from memory bus or from CPU bus
- **MD** can drive memory bus or CPU bus
- **MA** only gets address from CPU processor bus
ALU and Associated Registers

- Add control lines to select ALU function
  - INC4 for hardware supported PC increment
1-Bit ALU Logic-Level Design

PC increment

Negative numbers in B

AND gates select appropriate output
Control Sequences

- Register transfers are the concrete RTN
- Control sequence are the control signals that cause the RT

<table>
<thead>
<tr>
<th>Step</th>
<th>Concrete RTN</th>
<th>Control Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>T0</td>
<td>MA←PC: C←PC+4;</td>
<td>PC_{out}, MA_{in}, Inc4, C_{in}</td>
</tr>
<tr>
<td>T1</td>
<td>MD←M[MA]: PC←C</td>
<td>Read, C_{out}, PC_{in}, Wait</td>
</tr>
<tr>
<td>T2</td>
<td>IR←MD</td>
<td>MD_{out}, IR_{in}</td>
</tr>
<tr>
<td>T3</td>
<td>instruction_execution</td>
<td></td>
</tr>
</tbody>
</table>

Wait prevents control sequence from advancing to step T2 until memory asserts Done
Control Steps, Control Signals, and Timing

- Order control signals are written is irrelevant for a given time step
  - Step To:
    - \((\text{Inc}4, \ C_\text{in}, \ PC_{\text{out}}, \ MA_{\text{in}}) = (PC_{\text{out}}, \ MA_{\text{in}}, \ \text{Inc}4, \ C_\text{in})\)

- Timing distinction is made between gates and strobes
  - Gates early, strobes late in clock cycle

- Memory read should start as early as possible to reduce wait time

- \text{MA} must have correct value before being used for a read
Control for **ADD** Instruction

- \( \text{add}(:=\text{op}=12) \rightarrow R[ra] \leftarrow R[rb] + R[rc] \):

<table>
<thead>
<tr>
<th>Step</th>
<th>Concrete RTN</th>
<th>Control Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>T0</td>
<td>MA ← PC: C ← PC+4;</td>
<td>PC(<em>{\text{out}}), MA(</em>{\text{in}}), Inc4, C(_{\text{in}})</td>
</tr>
<tr>
<td>T1</td>
<td>MD ← M[MA]: PC ← C</td>
<td>Read, C(<em>{\text{out}}), PC(</em>{\text{in}}), Wait</td>
</tr>
<tr>
<td>T2</td>
<td>IR ← MD</td>
<td>MD(<em>{\text{out}}), IR(</em>{\text{in}})</td>
</tr>
<tr>
<td>T3</td>
<td>A ← R[rb]</td>
<td>Grb, R(<em>{\text{out}}), A(</em>{\text{in}})</td>
</tr>
<tr>
<td>T4</td>
<td>C ← A + R[rc];</td>
<td>Grc, R(<em>{\text{out}}), ADD, C(</em>{\text{in}})</td>
</tr>
<tr>
<td>T5</td>
<td>R[ra] ← C</td>
<td>C(<em>{\text{out}}), Gra, R(</em>{\text{in}}), End</td>
</tr>
</tbody>
</table>

- **Grx** used to gate correct 5-bit register select code
- **End** signals the control to start over at step To
RTN for **ADDI** Instruction

- \( \text{addi}(:=\text{op}=13) \rightarrow R[ra] \leftarrow R[rb] + c2<16..0> \) \{two's complement, sign-extend\}:

