מבוא למחשבים
Lecture 4

Concrete RTN
Control Signals

Dr. Ron Shmueli

חלק נכבד מהשקפים מבוסס על הספר:

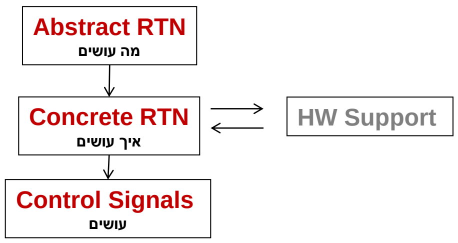
·Heuring and Jordan: "Computer System Design and Architecture", **Prentice Hall**, 2004

Chapter 4 Topics

- The Design Process
- A 1-bus Microarchitecture for SRC
- Data Path Implementation
- Logic Design for the 1-bus SRC
- The Control Unit
- The 2- and 3-bus Processor Designs
- The Machine Reset Process
- Machine Exceptions

Abstract and Concrete Register Transfer Descriptions

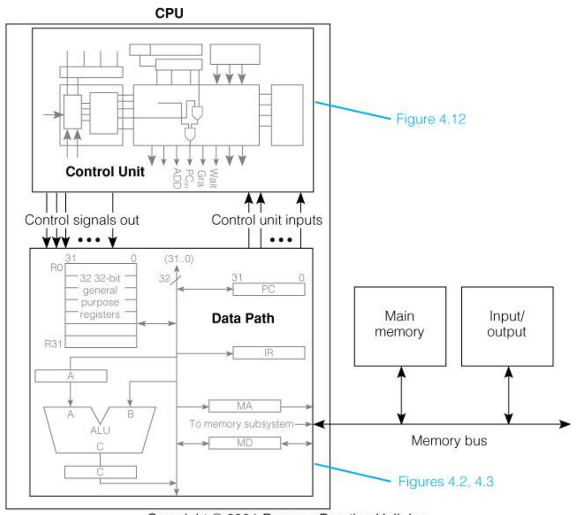
- The abstract RTN for SRC in Chapter 2 defines “what,” not “how”
- A concrete RTN uses a specific set of real registers and buses to accomplish the effect of an abstract RTN statement
- Several concrete RTNs could implement the same ISA



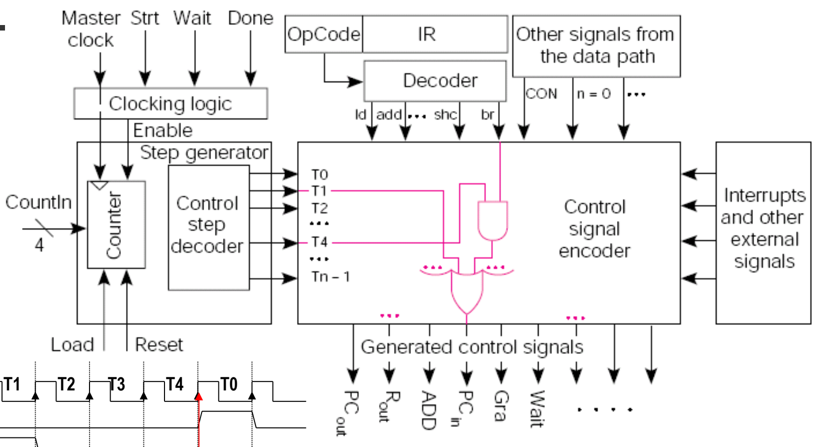
The Design Process

- informal description - Abstract RTN
- Block diagram architectures to support the abstract RTN, (HW)
  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



Control Unit (Time steps & cmd. Decoding)



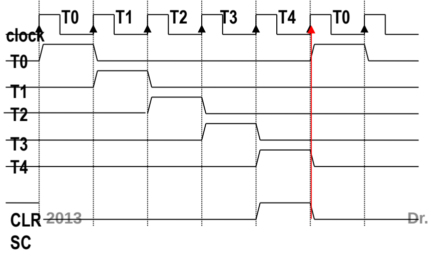


Fig. 4.2 High-Level View of the 1-Bus SRC Design

- One bus limitations
- Registers Access
- Memory data and Address
- ALU capabilities
  - first operand - always in A,
  - Second operand from bus
  - result goes to C

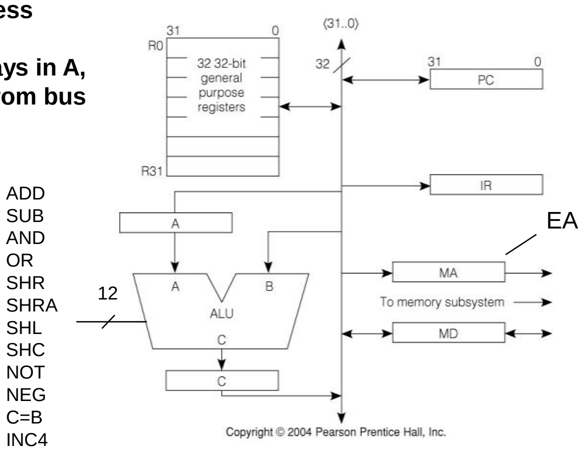
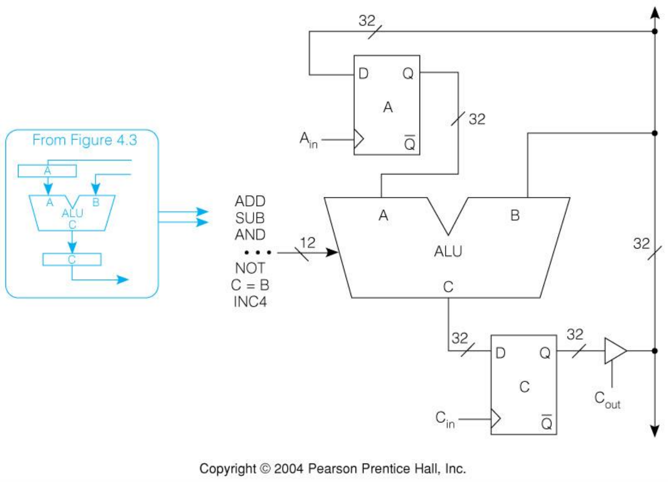


