CPE300: Digital System Architecture and Design

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

1-Bus Architecture and Datapath
10262011

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

• 1-Bus Microarchitecture and Datapath Review
• 1-Bus Logic Design
• Control Unit
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
1-Bus SRC Microarchitecture

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

- Concrete RTN adds detail to the datapath

Figure 4.4

Figure 4.5

Figure 4.6

Figure 4.7, 4.8

Figure 4.9

Figure 4.10

Condition bit flip-flop

IR register logic and data paths

Shift counter register
RTN for ADD Instruction

- Develop steps to execute instruction
- Abstract RTN
  - (IR$\leftarrow$M[PC]: PC$\leftarrow$PC+4; instruction_execution);
  - Instruction_execution := (... 
    - add(:=op=12)$\rightarrow$R[ra]$\leftarrow$R[rb]+R[rc]:
    - ...);
- 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$\leftarrow$PC: C$\leftarrow$PC+4;</td>
</tr>
<tr>
<td>T1</td>
<td>MD$\leftarrow$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>
RTN for **ADD** Instruction

- Develop steps to execute instruction
- Abstract RTN
  - (IR ← M[PC]: PC ← PC+4; instruction_execution);
  - Instruction_execution := (...  
    • add(:=op=12) → R[ra] ← R[rb]+R[rc];  
    • ...);

- 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 ← PC: C ← PC+4;</td>
</tr>
<tr>
<td>T1</td>
<td>MD ← M[MA]: PC ← C</td>
</tr>
<tr>
<td>T2</td>
<td>IR ← MD</td>
</tr>
<tr>
<td>T3</td>
<td>A ← R[rb]</td>
</tr>
<tr>
<td>T4</td>
<td>C ← A+R[rc];</td>
</tr>
<tr>
<td>T5</td>
<td>R[ra] ← C</td>
</tr>
</tbody>
</table>
**RTN for Load/Store Instruction**

- **Abstract RTN**
  - \( \text{ld}(\text{:=op}=1) \rightarrow R[ra] \leftarrow M[\text{disp}] \):
  - \( \text{st}(\text{:=op}=3) \rightarrow M[\text{disp}] \leftarrow R[ra] \):
    - \( \text{disp}<31..0>:=(\text{rb}=0) \rightarrow c2<16..0> \) \{sign-extend\}:
      - \( \text{rb} \neq 0 \rightarrow R[\text{rb}]+c2<16..0>\) \{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 \leftarrow (\text{rb}=0 \rightarrow 0: \text{rb} \neq 0 \rightarrow R[\text{rb}] ) );</td>
<td></td>
</tr>
<tr>
<td>T4</td>
<td>( C \leftarrow A+(16@\text{IR}&lt;16&gt;#\text{IR}&lt;15..0&gt;) );</td>
<td></td>
</tr>
<tr>
<td>T5</td>
<td>( \text{MA} \leftarrow C );</td>
<td></td>
</tr>
<tr>
<td>T6</td>
<td>( \text{MD} \leftarrow M[\text{MA}] );</td>
<td>( \text{MD} \leftarrow R[ra] )</td>
</tr>
<tr>
<td>T7</td>
<td>( R[ra] \leftarrow \text{MD} );</td>
<td>( \text{M}[\text{MA}] \leftarrow \text{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**
  - \(\text{br}(:=\text{op}=8) \rightarrow (\text{cond} \rightarrow \text{PC} \leftarrow \text{R}[\text{rb}]):\)
    - \(\text{cond}:= (\)
      - \(\text{c3}<2..0>=0 \rightarrow 0;\) ;never
      - \(\text{c3}<2..0>=1 \rightarrow 1;\) ;always
      - \(\text{c3}<2..0>=2 \rightarrow \text{R}[\text{rc}]=0;\) ;if register is zero
      - \(\text{c3}<2..0>=3 \rightarrow \text{R}[\text{rc}]\neq 0;\) ;if register is nonzero
      - \(\text{c3}<2..0>=4 \rightarrow \text{R}[\text{rc}]<31>=0;\) ;if register is positive or zero
      - \(\text{c3}<2..0>=5 \rightarrow \text{R}[\text{rc}]<31>=1 \); ;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}(\text{R}[\text{rc}]));</td>
</tr>
<tr>
<td>T4</td>
<td>\text{CON} \rightarrow \text{PC} \leftarrow \text{R}[\text{rb}];</td>
</tr>
</tbody>
</table>