<table>
<thead>
<tr>
<th>Step</th>
<th>Concrete RTN</th>
<th>Control Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>T0</td>
<td>MA\leftarrow PC: C\leftarrow PC+4;</td>
<td>(\text{PC}<em>{\text{out}}, \text{MA}</em>{\text{in}}, \text{Inc}4, C_{\text{in}})</td>
</tr>
<tr>
<td>T1</td>
<td>MD\leftarrow M[MA]: PC\leftarrow C</td>
<td>\text{Read, C}<em>{\text{out}}, \text{PC}</em>{\text{in}}, \text{Wait}</td>
</tr>
<tr>
<td>T2</td>
<td>IR\leftarrow MD</td>
<td>(\text{MD}<em>{\text{out}}, \text{IR}</em>{\text{in}})</td>
</tr>
<tr>
<td>T3</td>
<td>A\leftarrow R[rb]</td>
<td>\text{Grb, R}<em>{\text{out}}, A</em>{\text{in}}</td>
</tr>
<tr>
<td>T4</td>
<td>C\leftarrow A+c2 {sign-extend};</td>
<td>(c2_{\text{out}}, \text{ADD}, C_{\text{in}})</td>
</tr>
<tr>
<td>T5</td>
<td>R[ra]\leftarrow C</td>
<td>(C_{\text{out}}, \text{Gra, R}_{\text{in}}, \text{End})</td>
</tr>
</tbody>
</table>

- \(c2_{\text{out}}\) **signal sign extends** \(\text{IR}<16..0>\) and gates it to the bus
RTN for \texttt{st} Instruction

- \texttt{st(:=op=3)\rightarrow M[\text{disp}]\leftarrow R[ra]}:
  - \text{disp<31..0>:=\((rb=0)\rightarrow c2<16..0>\) \{sign-extend\}:
    - \((rb\neq0)\rightarrow R[rb]+c2<16..0>\)\{sign-ext, 2's comp\}

<table>
<thead>
<tr>
<th>Step</th>
<th>Concrete RTN</th>
<th>Control Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>T0-T2</td>
<td>instruction_fetch</td>
<td></td>
</tr>
<tr>
<td>T3</td>
<td>A\leftarrow (rb=0\rightarrow 0: \text{rb}\neq0\rightarrow R[rb]);</td>
<td>Grb, BA\textsubscript{out}, A\textsubscript{in}</td>
</tr>
<tr>
<td>T4</td>
<td>C\leftarrow A+(16@IR&lt;16&gt;#IR&lt;15..0&gt;);</td>
<td>c2\textsubscript{out}, ADD, C\textsubscript{in}</td>
</tr>
<tr>
<td>T5</td>
<td>MA\leftarrow C;</td>
<td>C\textsubscript{out}, MA\textsubscript{in}</td>
</tr>
<tr>
<td>T6</td>
<td>MD\leftarrow R[ra]</td>
<td>Gra, R\textsubscript{out}, MD\textsubscript{in}, Write</td>
</tr>
<tr>
<td>T7</td>
<td>M[MA]\leftarrow MD;</td>
<td>Wait, End</td>
</tr>
</tbody>
</table>

- \textbf{Notice the use of } BA\textsubscript{out} \textbf{in step T3 not } R\textsubscript{out} \textbf{as done in } addi
Shift Counter

- Concrete RTN for \texttt{shr} relies upon a 5-bit register to hold the shift count
- Must load, decrement, and have a way to test if the contents equal 0
Control for Shift Instruction

- \( \text{shr}(:=\text{op}=26) \rightarrow R[ra]<31..0> \leftarrow (n \neq 0) \# R[rb]<31..n> : \)
  - \( n := ((c3<4..0> = 0) \rightarrow R[rc]<4..0> : (c3<4..0> \neq 0) \rightarrow c3<4..0> ) : \)
    - ;shift count in reg.
    - ;count const. field

