מבוא למחשבים
Lecture 3

Abstract RTN

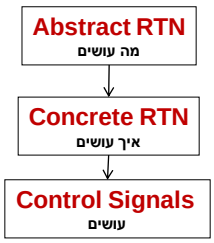
Dr. Ron Shmueli

חלק נכבד מהשקפים מבוסס על הספר:

- Heuring and Jordan: "Computer System Design and Architecture", **Prentice Hall**, 2004

RTN (Register Transfer Notation)

- Provides a formal means of describing machine structure and function.
  - Describe *what* a machine does (an Abstract RTN) without
  - Describe a *how* the machine does it (A Concrete RTN).
- We will describe RTN by using it to describe SRC.



סימונים של ה- RTN

←	העברה בכיוון המסומן R[4]←R[3] השמה	; :	מפריד -ביצוע אחת אחר השנייה מפריד -ביצוע בו זמנית באותו פולס שעון
[ ]	אינדקס מילה/אוגר M[x] M[100]←r[4]	@ #	שיכפול שרשור
<> <m..n >	קבוצת סיביות :IR<31..0> Op<4..0>:=IR<31..27>	{..}	הבהרה מילולית שקשורה לפעולה {דוגמא:משלים ל 2}
:=	השמה הגדרת תיוג Op<4..0>:=IR<31..27>	(..)	פעולה או אוסף פעולות
→	If then (op=12)→ R(ra)←R(rb)+R(rc)		פעולות השוואה שתוצאתן 0 או 1 פעולות אריתמטיות פעולות לוגיות.

Some RTN Features—
Using RTN to describe a machine's properties

Static Properties

Specifying registers

- IR⟨31..0⟩ specifies a register named "IR" having 32 bits numbered 31 to 0

- naming operator " := "
  - op⟨4..0⟩ := IR⟨31..27⟩ generates another name op⟨4..0⟩ to IR<31..28>

Dynamic Properties

- Conditional expressions:

  (op=12) → R[ra] ← R[rb] + R[rc]: ; defines the add instruction

"if" condition "then" RTN Assignment Operator

Using RTN to describe the SRC (static) Processor State

Processor state

PC⟨31..0⟩:	program counter (memory addr. of next inst.)
IR⟨31..0⟩:	instruction register
Run:	one bit run/halt indicator
Strt:	start signal
R[0..31]⟨31..0⟩:	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]⟨31..0⟩:

Name of registers

Register # in square brackets

msb #

.. specifies a range of indices

lsb#

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)

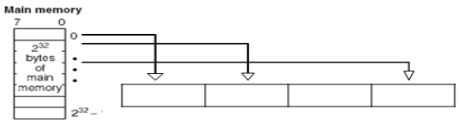
little-endian: Increasing numeric significance with increasing memory addresses.
big-endian: Decreasing numeric significance with increasing memory addresses

Main memory state

Mem[0..2^{32} - 1]⟨7..0⟩: 2^{32} addressable bytes of memory

M[x]⟨31..0⟩ := Mem[x]#Mem[x+1]#Mem[x+2]#Mem[x+3]:

Dummy parameter — Naming operator — Concatenation operator — All bits in register if no bit index given

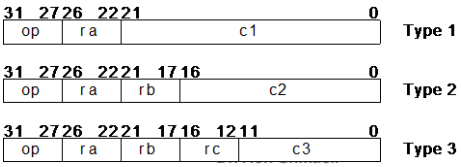


2013 7

RTN Instruction Formatting Uses Renaming of IR Bits

Instruction formats

op⟨4..0⟩ := IR⟨31..27⟩: operation code field
ra⟨4..0⟩ := IR⟨26..22⟩:target register field
rb⟨4..0⟩ := IR⟨21..17⟩:operand, address index, or branch target register
rc⟨4..0⟩ := IR⟨16..12⟩:second operand, conditional test, or shift count register
c1⟨21..0⟩ := IR⟨21..0⟩: long displacement field
c2⟨16..0⟩ := IR⟨16..0⟩: short displacement or immediate field
c3⟨11..0⟩ := IR⟨11..0⟩:count or modifier field



2013 8

Specifying dynamic properties of SRC: RTN Gives Specifics of Address Calculation

Effective address calculations (occur at runtime):

disp⟨31..0⟩ := ((rb=0) → c2⟨16..0⟩ {sign extend}: displacement
(rb≠0) → R[rb] + c2⟨16..0⟩ {sign extend, 2's comp.}): address
rel⟨31..0⟩ := PC⟨31..0⟩ + c1⟨21..0⟩ {sign extend, 2's comp.}: relative address

- New RTN notation is used
  - condition → expression means <u>if</u> condition <u>then</u> 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

דוגמא

Instruction	op	ra	rb	c1	Meaning	Addressing Mode
ldr r12, -48	2	12	–	-48	R[12] ← M[PC -48]	Relative
ld r22, 24(r4)	1	22	4	24	R[22] ← M[24+R[4]]	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 0000FFFFH
  - If c2⟨16⟩=1 (negative displacement) absolute addresses range from FFFF0000H to FFFFFFFFH
- 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)

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

(:) The order of execution does not matter
(;) The one on the left must be complete before the one on the right starts

IR ← M[PC]: PC ← PC + 4; which value of PC applies to M[PC] ?