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

- $c3<2..0>$ are just 3 low order bits of IR
- $\text{cond}()$ is evaluated by combinational logic circuit having inputs $R[rc]$ and $c3<2..0>$
- One bit 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**
  - `shr(:=op=26) ⇒ R[ra]<31..0> ← (n@0) # R[rb]<31..n>:
    - `n:=((c3<4..0>=0) ⇒ R[rc]<4..0>:
      (c3<4..0)≠0) ⇒ c3<4..0> ):`; shift count in reg.
      ;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><code>n←IR&lt;4..0&gt;</code></td>
</tr>
<tr>
<td>T4</td>
<td><code>(n=0) ⇒ (n←R[rc]&lt;4..0&gt;)</code></td>
</tr>
<tr>
<td>T5</td>
<td><code>C←R[rb]</code></td>
</tr>
<tr>
<td>T6</td>
<td><code>Shr(:=n≠0 ⇒ (C&lt;31..0) ← 0#C&lt;31..1&gt;: n←n-1; Shr))</code></td>
</tr>
<tr>
<td>T7</td>
<td><code>R[ra] ← C</code></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 \texttt{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

Condition bit flip-flop
IR register logic and data paths
Shift counter register
Register File and Control Signals

- **Register selection**
  - IR decode of register fields
  - $Gr_x$ 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
  - IR<21> is sign bit of c1 and must be sign extended
  - IR<16> is sign bit of c2 and must be sign extended

Copyright © 2004 Pearson Prentice Hall, Inc.
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$\leftarrow$PC: C$\leftarrow$PC+4;</td>
<td>PC$<em>{out}$, MA$</em>{in}$, Inc4, C$_{in}$</td>
</tr>
<tr>
<td>T1</td>
<td>MD$\leftarrow$M[MA]: PC$\leftarrow$C</td>
<td>Read, C$<em>{out}$, PC$</em>{in}$, Wait</td>
</tr>
<tr>
<td>T2</td>
<td>IR$\leftarrow$MD</td>
<td>MD$<em>{out}$, IR$</em>{in}$</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 T0:
    \[ (\text{Inc4}, C_{in}, PC_{out}, MA_{in}) = (PC_{out}, MA_{in}, \text{Inc4}, C_{in}) \]

- 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

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

- add(:=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$\leftarrow$PC: C$\leftarrow$PC+4;</td>
<td>$PC_{out}$, $MA_{in}$, Inc4, $C_{in}$</td>
</tr>
<tr>
<td>T1</td>
<td>MD$\leftarrow$M[MA]: PC$\leftarrow$C</td>
<td>Read, $C_{out}$, $PC_{in}$, Wait</td>
</tr>
<tr>
<td>T2</td>
<td>IR$\leftarrow$MD</td>
<td>$MD_{out}$, $IR_{in}$</td>
</tr>
<tr>
<td>T3</td>
<td>A$\leftarrow$R[rb]</td>
<td>Grb, $R_{out}$, $A_{in}$</td>
</tr>
<tr>
<td>T4</td>
<td>C$\leftarrow$A+R[rc];</td>
<td>Grc, $R_{out}$, ADD, $C_{in}$</td>
</tr>
<tr>
<td>T5</td>
<td>R[ra]$\leftarrow$C</td>
<td>$C_{out}$, Gra, $R_{in}$, End</td>
</tr>
</tbody>
</table>

- **Grx** used to gate correct 5-bit register select code
- **End** signals the control to start over at step T0
**RTN for **ADDI **Instruction**

- \( \text{addi}(:=\text{op}=13) \rightarrow R[ra] \leftarrow R[rb]+c2<16..0> \ \{\text{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>PC_{out}, MA_{in}, Inc4, C_{in}</td>
</tr>
<tr>
<td>T1</td>
<td>MD \leftarrow M[MA]; PC \leftarrow C</td>
<td>Read, C_{out}, PC_{in}, Wait</td>
</tr>
<tr>
<td>T2</td>
<td>IR \leftarrow MD</td>
<td>MD_{out}, IR_{in}</td>
</tr>
<tr>
<td>T3</td>
<td>A \leftarrow R[rb]</td>
<td>Grb, R_{out}, A_{in}</td>
</tr>
<tr>
<td>T4</td>
<td>C \leftarrow A+c2 \ {\text{sign-extend}};</td>
<td>c2_{out}, ADD, C_{in}</td>
</tr>
<tr>
<td>T5</td>
<td>R[ra] \leftarrow C</td>
<td>C_{out}, Gra, R_{in}, End</td>
</tr>
</tbody>
</table>

