Chapter 4: Processor Design

Topics

4.1 The Design Process
4.2 A 1-Bus Microarchitecture for the SRC
4.3 Data Path Implementation
4.4 Logic Design for the 1-Bus SRC
4.5 The Control Unit
4.6 The 2- and 3-Bus Processor Designs
4.7 The Machine Reset
4.8 Machine Exceptions
Abstract and Concrete Register Transfer Descriptions

- The abstract RTN for SRC in Chapter 2 defines “what,” not “how”
- A concrete RTN uses a specific set of real registers and buses to accomplish the effect of an abstract RTN statement
- Several concrete RTNs could implement the same ISA
A Note on the Design Process

- This chapter presents several SRC designs
- We started in Chapter 2 with an informal description
- In this chapter we will propose several block diagram architectures to support the abstract RTN, then we will:
  - Write concrete RTN steps consistent with the architecture
  - Keep track of demands made by concrete RTN on the hardware
- Design data path hardware and identify needed control signals
- Design a control unit to generate control signals
Fig 4.1  Block Diagram of 1-Bus SRC

CPU

Control Unit

Control signals out

Control unit inputs

Data Path

R0
32-bit general purpose registers

R31

A

B

ALU

C

(31..0)

31

PC

32

IR

Main memory

Input/output

Memory bus

Figures 4.2, 4.3

Figure 4.11

Fig 4.2 High-Level View of the 1-Bus SRC Design
Constraints Imposed by the Microarchitecture

- One bus connecting most registers allows many different RTs, but only one at a time
- Memory address must be copied into MA by CPU
- Memory data written from or read into MD
- First ALU operand always in A, result goes to C
- Second ALU operand always comes from bus
- Information only goes into IR and MA from bus
  - A decoder (not shown) interprets contents of IR
  - MA supplies address to memory, not to CPU bus
Abstract and Concrete RTN for SRC
add Instruction

Abstract RTN: (IR ← M[PC]: PC ← PC + 4; instruction_execution);
    instruction_execution := ( • • •
        add (:= op= 12) → R[ra] ← R[rb] + R[rc]:

Tbl 4.1 Concrete RTN for the add Instruction

<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>

- Parts of 2 RTs (IR ← M[PC]: PC ← PC + 4;) done in T0
- Single add RT takes 3 concrete RTs (T3, T4, T5)
Concrete RTN Gives Information About Sub-units

- The ALU must be able to add two 32-bit values
- ALU must also be able to increment B input by 4
- Memory read must use address from MA and return data to MD
- Two RTs separated by : in the concrete RTN, as in T0 and T1, are operations at the same clock
- Steps T0, T1, and T2 constitute instruction fetch, and will be the same for all instructions
- With this implementation, fetch and execute of the add instruction takes 6 clock cycles
Concrete RTN for Arithmetic Instructions: addi

Abstract RTN:
addi (:= op= 13) → R[ra] ← R[rb] + c2⟨16..0⟩
{2’s complement  sign extend} :

Concrete RTN for addi:

Step  RTN
T0.  MA ← PC:  C ← PC + 4;
T1.  MD ← M[MA]; PC ← C;
T2.  IR ← MD;  Instr Fetch
T3.  A ← R[rb];  Instr Execn.
T4.  C ← A + c2⟨16..0⟩ {sign ext.};
T5.  R[ra] ← C;

• Differs from add only in step T4
• Establishes requirement for sign extend hardware
Fig 4.3 More Complete View of Registers and Buses in the 1-Bus SRC Design, Including Some Control Signals

- Concrete RTN lets us add detail to the data path
  - Instruction register logic and new paths
  - Condition bit flip-flop
  - Shift count register

Keep this slide in mind as we discuss concrete RTN of instructions.
Abstract and Concrete RTN for Load and Store

\[
\text{ld} (\text{:= op= 1}) \rightarrow R[\text{ra}] \leftarrow M[\text{disp}] : \\
\text{st} (\text{:= op= 3}) \rightarrow M[\text{disp}] \leftarrow R[\text{ra}] :
\]

where

\[
\text{disp}^{31..0} := ((r_b=0) \rightarrow c2^{16..0} \{\text{sign \ ext.}\} : \\
(r_b\neq 0) \rightarrow R[r_b] + c2^{16..0} \{\text{sign extend, 2's comp.}\} ) :
\]