RTN Instruction Execution for Load and Store Instructions

instruction_execution := (
ld (:= op= 1) → R[ra] ← M[disp]: load register
ldr (:= op= 2) → R[ra] ← M[rel]: load register relative
st (:= op= 3) → M[disp] ← R[ra]: store register
str (:= op= 4) → M[rel] ← R[ra]: store register relative
la (:= op= 5) → R[ra] ← disp:load displacement address
lar (:= op= 6) → R[ra] ← rel: load relative address

- The in-line definition (:= op=1) saves writing a separate definition ld := op=1 for the ld mnemonic

An example:
If IR = 00001 00101 00011 00000000000001011
then ld → R[5] ← M[R[3] + 11]:

SRC RTN—The Main Loop

ii := instruction_interpretation:
ie := instruction_execution :

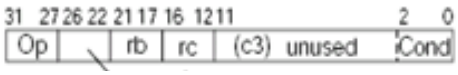
```
ii := ( ¬Run∧Strt → Run ← 1:
        Run → (IR ← M[PC]: PC ← PC + 4;
        ie) );
ie := (
    ld (:= op= 1) → R[ra] ← M[disp]:
    ldr (:= op= 2) → R[ra] ← M[rel]:
    . . .
    stop (:= op= 31) → Run ← 0:
);  ii
```

Big switch statement on the opcode

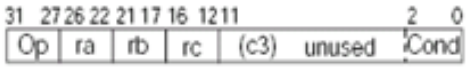
Thus ii and ie invoke each other, as coroutines.

RTN Descriptions of SRC Branch Instructions

4. br



5. brl



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

```
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
```

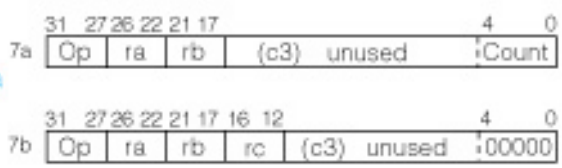
RTN for Arithmetic and Logic

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: <u>and</u> ∧ <u>or</u> ∨ and <u>not</u> ¬

RTN for Shift Instructions

7, shr, shra
shl, shc



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

n := ((c3⟨4..0⟩=0) → R[rc]⟨4..0⟩:
(c3⟨4..0⟩≠0) → c3⟨4..0⟩):
shr (:= op=26) → R[ra]⟨31..0⟩ ← (n @ 0) # R[rb]⟨31..n⟩:
shra (:= op=27) → R[ra]⟨31..0⟩ ← (n @ R[rb]⟨31⟩) # R[rb]⟨31..n⟩:
shl (:= op=28) → R[ra]⟨31..0⟩ ← R[rb]⟨31-n..0⟩ # (n @ 0):
shc (:= op=29) → R[ra]⟨31..0⟩ ← R[rb]⟨31-n..0⟩ # R[rb]⟨31..32-n⟩:

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

	13@R[2]⟨31⟩ #	R[2]⟨31..13⟩
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) → :	No operation
stop (:= op= 31) → Run ← 0:	Stop instruction
);	End of instruction_execution
instruction_interpretation.	

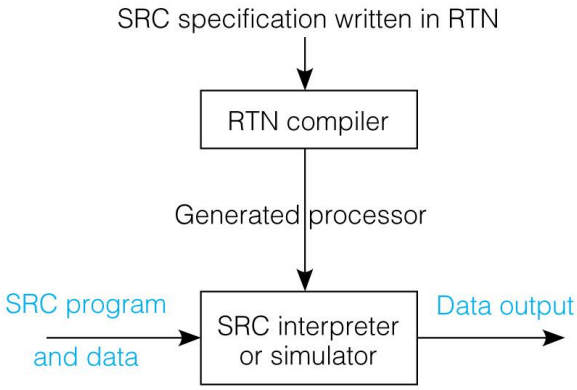
- We will find special use for nop in pipelining
- The machine waits for Strt after executing stop
- The long conditional statement defining instruction_execution ends with a direction to go repeat instruction_interpretation, which will fetch and execute the next instruction (if Run still =1)

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 2.11 may help clear this up...

Fig 2.11 The Relationship of RTN to SRC



From Abstract RTN to Concrete RTN to Control Sequences

- The ability to begin with an abstract description, then describe a hardware design and resulting concrete RTN and control sequence is powerful.
- We shall use this method in Chapter 4 to develop various hardware designs for SRC