Fig. 4.7 The ALU and Its Associated Registers

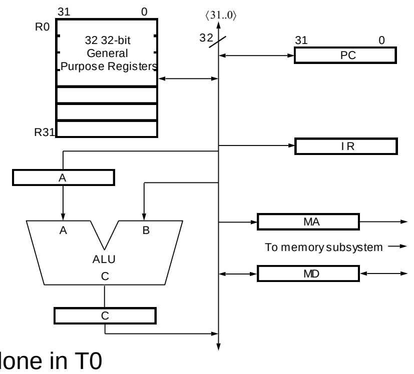


Abstract and Concrete RTN for SRC add Instruction

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 add:

Step	RTN	
T0.	MA ← PC: C ← PC + 4;	
T1.	MD ← M[MA]: PC ← C;	
T2.	IR ← MD;	IF
T3.		IEx.
T4.		
:		



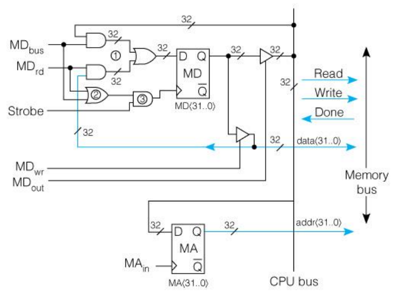
- Parts of 2 RTs (IR ← M[PC]: PC ← PC + 4;) done in T0
-

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
- 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

MD – Mem. Data

- אותות התפעול של ה MD בכתיבת ה – (Control signals
  - העברת נתון מ- CPU BUS ל- MD
    - מידע זמין בערוץ
    - MDin דוגם מידע ל MD (Mdbus + Strob פנימי ביחידת בקרה)
  - העברת נתון מ-MD ל- CPU BUS
    - מידע זמין ב MD
    - MDout (פתיחת חוצץ לערוץ).
  - כתיבה לזיכרון (העברה מ MD לזיכרון)
    - פתיחת ה MA
    - Write (MDwr פנימי ליחידת בקרה).
    - Wait (ממתין לאות done מהזיכרון ביחידת הבקרה)
  - קריאה מהזיכרון (העברה מהזיכרון ל MD)
    - פתיחת ה MA
    - Read (MDrd פנימי ליחידת בקרה).
    - Wait (כאשר מגיעה האות done מהזיכרון המידע נדגם על MD חלק מיחידת הבקרה)
- (הפשטה של בעית תזמון -אוגר מוציא מידע בעלית שעון ומכניס מידע בירידת שעון)



From Concrete RTN to Control Signals: The Control Sequence

<u>Tbl 4.6—The Instruction Fetch</u>

Step	Concrete RTN	Control Sequence
T0.	MA ← PC: C ← PC+4;	PC_{out}, MA_{in}, Inc4, C_{in}
T1.	MD ← M[MA]: PC ← C;	Read, C_{out}, PC_{in}, Wait
T2.	IR ← MD;	MD_{out}, IR_{in}
T3.	Instruction_execution	

- The register transfers are the concrete RTN
- The control signals that cause the register transfers make up the control sequence
- Wait prevents the control from advancing to step T3 until the memory asserts Done

Control Steps, Control Signals, and Timing

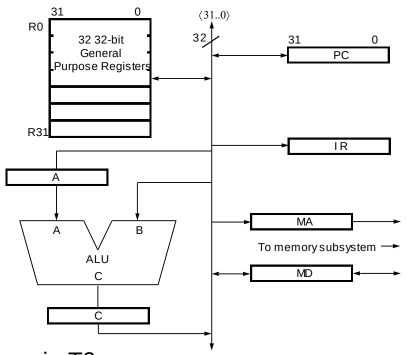
- Within a given time step, the order in which control signals are written is irrelevant
  - In step T0, C_{in}, Inc4, MA_{in}, PC_{out} == PC_{out}, MA_{in}, Inc4, C_{in}
- The only timing distinction within a step is between gates and strobes
- The memory read should be started as early as possible to reduce the wait
- MA must have the right value before being used for the read
- Depending on memory timing, Read could be in T0

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 add:

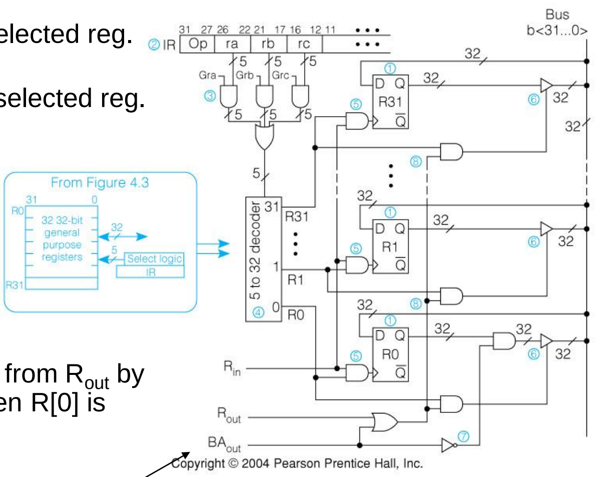
Step	RTN	
T0.	MA ← PC: C ← PC + 4;	
T1.	MD ← M[MA]: PC ← C;	
T2.	IR ← MD;	IF
T3.	A ← R[rb];	IEx.
T4.	C ← A + R[rc];	
T5.	R[ra] ← C;	



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

Fig. 4.4 The SRC Register File and Its Control Signals

- R_{out} gates selected reg. onto bus
- R_{in} strobed selected reg. from bus



- BA_{out} differs from R_{out} by gating 0 when R[0] is selected

BA = Base Address

Control Sequence for the SRC add Instruction

add (:= op= 12) $\rightarrow$ R[ra] $\leftarrow$ R[rb] + R[rc]:

Tbl 4.7 The Add Instruction

Step	Concrete RTN	Control Sequence
T0.	MA $\leftarrow$ PC: C $\leftarrow$ PC+4;	PC_{out}, MA_{in}, Inc4, C_{in}, Read
T1.	MD $\leftarrow$ M[MA]: PC $\leftarrow$ C;	C_{out}, PC_{in}, Wait
T2.	IR $\leftarrow$ MD;	MD_{out}, IR_{in}
T3.	A $\leftarrow$ R[rb];	Grb, R_{out}, A_{in}
T4.	C $\leftarrow$ A + R[rc];	Grc, R_{out}, ADD, C_{in}
T5.	R[ra] $\leftarrow$ C;	C_{out}, Gra, R_{in}, End

- Note the use of Gra, Grb, & Grc to gate the correct 5 bit register select code to the regs.
- End signals the control to start over at step T0