- \( C2_{out} \) **signal sign extends** \( IR<16..0> \) **and gates it to the bus**
RTN for \texttt{st} Instruction

- \texttt{st}(:=op=3) \rightarrow M[\text{disp}] \leftarrow R[ra]:
  - disp<31..0>:=((rb=0) \rightarrow c2<16..0> \ {\text{sign-extend}}:
    - (rb\neq 0) \rightarrow R[rb]+c2<16..0>\{\text{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: rb\neq 0 \rightarrow R[rb]);</td>
<td>Grb, BA_out, A_in</td>
</tr>
<tr>
<td>T4</td>
<td>C\leftarrow A+(16@IR&lt;16&gt;@IR&lt;15..0&gt;);</td>
<td>c2_out, ADD, C_in</td>
</tr>
<tr>
<td>T5</td>
<td>MA\leftarrow C;</td>
<td>C_out, MA_in</td>
</tr>
<tr>
<td>T6</td>
<td>MD\leftarrow R[ra]</td>
<td>Gra, R_out, MD_in, Write</td>
</tr>
<tr>
<td>T7</td>
<td>M[MA]\leftarrow MD;</td>
<td>Wait, End</td>
</tr>
</tbody>
</table>

- Notice the use of BA\_out in step T3 not R\_out as done in addi
Shift Counter

- Concrete RTN for \( \text{shr} \) 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

- \(\text{shr}(:=\text{op}=26) \rightarrow R[ra]<31..0> \leftarrow (n@0) \# R[rb]<31..n>:\)
  - \(n:=((c3<4..0>=0) \rightarrow R[rc]<4..0>:\)
    - \((c3<4..0)\neq 0) \rightarrow c3<4..0>:\)
      ; 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 (Grc, R_{out}, Ld))</td>
</tr>
<tr>
<td>T5</td>
<td>(C \leftarrow R[rb])</td>
<td>Grb, R_{out}, C=B, C_{in}</td>
</tr>
</tbody>
</table>
| T6   | 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   | \(R[ra] \leftarrow C\) | \(C_{out}, \text{Gra}, R_{in}, \text{End}\) |

- 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
  - 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

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

<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>( Grc, R_{out}, \text{CON}_{in} )</td>
</tr>
<tr>
<td>T4</td>
<td>( \text{CON} \rightarrow \text{PC} \leftarrow R[rb] );</td>
<td>( Grb, R_{out}, \text{CON} \rightarrow \text{PC}_{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 \( \text{PC}_{in} \) is conditional in T4 since gating \( R[rb] \) to bus makes no difference if it is not used
Summary of Design Process

- Informal description ⇒ formal RTN description ⇒ block diagram arch. ⇒ concrete RTN steps ⇒ hardware design of blocks ⇒ control sequences ⇒ 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
Clocking the Datapath

- Register transfers result from information processing
  - Register transfer timing – register to register

- Level sensitive latch flip-flops in example

- \( t_{R2\text{valid}} \) is the period from begin of gate signal till inputs at R2 are valid

- \( t_{\text{comb}} \) is delay through combinational logic, such as ALU or cond logic
Signal Timing on the Datapath

- Several delays occur in getting data from R1 to R2
  - Gate delay through the 3-state bus driver—\( t_g \)
  - Worst case propagation delay on bus—\( t_{bp} \)
  - Delay through any logic, such as ALU—\( t_{comb} \)
  - Set up time for data to affect state of R2—\( t_{su} \)
- Data can be strobed into R2 after this time
  \[ t_{R2valid} = t_g + t_{bp} + t_{comb} + t_{su} \]
- Diagram shows strobe signal in the form for a latch. It must be high for a minimum time—\( t_w \)
- There is a hold time, \( t_h \), for data after strobe ends
Signal Timing and Minimum Clock Cycle

