RTN (Register Transfer Notation)

- Provides a formal means of describing machine structure and function
- Is at the “just right” level for machine descriptions
- Does not replace hardware description languages
- Can be used to describe *what* a machine does (an abstract RTN) without describing *how* the machine does it
- Can also be used to describe a particular hardware implementation (a concrete RTN)
RTN (cont’d.)

- At first you may find this “meta description” confusing, because it is a language that is used to describe a language
- You will find that developing a familiarity with RTN will aid greatly in your understanding of new machine design concepts
- We will describe RTN by using it to describe SRC
Some RTN Features—
Using RTN to Describe a Machine’s
Static Properties

Static Properties

• Specifying registers
  • IR⟨31..0⟩ specifies a register named “IR” having 32 bits numbered 31 to 0
  • “Naming” using the := naming operator:
  • op⟨4..0⟩ := IR⟨31..27⟩ specifies that the 5 msbs of IR be called op, with bits 4..0
  • Notice that this does not create a new register, it just generates another name, or “alias,” for an already existing register or part of a register
Using RTN to Describe Dynamic Properties

Dynamic Properties

- Conditional expressions:
  \((\text{op}=12) \rightarrow \text{R}[ra] \leftarrow \text{R}[rb] + \text{R}[rc]:\) ; defines the add instruction

“if” condition “then” RTN Assignment Operator

This fragment of RTN describes the SRC add instruction. It says, “when the op field of IR = 12, then store in the register specified by the ra field, the result of adding the register specified by the rb field to the register specified by the rc field.”
Using RTN to Describe the SRC (Static) Processor State

Processor state

PC\langle31..0\rangle: program counter
(memory addr. of next inst.)
IR\langle31..0\rangle: instruction register
Run: one bit run/halt indicator
Start: start signal
R[0..31]\langle31..0\rangle: general purpose registers
RTN Register Declarations

- General register specifications shows some features of the notation
- Describes a set of 32 32-bit registers with names R[0] to R[31]

\[ R[0..31]^{\langle 31..0 \rangle} : \]

- Name of registers
- Register # in square brackets
- msb #
- lsb#
- .. specifies a range of indices
- Bit # in angle brackets
- Colon separates statements with no ordering
Memory Declaration: RTN Naming Operator

- Defining names with formal parameters is a powerful formatting tool
- Used here to define word memory (big-endian)

Main memory state

\[
\begin{align*}
\text{Mem}[0..2^{32} - 1][7..0] &: 2^{32} \text{ addressable bytes of memory}\\
M[x][31..0] &= \text{Mem}[x]\#\text{Mem}[x+1]\#\text{Mem}[x+2]\#\text{Mem}[x+3]\\
\end{align*}
\]

- Dummy parameter
- Naming operator
- Concatenation operator
- All bits in register if no bit index given
RTN Instruction Formatting Uses Renaming of IR Bits

Instruction formats

\[
\begin{align*}
\text{op\{4..0\} := IR\{31..27\}}: & \quad \text{operation code field} \\
\text{ra\{4..0\} := IR\{26..22\}}: & \quad \text{target register field} \\
\text{rb\{4..0\} := IR\{21..17\}}: & \quad \text{operand, address index, or branch target register} \\
\text{rc\{4..0\} := IR\{16..12\}}: & \quad \text{second operand, conditional test, or shift count register} \\
\text{c1\{21..0\} := IR\{21..0\}}: & \quad \text{long displacement field} \\
\text{c2\{16..0\} := IR\{16..0\}}: & \quad \text{short displacement or immediate field} \\
\text{c3\{11..0\} := IR\{11..0\}}: & \quad \text{count or modifier field}
\end{align*}
\]
Specifying Dynamic Properties of SRC: RTN Gives Specifics of Address Calculation

Effective address calculations (occur at runtime):

\[
\text{disp}\langle 31..0 \rangle := \begin{cases} (\text{rb}=0) & \rightarrow \text{c2}\langle 16..0 \rangle \{\text{sign extend}\} : \text{displacement} \\ (\text{rb} \neq 0) & \rightarrow R[\text{rb}] + \text{c2}\langle 16..0 \rangle \{\text{sign extend, 2’s comp.}\} : \text{address} \end{cases}
\]

\[
\text{rel}\langle 31..0 \rangle := \text{PC}\langle 31..0 \rangle + \text{c1}\langle 21..0 \rangle \{\text{sign extend, 2’s comp.}\} : \text{relative address}
\]