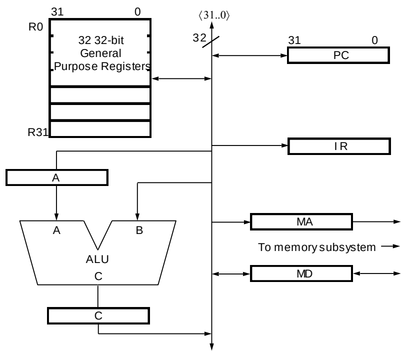
Concrete RTN for Arithmetic Instructions: addi

Abstract RTN:

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

Tbl 4.2 Concrete RTN for addi:

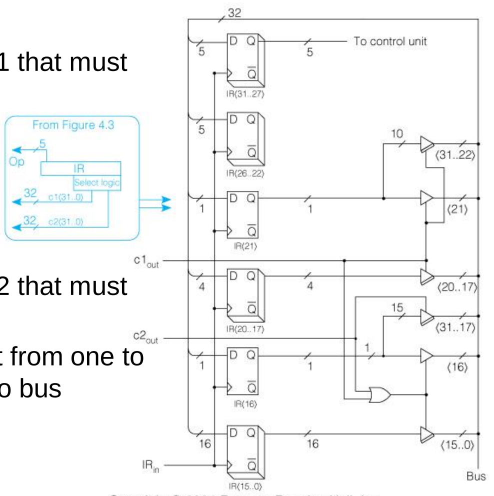
Step	RTN
T0.	MA ← PC: C ← PC + 4;
T1.	MD ← M[MA]; PC ← C;
T2.	IR ← MD;
T3.	A ← R[rb];
T4.	C ← A + c2⟨16..0⟩ {sign ext.};
T5.	R[ra] ← C;



- Differs from add only in step T4
- Establishes requirement for sign extend hardware

Fig. 4.5 Extracting c1, c2, and op from the Instruction Register

- I⟨21⟩ is the sign bit of C1 that must be extended
- I⟨16⟩ is the sign bit of C2 that must be extended
- Sign bits are fanned out from one to several bits and gated to bus



Control Sequence for the SRC addi Instruction

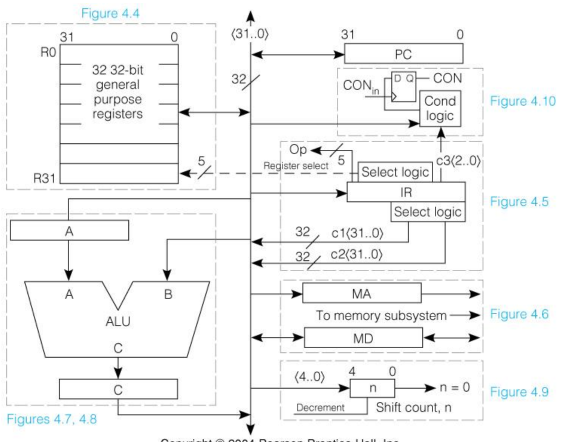
addi (:= op= 13) → R[ra] ← R[rb] + c2⟨16..0⟩ {2's comp., sign ext.} :

Tbl 4.8 The addi Instruction

Step	Concrete RTN	Control Sequence
T0.	MA ← PC: C ← PC + 4;	PC_{out}, MA_{in}, Inc4, C_{in}, Read
T1.	MD ← M[MA]; PC ← C;	C_{out}, PC_{in}, Wait
T2.	IR ← MD;	MD_{out}, IR_{in}
T3.	A ← R[rb];	Grb, R_{out}, A_{in}
T4.	C ← A + c2⟨16..0⟩ {sign ext.};	$c2_{out}$, ADD, C_{in}
T5.	R[ra] ← C;	C_{out}, Gra, R_{in}, End

- The $c2_{out}$ signal sign extends IR⟨16..0⟩ and gates it to the bus

Fig. 4.3 More Complete view of Registers and Buses in 1-bus SRC Design—Including Some Control Signals



- Concrete RTN lets us add detail to the data path
  - Instruction register logic & 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

ld (:= op= 1) → R[ra] ← M[disp] :
st (:= op= 3) → M[disp] ← R[ra] :
where
disp⟨31..0⟩ := ((rb=0) → c2⟨16..0⟩ {sign ext.} :
(rb≠0) → R[rb] + c2⟨16..0⟩ {sign extend, 2's comp.}) :

Tbl 4.3

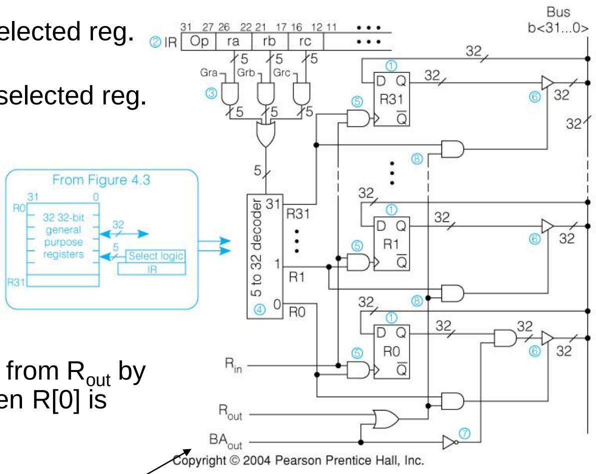
Step	RTN for ld	RTN for st
T0-T2	Instruction fetch	
T3.	A ← (rb=0 → 0: rb≠0 → R[rb]);	
T4.	C ← A + (16@IR⟨16⟩#IR⟨15..0⟩);	
T5.	MA ← C;	
T6.	MD ← M[MA];	MD ← R[ra];
T7.	R[ra] ← MD;	M[MA] ← MD;

Notes for Load and Store RTN

- Steps T0 through T2 are the same as for add and addi, and for <u>all instructions</u>
- 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 (BAout)
- 15 bit sign extension is needed for IR⟨16..0⟩
- Memory read into MD occurs at T6 of ld
- Write of MD into memory occurs at T7 of st

Fig. 4.4 The SRC Register File and Its Control Signals

- R_{out} gates selected reg. onto bus
- R_{in} strobed selected reg. from bus
- BA_{out} differs from R_{out} by gating 0 when R[0] is selected



BA = Base Address

Control Sequence for the SRC st Instruction

st (:= op= 3) → M[disp] ← R[ra] :
disp⟨31..0⟩ := ((rb=0) → c2⟨16..0⟩ {sign ext.} :
(rb≠0) → R[rb] + c2⟨16..0⟩ {sign extend, 2's comp.}) :