• A total latch propagation delay is the sum
  \[ T_l = t_{su} + t_w + t_h \]
  ▫ All above times are specified for latch
  ▫ \( t_h \) may be very small or zero

• The minimum clock period is determined by finding longest path from flip-flop output to flip-flop input
  ▫ This is usually a path through the ALU
  ▫ Conditional signals add a little gate delay
  ▫ Minimum clock period is
  \[ t_{min} = t_g + t_{bp} + t_{comb} + t_l \]
Consequences of Flip-Flop Type

- Flip-flop types (Appendix A.12)
  - Level-triggered (latch) – state can change while clock is high
  - Edge-triggered – state changes only on a clock transition (high-to-low or low-to-high)
  - Master-slave – breaks feedback from output/input of register allowing on a single state change per clock cycle
- During the high part of a strobe a latch changes its output
  - If this output can affect its input, an error can occur (feedback)
- This can influence even the kind of concrete RTs that can be written for a data path

- If the C register is implemented with latches, then
  \[ C \leftarrow C + MD; \]  
  is not legal
- If the C register is implemented with master-slave or edge triggered flip-flops, it is OK
The Control Unit

• Brain of a machine
• Datapath implementation led to control sequences to implement instructions
• Control unit will generate the control sequences
  ▫ Logic to enable control signal
  ▫ Timing of signals
• The control unit’s job is to generate the control signals in the proper sequence
• Things the control signals depend on
  ▫ The time step Ti
  ▫ The instruction op code (for steps other than T0, T1, T2)
  ▫ Some few datapath signals like CON, n=0, etc.
  ▫ Some external signals: reset, interrupt, etc. (to be covered)
• The components of the control unit are: a time state generator, instruction decoder, and combinational logic to generate control signals
Detailed Control Unit

Clock and control sequence

Instruction decode

Exception signals

Control signals for datapath

Clocking logic

Enable

Step generator

CountIn

Load

Reset

Master clock

Strt

Wait

Done

Counter

Control step decoder

OpCode

IR

Decoder

Other signals from the data path

CON

n = 0

...
Control Signal Encoder Logic

- Write equation describing control signal
  - Find all occurrences of control signal in entire set of control sequences
  - Equation implemented by digital logic gates

<table>
<thead>
<tr>
<th>Step</th>
<th>Fetch Control Sequence</th>
<th>Step</th>
<th>ADD Control Sequence</th>
<th>Step</th>
<th>ADDI Control Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>T0</td>
<td>PC_{out}, MA_{in}, Inc4, C_{in}</td>
<td>T3</td>
<td>Grb, R_{out}, A_{in}</td>
<td>T3</td>
<td>Grb, R_{out}, A_{in}</td>
</tr>
<tr>
<td>T1</td>
<td>Read, C_{out}, PC_{in}, Wait</td>
<td>T4</td>
<td>Grc, R_{out}, ADD, C_{in}</td>
<td>T4</td>
<td>c2_{out}, ADD, C_{in}</td>
</tr>
<tr>
<td>T2</td>
<td>MD_{out}, IR_{in}</td>
<td>T5</td>
<td>C_{out}, Gra, R_{in}, End</td>
<td>T5</td>
<td>C_{out}, Gra, R_{in}, End</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Step</th>
<th>SHR Control Sequence</th>
<th>Step</th>
<th>ST Control Sequence</th>
<th>Step</th>
<th>BR Control Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>T3</td>
<td>c_{1out}, Ld</td>
<td>T3</td>
<td>Grb, BA_{out}, A_{in}</td>
<td>T3</td>
<td>Grc, R_{out}, CON_{in}</td>
</tr>
<tr>
<td>T4</td>
<td>n=0 \rightarrow (Grc, R_{out}, Ld)</td>
<td>T4</td>
<td>c2_{out}, ADD, C_{in}</td>
<td>T4</td>
<td>Grb, R_{out}, CON \rightarrow PC_{in}, End</td>
</tr>
<tr>
<td>T5</td>
<td>Grb, R_{out}, C=B, C_{in}</td>
<td>T5</td>
<td>C_{out}, MA_{in}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T6</td>
<td>n\neq0 \rightarrow (C_{out}, SHR, C_{in}, Decr, Goto6)</td>
<td>T6</td>
<td>Gra, R_{out}, MD_{in}, Write</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T7</td>
<td>C_{out}, Gra, R_{in}, End</td>
<td>T7</td>
<td>Wait, End</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Control Signal Examples