<table>
<thead>
<tr>
<th>Step</th>
<th>Concrete RTN</th>
<th>Control Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>T0-T2</td>
<td>Instruction Fetch</td>
<td></td>
</tr>
<tr>
<td>T3</td>
<td>( n \leftarrow IR&lt;4..0&gt; )</td>
<td>clout, Ld</td>
</tr>
<tr>
<td>T4</td>
<td>((n=0) \rightarrow (n \leftarrow R[rc]&lt;4..0&gt;) ;)</td>
<td>(n=0 \rightarrow (Gr_c, R_{out}, Ld))</td>
</tr>
<tr>
<td>T5</td>
<td>( C \leftarrow R[rb] )</td>
<td>Gr_b, R_{out}, C=B, C_{in}</td>
</tr>
</tbody>
</table>
| T6   | \( \text{Shr}(:=n \neq 0 \rightarrow \) \( C<31..0> \leftarrow 0 \# C<31..1> ; \)
    | \( n \leftarrow n-1; \text{Shr} \) \) | \(n \neq 0 \rightarrow (C_{out}, \text{SHR}, C_{in}, \text{Decr}, Goto6)\) |
| T7   | \( R[ra] \leftarrow C \) | \(C_{out}, Gra, R_{in}, \text{End} \) |

- Conditional control signals and repeating control are new concepts
  - Goto6 – repeats step T6 but must be carefully timed for the looping
Branching

- Branch conditions dependent on cond field an a register value (not flag or flag register)
  - \( \text{cond} := ( \)
    - \( c3<2..0>=0 \rightarrow 0: \); never
    - \( c3<2..0>=1 \rightarrow 1: \); always
    - \( c3<2..0>=2 \rightarrow R[rc]=0: \); if register is zero
    - \( c3<2..0>=3 \rightarrow R[rc] \neq 0: \); if register is nonzero
    - \( c3<2..0>=4 \rightarrow R[rc]<31>=0: \); if register is positive or zero
    - \( c3<2..0>=5 \rightarrow R[rc]<31>=1: \); if register is negative
  - \( \text{Logic expression for condition} \)
    - \( \text{cond} = (c3<2..0>=1) \lor (c3<2..0>=2) \land (R[rc]=0) \lor (c3<2..0>=3) \land \neg (R[rc]=0) \lor (c3<2..0>=4) \land \neg R[rc]<31> \lor (c3<2..0>=5) \land R[rc]<31> \)
Conditional Value Computation

- NOR gate does test of $R[rc]=0$ on bus
Control for Branch Instruction

- \( br(\text{:=op}=8) \Rightarrow (\text{cond} \Rightarrow \text{PC} \leftarrow R[rb]) \):

<table>
<thead>
<tr>
<th>Step</th>
<th>Concrete RTN</th>
<th>Control Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>T0-T2</td>
<td>Instruction Fetch</td>
<td></td>
</tr>
<tr>
<td>T3</td>
<td>( \text{CON} \leftarrow \text{cond}(R[rc]) );</td>
<td>( \text{Gr}, \text{R}<em>{\text{out}}, \text{CON}</em>{\text{in}} )</td>
</tr>
<tr>
<td>T4</td>
<td>( \text{CON} \Rightarrow \text{PC} \leftarrow R[rb] );</td>
<td>( \text{Grb}, \text{R}<em>{\text{out}}, \text{CON} \rightarrow \text{PC}</em>{\text{in}}, \text{End} )</td>
</tr>
</tbody>
</table>

- Condition logic always connected to \( \text{CON} \)
  - \( R[rc] \) only needs to be placed on bus in T3
- Only \( \text{PC}_{\text{in}} \) is conditional in T4 since gating \( R[rb] \) to bus makes no difference if it is not used
Summary of Design Process

- Informal description $\Rightarrow$ formal RTN description $\Rightarrow$ block diagram arch. $\Rightarrow$ concrete RTN steps $\Rightarrow$ hardware design of blocks $\Rightarrow$ control sequences $\Rightarrow$ control unit and timing

- At each level, more decisions must be made
  - These decisions refine the design
  - Also place requirements on hardware still to be designed

- The nice one way process above has circularity
  - Decisions at later stages cause changes in earlier ones
  - Happens less in a text than in reality because
    - Can be fixed on re-reading
    - Confusing to first time student