The st Instruction

Step	Concrete RTN	Control Sequence
T0-T2	Instruction fetch	Instruction fetch
T3.	A ← (rb=0) → 0: rb≠0 → R[rb];	Grb, BA_{out}, A_{in} } address arithmetic
T4.	C ← A + c2⟨16..0⟩ {sign ext.};	$c2_{out}$, ADD, C_{in} } address arithmetic
T5.	MA ← C;	C_{out}, MA_{in}
T6.	MD ← R[ra];	Gra, R_{out}, MD_{in}, Write
T7.	M[MA] ← MD;	Wait, End

- Note BA_{out} in T3 compared to R_{out} in T3 of addi

Concrete RTN for Conditional Branch

```
br (:= op= 8) → (cond → PC ← R[rb]):
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
```

Tbl 4.4

Step	Concrete RTN
T0-T2	Instruction fetch
T3.	CON ← cond(R[rc]);
T4.	CON → PC ← R[rb];

Notes on Conditional Branch RTN

- c3⟨2..0⟩ are just the low order 3 bits of IR
- cond() is evaluated by a combinational logic circuit having inputs from R[rc] and c3⟨2..0⟩
- 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 reg.

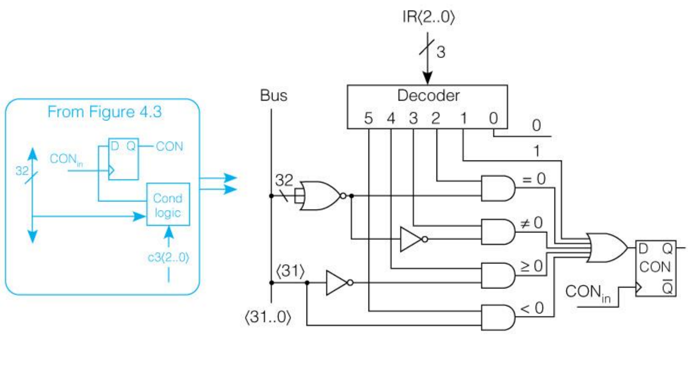
Branching

```
cond := ( c3⟨2..0⟩=0 → 0:
          c3⟨2..0⟩=1 → 1:
          c3⟨2..0⟩=2 → R[rc]=0:
          c3⟨2..0⟩=3 → R[rc]≠0:
          c3⟨2..0⟩=4 → R[rc]⟨31⟩=0:
          c3⟨2..0⟩=5 → R[rc]⟨31⟩=1 ):
```

- This is equivalent to the logic expression

cond = (c3⟨2..0⟩=1) ∨ (c3⟨2..0⟩=2)∧(R[rc]=0) ∨
(c3⟨2..0⟩=3)∧¬(R[rc]=0) ∨ (c3⟨2..0⟩=4)∧¬R[rc]⟨31⟩ ∨
(c3⟨2..0⟩=5)∧R[rc]⟨31⟩

Fig. 4.10 Computation of the Conditional Value CON



- NOR gate does =0 test of R[rc] on bus

Tbl 4.11 Control Sequence for SRC Branch Instruction, br

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

Step	Concrete RTN	Control Sequence
T0-T2	Instruction fetch	Instruction fetch
T3.	CON $\leftarrow$ cond(R[rc]);	Grc, R_{out}, CON_{in}
T4.	CON $\rightarrow$ PC $\leftarrow$ R[rb];	Grb, R_{out}, CON $\rightarrow$ PC_{in}, End

- Condition logic is always connected to CON, so R[rc] only needs to be put on bus in T3
- Only PC_{in} is conditional in T4 since gating R[rb] to bus makes no difference if it is not used

Abstract and Concrete RTN for SRC Shift Right

shr (:= op = 26) $\rightarrow$ R[ra]$\langle$31..0$\rangle$ $\leftarrow$ (n @ 0) # R[rb]$\langle$31..n$\rangle$:
n := ((c3$\langle$4..0$\rangle$=0) $\rightarrow$ R[rc]$\langle$4..0$\rangle$: shift count in reg.
(c3$\langle$4..0$\rangle$≠0) $\rightarrow$ c3$\langle$4..0$\rangle$): or const. field

Tbl 4.5

Step	Concrete RTN
T0-T2	Instruction fetch
T3.	n $\leftarrow$ IR$\langle$4..0$\rangle$;
T4.	(n=0) $\rightarrow$ (n $\leftarrow$ R[rc]$\langle$4..0$\rangle$);
T5.	C $\leftarrow$ R[rb];
T6.	Shr (:= (n≠0) $\rightarrow$ (C$\langle$31..0$\rangle$ $\leftarrow$ 0#C$\langle$31..1$\rangle$: n $\leftarrow$ n-1; Shr));
T7.	R[ra] $\leftarrow$ C;

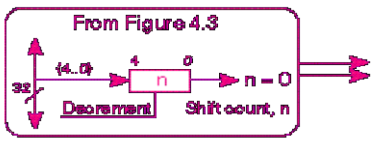
step T6 is repeated n times

Notes on SRC Shift RTN

- In the abstract RTN, n is defined with :=
- In the concrete RTN, it is a physical register
- n not only holds the shift count but is used as a counter in step T6
- Step T6 is repeated n times as shown by the recursion in the RTN
- The control for such repeated steps will be treated later

Fig. 4.9 The Shift Counter

- The concrete RTN for shr relies upon a 5 bit register to hold the shift count
- It must load, decrement, and have an = 0 test



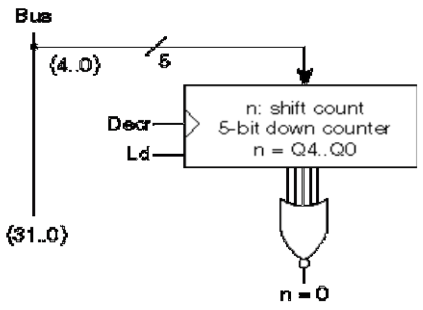
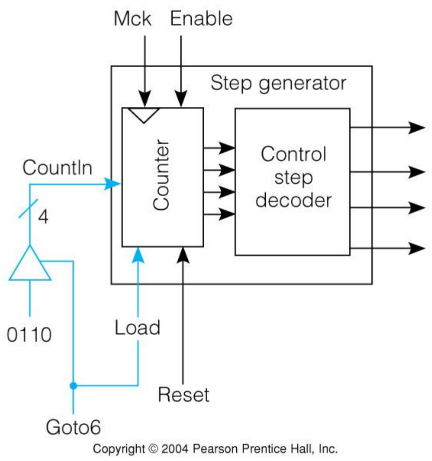