- Renaming defines displacement and relative addresses
- New RTN notation is used
  - condition → expression means if condition then expression
  - modifiers in { } describe type of arithmetic or how short numbers are extended to longer ones
  - arithmetic operators (+ - * / etc.) can be used in expressions
- Register R[0] cannot be added to a displacement
Detailed Questions Answered by the RTN for Addresses

- What set of memory cells can be addressed by direct addressing (displacement with rb=0)
  - If c2(16)=0 (positive displacement) absolute addresses range from 00000000H to 0000FFFFFFH
  - If c2(16)=1 (negative displacement) absolute addresses range from FFFF0000H to FFFFFFFFHF
- What range of memory addresses can be specified by a relative address
  - The largest positive value of C1(21..0) is $2^{21}-1$ and its most negative value is $-2^{21}$, so addresses up to $2^{21}-1$ forward and $2^{21}$ backward from the current PC value can be specified
- Note the difference between rb and R[rb]
Instruction Interpretation: RTN
Description of Fetch-Execute

- Need to describe actions (not just declarations)
- Some new notation

Logical NOT
Logical AND

instruction_interpretation := (  
Run ∧ Strt → Run ← 1:  
Run → (IR ← M[PC]: PC ← PC + 4; instruction_execution) );

Register transfer
Separates statements that occur in sequence
RTN Sequence and Clocking

- In general, RTN statements separated by `:` take place during the same clock pulse.
- Statements separated by `;` take place on successive clock pulses.
- This is not entirely accurate since some things written with one RTN statement can take several clocks to perform.
- More precise difference between `:` and `;`
  - The order of execution of statements separated by `:` does not matter.
  - If statements are separated by `;`, the one on the left must be complete before the one on the right starts.
More About Instruction Interpretation

RTN

- In the expression \( IR \leftarrow M[PC] ; \ PC \leftarrow PC + 4 \); which value of PC applies to \( M[PC] \)?
- The rule in RTN is that all right hand sides of "::"-separated RTs are evaluated before any LHS is changed
  - In logic design, this corresponds to "master-slave" operation of flip-flops
- We see what happens when Run is true and when Run is false but Strt is true. What about the case of Run and Strt both false?
  - Since no action is specified for this case, the RTN implicitly says that no action occurs in this case
Individual Instructions

- instruction_interpretation contained a forward reference to instruction_execution
- instruction_execution is a long list of conditional operations
  - The condition is that the op code specifies a given instruction
  - The operation describes what that instruction does
- Note that the operations of the instruction are done after (;) the instruction is put into IR and the PC has been advanced to the next instruction
RTN Instruction Execution for Load and Store Instructions

\[
\text{instruction\_execution} := ( \\
\text{ld (:= op= 1) } \rightarrow \text{ R[ra]} \leftarrow \text{ M[disp]}: \quad \text{load register} \\
\text{ldr (:= op= 2) } \rightarrow \text{ R[ra]} \leftarrow \text{ M[rel]}: \quad \text{load register relative} \\
\text{st (:= op= 3) } \rightarrow \text{ M[disp]} \leftarrow \text{ R[ra]}: \quad \text{store register} \\
\text{str (:= op= 4) } \rightarrow \text{ M[rel]} \leftarrow \text{ R[ra]}: \quad \text{store register relative} \\
\text{la (:= op= 5 ) } \rightarrow \text{ R[ra]} \leftarrow \text{ disp}: \quad \text{load displacement address} \\
\text{lar (:= op= 6) } \rightarrow \text{ R[ra]} \leftarrow \text{ rel}: \quad \text{load relative address}
\]