Tbl 4.3 The ld and St (load/store register from memory) Instructions

<table>
<thead>
<tr>
<th>Step</th>
<th>RTN for ld</th>
<th>RTN for st</th>
</tr>
</thead>
<tbody>
<tr>
<td>T0–T2</td>
<td>Instruction fetch</td>
<td></td>
</tr>
<tr>
<td>T3</td>
<td>A \leftarrow (r_b = 0 \rightarrow 0: r_b \neq 0 \rightarrow R[r_b]);</td>
<td></td>
</tr>
<tr>
<td>T4</td>
<td>C \leftarrow A + (16@ir^{16}#ir^{15..0});</td>
<td></td>
</tr>
<tr>
<td>T5</td>
<td>MA \leftarrow C;</td>
<td></td>
</tr>
<tr>
<td>T6</td>
<td>MD \leftarrow M[MA];</td>
<td>MD \leftarrow R[ra];</td>
</tr>
<tr>
<td>T7</td>
<td>R[ra] \leftarrow MD;</td>
<td>M[MA] \leftarrow MD;</td>
</tr>
</tbody>
</table>
Notes for Load and Store RTN

- Steps T0 through T2 are the same as for add and addi, and for all instructions

- In addition, steps T3 through T5 are the same for ld and st, because they calculate disp
- A way is needed to use 0 for R[rb] when rb = 0
- 15-bit sign extension is needed for IR\langle16..0\rangle

- Memory read into MD occurs at T6 of ld
- Write of MD into memory occurs at T7 of st
Concrete RTN for Conditional Branch

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

\[ \text{cond := ( } \begin{align*} 
& c3\langle 2..0 \rangle = 0 \rightarrow 0: & \text{never} \\
& c3\langle 2..0 \rangle = 1 \rightarrow 1: & \text{always} \\
& c3\langle 2..0 \rangle = 2 \rightarrow \text{R}[rc] = 0: & \text{if register is zero} \\
& c3\langle 2..0 \rangle = 3 \rightarrow \text{R}[rc] \neq 0: & \text{if register is nonzero} \\
& c3\langle 2..0 \rangle = 4 \rightarrow \text{R}[rc]\langle 31 \rangle = 0: & \text{if positive or zero} \\
& c3\langle 2..0 \rangle = 5 \rightarrow \text{R}[rc]\langle 31 \rangle = 1 : & \text{if negative} 
\end{align*} \]

Tbl 4.4 The Branch Instruction, br

<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>CON \leftarrow \text{cond} (\text{R}[rc]);</td>
</tr>
<tr>
<td>T4</td>
<td>CON \rightarrow \text{PC} \leftarrow \text{R}[rb];</td>
</tr>
</tbody>
</table>
Notes on Conditional Branch RTN

- $c3\langle 2..0 \rangle$ are just the low-order 3 bits of IR

- `cond()` is evaluated by a combinational logic circuit having inputs from R[rc] and $c3\langle 2..0 \rangle$

- The one bit register CON is not accessible to the programmer and only holds the output of the combinational logic for the condition

- If the branch succeeds, the program counter is replaced by the contents of a general register
Abstract and Concrete RTN for SRC Shift Right

\[ \text{shr (:= op = 26)} \rightarrow R[ra]_{31..0} \leftrightarrow (n @ 0) \# R[rb]_{31..n} : \]
\[ n := ( (c3_{4..0} = 0) \rightarrow R[rc]_{4..0} : \text{Shift count in register} \]
\[ (c3_{4..0} \neq 0) \rightarrow c3_{4..0} ) : \text{or constant field of instruction} \]

Tbl 4.5 The shr Instruction

<table>
<thead>
<tr>
<th>Step</th>
<th>Concrete RTN</th>
</tr>
</thead>
<tbody>
<tr>
<td>T0–T2</td>
<td>Instruction fetch</td>
</tr>
<tr>
<td>T3</td>
<td>( n \leftarrow IR\langle 4..0 \rangle ; )</td>
</tr>
<tr>
<td>T4</td>
<td>((n = 0) \rightarrow (n \leftarrow R[rc]_{4..0}) ; )</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\langle 31..0 \rangle \leftarrow 0 # C\langle 31..1 \rangle : n \leftarrow n - 1 ; \text{Shr} ))} ; )</td>
</tr>
<tr>
<td>T7</td>
<td>( R[ra] \leftarrow C ; )</td>
</tr>
</tbody>
</table>

step T6 is repeated n times