<table>
<thead>
<tr>
<th>Step</th>
<th>Fetch Control Sequence</th>
<th>Step</th>
<th>ADD Control Sequence</th>
<th>Step</th>
<th>ADDI Control Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>T0</td>
<td>$P_{\text{out}}, M_{\text{in}}, \text{Inc}<em>4, C</em>{\text{in}}$</td>
<td>T3</td>
<td>$G_{\text{r}}, R_{\text{out}}, A_{\text{in}}$</td>
<td>T3</td>
<td>$G_{\text{r}}, R_{\text{out}}, A_{\text{in}}$</td>
</tr>
<tr>
<td>T1</td>
<td>$\text{Read}, C_{\text{out}}, P_{\text{in}}, \text{Wait}$</td>
<td>T4</td>
<td>$G_{\text{r}}, R_{\text{out}}, \text{ADD}, C_{\text{in}}$</td>
<td>T4</td>
<td>$c_{\text{out}}, \text{ADD}, C_{\text{in}}$</td>
</tr>
<tr>
<td>T2</td>
<td>$M_{\text{out}}, I_{\text{Rin}}$</td>
<td>T5</td>
<td>$C_{\text{out}}, \text{Gra}, R_{\text{in}}, \text{End}$</td>
<td>T5</td>
<td>$C_{\text{out}}, \text{Gra}, R_{\text{in}}, \text{End}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Step</th>
<th>SHR Control Sequence</th>
<th>Step</th>
<th>ST Control Sequence</th>
<th>Step</th>
<th>BR Control Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>T3</td>
<td>$c_{\text{out}}, \text{Ld}$</td>
<td>T3</td>
<td>$G_{\text{r}}, B_{\text{out}}, A_{\text{in}}$</td>
<td>T3</td>
<td>$G_{\text{r}}, R_{\text{out}}, \text{CON}_{\text{in}}$</td>
</tr>
<tr>
<td>T4</td>
<td>$n=0 \rightarrow (G_{\text{r}}, R_{\text{out}}, \text{Ld})$</td>
<td>T4</td>
<td>$c_{\text{out}}, \text{ADD}, C_{\text{in}}$</td>
<td>T4</td>
<td>$G_{\text{r}}, R_{\text{out}}, \text{CON}<em>{\text{in}} \rightarrow \text{PC}</em>{\text{in}}, \text{End}$</td>
</tr>
<tr>
<td>T5</td>
<td>$G_{\text{r}}, R_{\text{out}}, C=B, C_{\text{in}}$</td>
<td>T5</td>
<td>$C_{\text{out}}, M_{\text{Ain}}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T6</td>
<td>$n \neq 0 \rightarrow (C_{\text{out}}, \text{SHR}, C_{\text{in}}, \text{Decr}, \text{Goto6})$</td>
<td>T6</td>
<td>$\text{Gra}, R_{\text{out}}, M_{\text{Din}}, \text{Write}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T7</td>
<td>$C_{\text{out}}, \text{Gra}, R_{\text{in}}, \text{End}$</td>
<td>T7</td>
<td>$\text{Wait}, \text{End}$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- $\text{Gra} = T5 \cdot (\text{add} + \text{addi}) + T6 \cdot \text{st} + T7 \cdot \text{shr} + \ldots$

- **Use of datapath conditions**
  - $G_{\text{r}} = T4 \cdot \text{add} + T4 \cdot (n=0) \cdot \text{shr} + \ldots$
Branching in the Control Unit

- Tri-state gates allow 6 to be applied to counter input
- Reset will synchronously reset counter to step To
- Mck is the master clock oscillator signal
Clocking Logic

- Generates **Run** signal
- Generate synchronized done signal **SDone**
- Generates **R, W** from **Read, Write** control
- Generates **Enable** which controls counter
Completed 1-Bus Design

- High level architecture block diagram
- Concrete RTN steps
- Hardware design of registers and data path logic
- Revision of concrete RTN steps where needed
- Control sequences
- Register clocking decisions
- Logic equations for control signals
- Time step generator design
- Clock run, stop, and synchronization logic