Fig. 4.14 Branching in the Control Unit



- 3-state gates allow 6 to be applied to counter input
- Reset will synchronously reset counter to step T0

Tbl 4.10 Control Sequence for the SRC shr Instruction—Looping

Step	Concrete RTN	Control Sequence
T0-T2	Instruction fetch	Instruction fetch
T3.	n ← IR⟨4..0⟩;	$c1_{out}$, Ld
T4.	(n=0) → (n ← R[rc]⟨4..0⟩);	n=0 → (Grc, R_{out}, Ld)
T5.	C ← R[rb];	Grb, R_{out}, C=B, C_{in}
T6.	Shr (:= (n≠0) → (C⟨31..0⟩ ← 0#C⟨31..1⟩: n ← n-1; Shr));	n≠0 → (C_{out}, SHR, C_{in}, Decr, Goto6)
T7.	R[ra] ← C;	C_{out}, Gra, R_{in}, End

- Conditional control signals and repeating a control step are new concepts

Summary of the Design Process

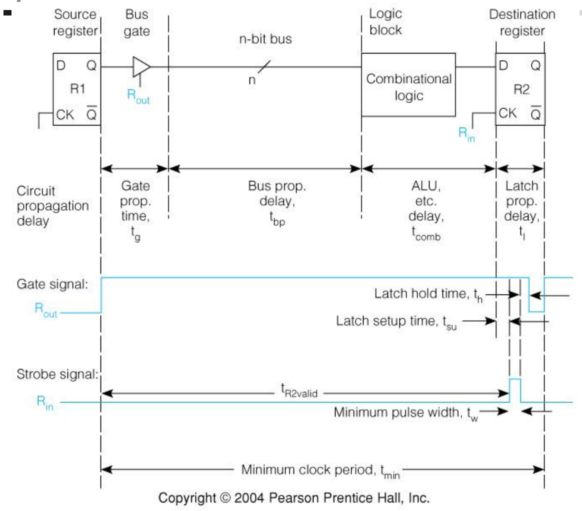
Informal description ⇒ formal RTN description ⇒ block diagram arch. ⇒ concrete RTN steps ⇒ hardware design of blocks ⇒ control sequences ⇒ control unit and timing

- 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

דוגמא

1. כתוב תוכנית asm המבצעת פעולת XOR (מצגת 2 שקף 37)
2. תכנן פקודת XOR חדשה המבצעת R[ra]← R[rb] xor R[rc] תן RTN קונקרטי ואותות בקרה.
   1. כאשר ניתן להוסיף יכולת XOR ל ALU
   2. כאשר לא ניתן להוסיף יכולת XOR ל ALU
3. מה זמן הביצוע בכל אחד מהמקרים (הסבר עלות תועלת)

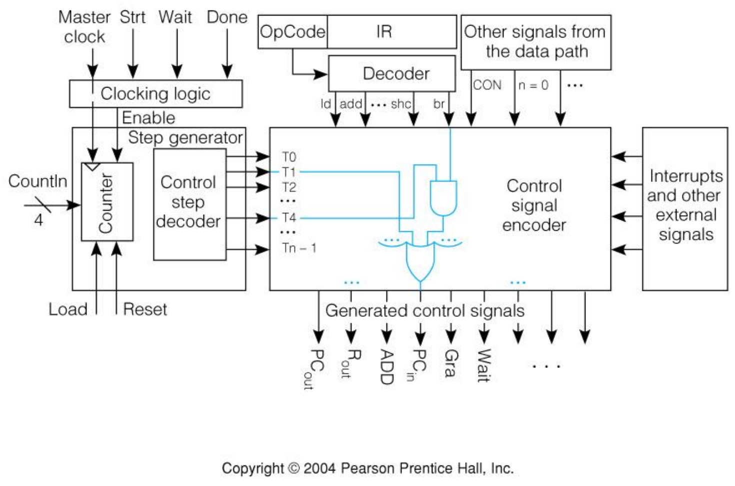
Fig. 4.11 Clocking the Data Path: Register Transfer Timing



- $t_{R2valid}$ is the period from begin of gate signal till inputs to R2 are valid
- t_{comb} is delay through combinational logic, such as ALU or cond logic

minimum clock period $t_{min} = t_g + t_{bp} + t_{comb} + t_{su} + t_w + t_h$

Fig. 4.12 Control Unit Detail with Inputs and Outputs


Master clock
Strt
Wait
Done
OpCode
IR
Other signals from the data path
Decoder
CON
n = 0
Clocking logic
Enable
Step generator
ld add ... shc br
CountIn
4
Counter
Control step decoder
T0 T1 T2 ... T4 ... Tn – 1
Control signal encoder
Interrupts and other external signals
Load
Reset
Generated control signals
PC_out R_out ADD PC_in Gra Wait ...
Copyright © 2004 Pearson Prentice Hall, Inc.

Synthesizing Control Signal Encoder Logic

Step	Control Sequence
T0.	PC_{out}, MA_{in}, Inc4, C_{in}, Read
T1.	C_{out}, PC_{in}, Wait
T2.	MD_{out}, IR_{in}

add		addi		st		shr	
Step	Control Sequence	Step	Control Sequence	Step	Control Sequence	Step	Control Sequence
T3.	Grb, R_{out}, A_{in}	T3.	Grb, R_{out}, A_{in}	T3.	Grb, BA_{out}, A_{in}	T3.	$c1_{out}$, Ld
T4.	Grc, R_{out}, ADD, C_{in}	T4.	$c2_{out}$, ADD, C_{in}	T4.	$c2_{out}$, ADD, C_{in}	T4.	n=0 → (Grc, R_{out}, Ld)
T5.	C_{out}, **Gra,** R_{in}, End	T5.	C_{out}, **Gra,** R_{in}, End	T5.	C_{out}, MA_{in}	T5.	Grb, R_{out}, C=B
				T6.	**Gra,** R_{out}, MD_{in}, Write	T6.	n≠0 → (C_{out}, SHR, C_{in}, Decr, Goto7)
				T7.	Wait, End	T7.	C_{out}, **Gra,** R_{in}, End

• • •

Design process:

- Comb through the entire set of control sequences.
- Find all occurrences of each control signal.
- Write an equation describing that signal.

Example: Gra = T5·(add + addi) + T6·st + T7·shr + ...