- The in-line definition (:= op=1) saves writing a separate definition \text{ld} := op=1 for the \text{ld} mnemonic
- The previous definitions of disp and rel are needed to understand all the details
SRC RTN—The Main Loop

\[

ew := \text{instruction\_interpretation:} \\
\text{ie := instruction\_execution:} \\
\new ::= ( \neg \text{Run} \land \text{Strt} \rightarrow \text{Run} \leftarrow 1: \\
\quad \text{Run} \rightarrow (\text{IR} \leftarrow M[\text{PC}]: \text{PC} \leftarrow \text{PC} + 4; \\
\quad \text{ie}) ); \\
\text{ie ::= (} \\
\quad \text{ld} (:= \text{op} = 1) \rightarrow \text{R}[\text{ra}] \leftarrow M[\text{disp}]; \quad \text{Big switch} \\
\quad \text{ldr} (:= \text{op} = 2) \rightarrow \text{R}[\text{ra}] \leftarrow M[\text{rel}]; \quad \text{statement} \\
\quad \ldots \quad \text{on the opcode} \\
\quad \text{stop} (:= \text{op} = 31) \rightarrow \text{Run} \leftarrow 0: \\
\quad ); \; \new
\]

Thus \new and \text{ie} invoke each other, as coroutines.
Use of RTN Definitions: Text Substitution Semantics

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

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

\[ \text{ld} (:= \text{op} = 1) \rightarrow R[ra] \leftarrow M[\]
\[ ((\text{rb} = 0) \rightarrow c2^{16..0} \{\text{sign extend}\}; \]
\[ (\text{rb} \neq 0) \rightarrow R[\text{rb}] + c2^{16..0} \{\text{sign extend, 2’s comp.}\} )]: \]

- An example:
  - If \( IR = 00001 \ 00101 \ 00011 \ 000000000000001011 \)
  - then \( \text{ld} \rightarrow R[5] \leftarrow M[ R[3] + 11 ]: \)
RTN Descriptions of SRC Branch Instructions

- Branch condition determined by 3 lsbs of instruction
- Link register (R[ra]) set to point to next instruction

cond := ( c3⟨2..0⟩=0 → 0: never 
          c3⟨2..0⟩=1 → 1: always 
          c3⟨2..0⟩=2 → R[rc]=0: if register is zero 
          c3⟨2..0⟩=3 → R[rc]≠0: if register is nonzero 
          c3⟨2..0⟩=4 → R[rc]⟨31⟩=0: if positive or zero 
          c3⟨2..0⟩=5 → R[rc]⟨31⟩=1 ): if negative 

br (:= op= 8) → (cond → PC ← R[rb]): conditional branch 
brl (:= op= 9) → (R[ra] ← PC: 
               cond → (PC ← R[rb]) ): branch and link
RTN for Arithmetic and Logic

add (:= op=12) → R[ra] ← R[rb] + R[rc]:
addi (:= op=13) → R[ra] ← R[rb] + c2⟨16..0⟩ {2's comp. sign ext.}:
sub (:= op=14) → R[ra] ← R[rb] - R[rc]:
neg (:= op=15) → R[ra] ← -R[rc]:
and (:= op=20) → R[ra] ← R[rb] ∧ R[rc]:
andi (:= op=21) → R[ra] ← R[rb] ∧ c2⟨16..0⟩ {sign extend}:
or (:= op=22) → R[ra] ← R[rb] ∨ R[rc]:
ori (:= op=23) → R[ra] ← R[rb] ∨ c2⟨16..0⟩ {sign extend}:
not (:= op=24) → R[ra] ← ¬R[rc]:

• Logical operators: and ∧ or ∨ and not ¬
RTN for Shift Instructions

- Count may be 5 lsbs of a register or the instruction
- Notation: @ - replication, # - concatenation

\[
\begin{align*}
n &:=( (c3<4..0>=0) \rightarrow R[rc]<4..0>:
(c3<4..0)!=0) \rightarrow c3<4..0> ):
shr &: (op=26) \rightarrow R[ra]<31..0> \leftarrow (n @ 0) \# R[rb]<31..n>:
shra &: (op=27) \rightarrow R[ra]<31..0> \leftarrow (n @ R[rb]<31>) \# R[rb]<31..n>:
shl &: (op=28) \rightarrow R[ra]<31..0> \leftarrow R[rb]<31-n..0> \# (n @ 0):
shc &: (op=29) \rightarrow R[ra]<31..0> \leftarrow R[rb]<31-n..0> \# R[rb]<31..32-n>:
\end{align*}
\]
Example of Replication and Concatenation in Shift