Use of Data Path Conditions in Control Signal Logic

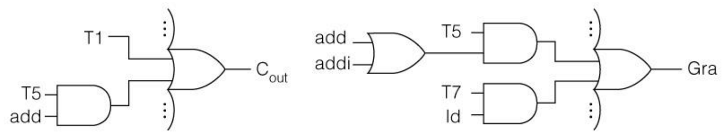
Step	Control Sequence
T0.	PC_{out}, MA_{in}, Inc4, C_{in}, Read
T1.	C_{out}, PC_{in}, Wait
T2.	MD_{out}, IR_{in}

add		addi		st		shr	
Step	Control Sequence	Step	Control Sequence	Step	Control Sequence	Step	Control Sequence
T3.	Grb, R_{out}, A_{in}	T3.	Grb, R_{out}, A_{in}	T3.	Grb, BA_{out}, A_{in}	T3.	$c1_{out}$, Ld
T4.	**Grc,** R_{out}, ADD, C_{in}	T4.	$c2_{out}$, ADD, C_{in}	T4.	$c2_{out}$, ADD, C_{in}	T4.	**n=0 → (Grc,** R_{out}, Ld)
T5.	C_{out}, Gra, R_{in}, End	T5.	C_{out}, Gra, R_{in}, End	T5.	C_{out}, MA_{in}	T5.	Grb, R_{out}, C=B
				T6.	Gra, R_{out}, MD_{in}, Write	T6.	n≠0 → (C_{out}, SHR, C_{in}, Decr, Goto7)
				T7.	Wait, End	T7.	C_{out}, Gra, R_{in}, End

• • •

Example: Grc = T4·add + T4·(n=0)·shr + ...

Fig. 4.13 Generation of the logic for C_{out} and G_{ra}



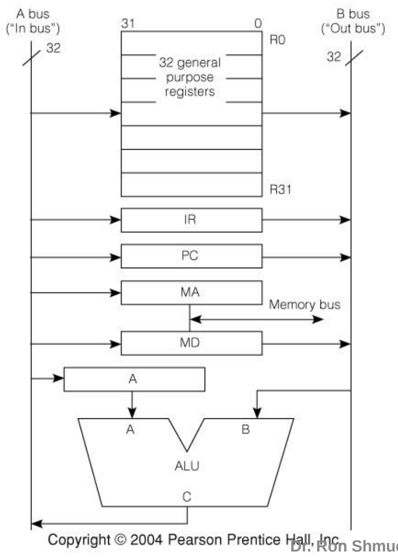
Have Completed One-Bus Design of SRC

- 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

Other Architectural designs will require a different RTN

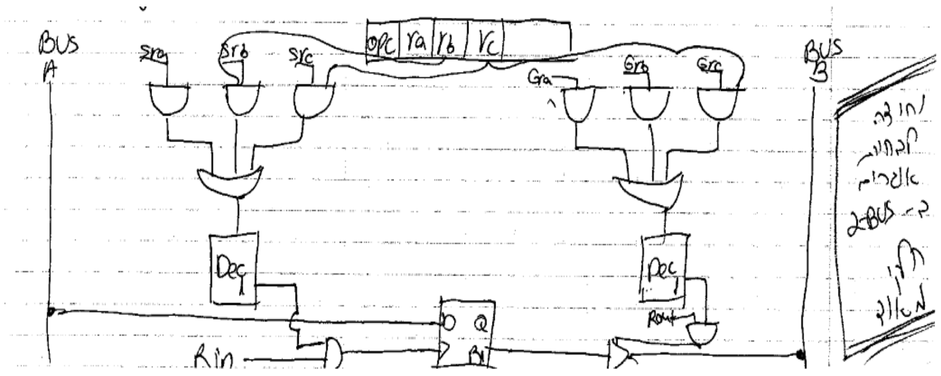
- More data paths allow more things to be done in one step
- Consider a two bus design
- By separating input and output of ALU on different buses, the C register is eliminated
- Steps can be saved by strobing ALU results directly into their destinations

Fig. 4.16 The 2-bus Microarchitecture



- Bus A carries data going into registers
- Bus B carries data being gated out of registers
- ALU function C=B is used for all simple register transfers

תכנון רעיוני של יחידת בחירת אוגרים בשני ערוצים



Tbl 4.13 Concrete RTN and Control Sequence for 2-bus SRC add

Step	Concrete RTN	Control Sequence
T0.	MA ← PC;	PC_{out}, C=B, MA_{in}, Read
T1.	PC ← PC + 4: MD ← M[MA];	PC_{out}, Inc4, PC_{in}, Wait
T2.	IR ← MD;	MD_{out}, C=B, IR_{in}
T3.	A ← R[rb];	Grb, R_{out}, C=B, A_{in}
T4.	R[ra] ← A + R[rc];	Grc, R_{out}, ADD, Sra, R_{in}, End

- Note the appearance of Grc to gate the output of the register rc onto the B bus and Sra to select ra to receive data strobed from the A bus
- Two register select decoders will be needed
- Transparent latches will be required for MA at step T0

Performance Measures

- MIPS: Millions of Instructions Per Second
  - Same job may take more instructions on one machine than on another
- MFLOPS: Million Floating Point OPs Per Second
  - Other instructions counted as overhead for the floating point
- Whetstones: Synthetic benchmark
  - A program made-up to test specific performance features
- Dhrystones: Synthetic competitor for Whetstone
  - Made up to “correct” Whetstone’s emphasis on floating point
- SPEC: Selection of “real” programs
  - Taken from the C/Unix world

Composite Performance Measure מדידת ביצועים משולבת

- המדד העיקרי במדידת ביצועים הוא מהירות

Response time , Execution time or Latency

$$\text{Execution time } = T = IC \times CPI \times \tau$$

IC = how many instructions have executed
CPI = the average number of clock cycles per instruction.
τ = **clock period, (700Mhz Pentium, 600MHz Alpha …)**

יש תלות בין הפרמטרים –פקודה עם CPI נמוך ככל הנראה מבצעת פחות ויתכן ויידרש IC גבוה יותר.

Performance and Design

$$\%Speedup = \frac{T_{old} - T_{new}}{T_{new}} \times 100$$

כאשר הכנסנו שיפור במעבד (מעבר מ 1Bus (old) ל 2Bus (new)

$$\%Speedup = \frac{T_{1-bus} - T_{2-bus}}{T_{2-bus}} \times 100$$

$$Where$$

$$T = Exec'n.Time = IC \times CPI \times \tau$$

Speedup Due To Going to 2 Buses

•Assume for now that IC and t don’t change in going from 1 bus to 2 buses
•Naively assume that CPI goes from 8 to 7 clocks.

$$\%Speedup = \frac{T_{1-bus} - T_{2-bus}}{T_{2-bus}} \times 100$$

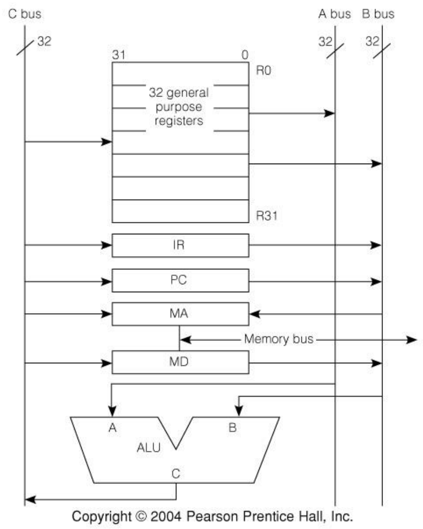
$$= \frac{IC \times 8 \times \tau - IC \times 7 \times \tau}{IC \times 7 \times \tau} \times 100 = \frac{8-7}{7} \times 100 = 14\%$$

Class Problem:
How will this speedup change if clock period of 2-bus machine is increased by 10%?

3-bus Architecture Shortens Sequences Even More

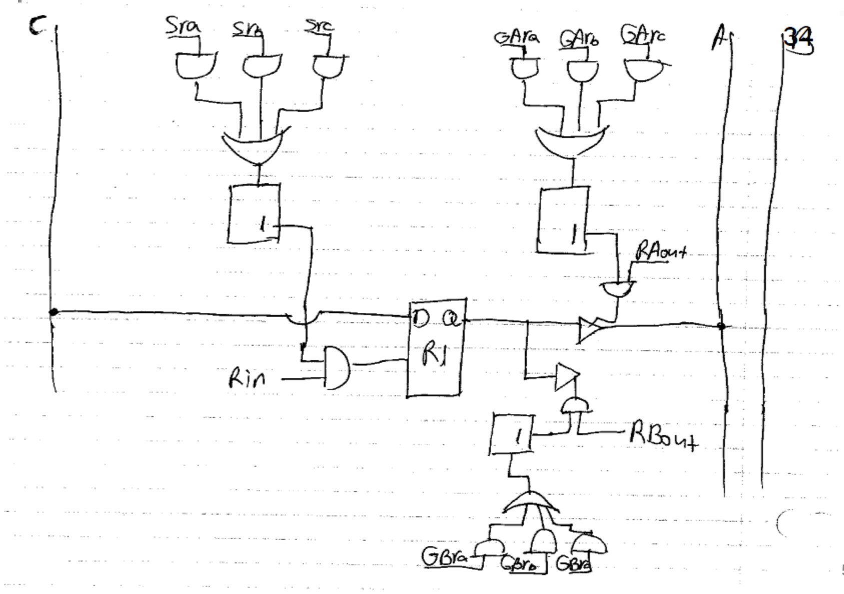
- A 3-bus architecture allows both operand inputs and the output of the ALU to be connected to buses
- Both the C output register and the A input register are eliminated
- Careful connection of register inputs and outputs can allow multiple RTs in a step

Fig. 4.17 The 3-Bus SRC Design



- A-bus is ALU operand 1, B-bus is ALU operand 2, and C-bus is ALU output
- Note MA input connected to the B-bus

תכנון רעיוני של יחידת בחירת אוגרים בשלושה ערוצים



Tbl 4.15 SRC add Instruction for the 3-bus Microarchitecture

Step	Concrete RTN	Control Sequence
T0.	MA ← PC: PC ← PC + 4: MD ← M[MA];	PC_{out}, MA_{in}, Inc4, PC_{in}, Read, Wait
T1.	IR ← MD;	MD_{out}, C=B, IR_{in}
T2.	R[ra] ← R[rb] + R[rc];	GArc, RA_{out}, GBrb, RB_{out}, ADD, Sra, R_{in}, End

- Note the use of 3 register selection signals in step T2: GArc, GBrb, and Sra
- In step T0, PC moves to MA over bus B and goes through the ALU Inc4 operation to reach PC again by way of bus C
  - PC must be edge triggered or master-slave
- Once more MA must be a transparent latch

Performance and Design

- How does going to three buses affect performance?
- Assume average CPI goes from 8 to 4, while τ increases by 10%:

$$\%Speedup = \frac{IC \times 8 \times \tau - IC \times 4 \times 1.1\tau}{IC \times 4 \times 1.1\tau} \times 100 = \frac{8-4.4}{4.4} \times 100 = 82\%$$

Other Performance Measures

- MIPS: Millions of Instructions Per Second
  - Same job may take more instructions on one machine than on another
- MFLOPS: Million Floating Point OPs Per Second
  - Other instructions counted as overhead for the floating point
- Whetstones: Synthetic benchmark
  - A program made-up to test specific performance features
- Dhrystones: Synthetic competitor for Whetstone
  - Made up to “correct” Whetstone’s emphasis on floating point
- SPEC: Selection of “real” programs
  - Taken from the C/Unix world

Native MIPS : Millions of Instructions Per Second

$$MIPS \equiv \frac{Instruction\ Count}{CPU\ Time\ x\ 10^6} = \frac{Instruction\ Count}{ICxCPIx[Clock\ cycle\ Time]x\ 10^6} = \frac{IC\ x\ [Clock\ Rate]}{ICxCPIx10^6}$$

$$\text{MIPS} = \frac{[Clock\ Rate]}{CPIx10^6}$$

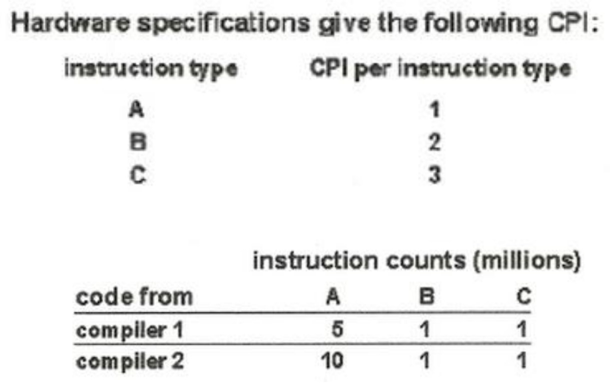
- Native MIPS
  - פופולרי
  - אינו לוקח בחשבון את ה IC
  - לא ניתן להשוות בעזרתו מכונות עם סט פקודות שונה.
  - מדד MIPS גבוהה יותר מעיד על ביצועים טובים יותר

Native MIPS vs. Composite Performance Measure

דוגמא

נתון ה- IC של תוכנית שעברה קומפילציה בשני מהדרים שונים ומורצת ע"י אותו CPU

- בתוכנית 3 סוגי פקודות A,B,C



Hardware specifications give the following CPI:

instruction type	CPI per instruction type
A	1
B	2
C	3

	instruction counts (millions)		
code from	A	B	C
compiler 1	5	1	1
compiler 2	10	1	1

המשך דוגמא

- חישוב מספר הפקודות IC?
- $IC1=(5+1+1)10^6 = 7x10^6$ $IC2=(10+1+1)10^6= 12 x10^6$
- חישוב ה CPI הממוצע?