- Arithmetic shift right by 13 concatenates 13 copies of the sign bit with the upper 19 bits of the operand

```
shra r1, r2, 13

R[2]= 1001 0111 1110 1010 1110 1100 0001 0110


R[1]= 1111 1111 1111 1 100 1011 1111 0101 0111
```
Assembly Language for Shift

- Form of assembly language instruction tells whether to set c3=0

```
shr ra, rb, rc       ;Shift rb right into ra by 5 lsbs of rc
shr ra, rb, count   ;Shift rb right into ra by 5 lsbs of inst
shra ra, rb, rc     ;AShift rb right into ra by 5 lsbs of rc
shra ra, rb, count  ;AShift rb right into ra by 5 lsbs of inst
shl ra, rb, rc      ;Shift rb left into ra by 5 lsbs of rc
shl ra, rb, count   ;Shift rb left into ra by 5 lsbs of inst
shc ra, rb, rc      ;Shift rb circ. into ra by 5 lsbs of rc
shc ra, rb, count   ;Shift rb circ. into ra by 5 lsbs of inst
```
End of RTN Definition of instruction_execution

nop (:= op= 0) → :
stop (:= op= 31) → Run ← 0:
);
instruction_interpretation.

- We will find special use for nop in pipelining
- The machine waits for Strt after executing stop
- The long conditional statement defining instruction_execution ends with a direction to go repeat instruction_interpretation, which will fetch and execute the next instruction (if Run still =1)
Confused about RTN and SRC?

- SRC is a *Machine Language*
  - It can be interpreted by either hardware or software simulator.
- RTN is a *Specification Language*
  - Specification languages are languages that are used to specify other languages or systems—a *metalanguage*.
  - Other examples: LEX, YACC, VHDL, Verilog

Figure 2.10 may help clear this up...
Fig 2.10 The Relationship of RTN to SRC

SRC specification written in RTN

RTN compiler

Generated processor

SRC program or simulator

SRC program and data

Data output
A Note About Specification Languages

- They allow the description of *what* without having to specify *how*.
- They allow precise and unambiguous specifications, unlike natural language.
- They reduce errors:
  - Errors due to misinterpretation of imprecise specifications written in natural language.
  - Errors due to confusion in design and implementation—“human error.”
- Now the designer must debug the specification!
- Specifications can be automatically checked and processed by tools.
  - An RTN specification could be input to a simulator generator that would produce a simulator for the specified machine.
  - An RTN specification could be input to a compiler generator that would generate a compiler for the language, whose output could be run on the simulator.
Addressing Modes Described in RTN (Not SRC)

<table>
<thead>
<tr>
<th>Mode name</th>
<th>Assembler Syntax</th>
<th>RTN meaning</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register</td>
<td>Ra</td>
<td>R[t] ← R[a]</td>
<td>Tmp. Var.</td>
</tr>
<tr>
<td>Register indirect</td>
<td>(Ra)</td>
<td>R[t] ← M[R[a]]</td>
<td>Pointer</td>
</tr>
<tr>
<td>Immediate</td>
<td>#X</td>
<td>R[t] ← X</td>
<td>Constant</td>
</tr>
<tr>
<td>Indexed, based, or displacement</td>
<td>X(Ra)</td>
<td>R[t] ← M[X + R[a]]</td>
<td>Arrays, structs</td>
</tr>
<tr>
<td>Relative</td>
<td>X(PC)</td>
<td>R[t] ← M[X + PC]</td>
<td>Vals stored w pgm</td>
</tr>
<tr>
<td>Autoincrement</td>
<td>(Ra)+</td>
<td>R[t] ← M[R[a]]; R[a] ← R[a] + 1</td>
<td>Sequential</td>
</tr>
<tr>
<td>Autodecrement</td>
<td>- (Ra)</td>
<td>R[a] ← R[a] - 1; R[t] ← M[R[a]]</td>
<td>access.</td>
</tr>
</tbody>
</table>