- $\text{CPI 1} = \frac{((5x1)+(1x2)+(1x3)) \times 10^6}{(5+1+1) \times 10^6} = \frac{10}{7} = 1.428$
- $\text{CPI 2} = \frac{((10x1)+(1x2)+(1x3)) \times 10^6}{(10+1+1) \times 10^6} = \frac{15}{12} = 1.25$

- חישוב זמן הביצוע T?

$T1= 7M *1.428*\Delta=9.996M*\Delta$ $T2= 12M *1.25*\Delta=15*\Delta$

$SU=(T2-T1)/T1=(15-10)/15 =33\%$

- מכאן קומפיילר 1 טוב יותר ב 33%

שימוש ב MIPS לאומדן ביצועים נותן את ההיפך

- $\text{MIPS 1} = 100\text{ MHz} / (1.428 \times 10^6) = 70$
- $\text{MIPS 2} = 100\text{ MHz} / (1.25 \times 10^6) = 80$

f=100 MHz

כלומר קומפיילר 2 טוב יותר

Dr. Ron Shmueli

דוגמא

- מתכנת של קומפיילר בוחן שתי אפשרויות ליישום פקודה בשפה עילית,
- מהי האפשרות המועדפת?

Instruction count per Type			
Sequence	A	B	C
Option 1	2	1	2
Option 2	4	1	1

HW Specification give the following CPI			
Instruction Type	A	B	C
CPI per Instruction	1	2	3

IC1=(2+1+1) =5 IC2=(4+1+1) = 6

CPI1=(2*1+1*2+2*3)/5= 2 CPI2=(4*1+1*2+1*3)/6= 1.5

T1=IC1*CPI1*dt=5*2*dt= 10dt T2=IC2*CPI2*dt=6*1.5*dt= 9dt

SU=100*(T1-T2)/T2= 100*(10-9)/9= 11% אפשרות 2 מהירה יותר ב-

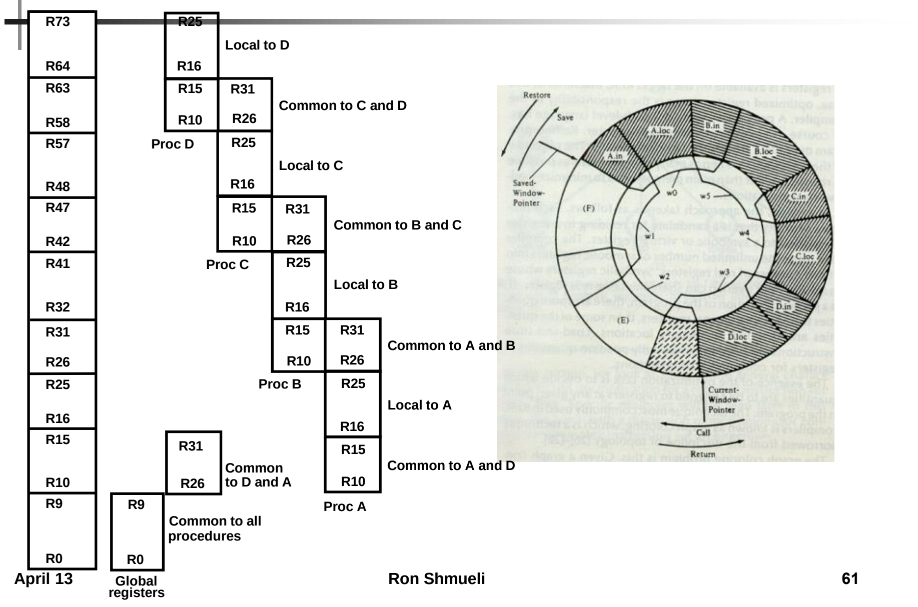
שימוש ב MIPS ייתן מדד דומה :

$$\text{MIPS} = \frac{[Clock\ Rate]}{CPIx10^6}$$

$$\boxed{\text{MIPS2} = \frac{[Clock\ Rate]}{1.5x10^6} > \text{MIPS1} = \frac{[Clock\ Rate]}{2x10^6}}$$

2013 Dr. Ron Shmueli 60

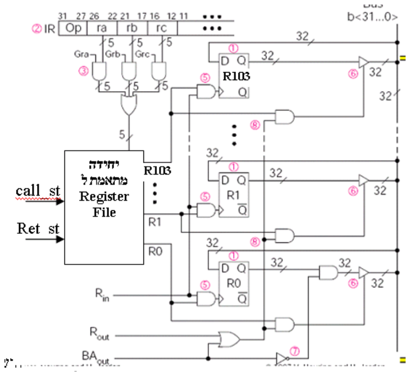
OVERLAPPED REGISTER WINDOWS

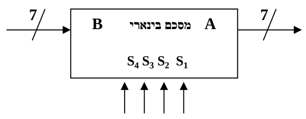


April 13 Ron Shmueli 61

- **<u>דוגמא ממבחן</u>**
- <u>ה- SRC בארכיטקטורת ערוץ יחיד</u> הותאם לתמיכה בשפות תכנות עיליות ע"י הגדלת מספר האוגרים ב- Register file ל- 104 אוגרים ושימוש בטכניקה - Overlapped Register Windows
- לתוכנית הראשית ממופים 32 האוגרים R0-R31. בקריאה לשגרה A, 32 האוגרים R24-R55 ממופים לשגרה A. כאשר האוגרים R24-R31 משותפים לתוכנית הראשית ולשגרה A. השגרה A ניגשת לאוגרים ע"פ R0-R31 ואלו ממופים בחומרה לכתובות R24-R55 בהתאמה. כאשר מתבצעת קריאה ע"י השגרה A לשגרה B, ממופים 32 האוגרים R48-R79 לשגרה B, שמונת האוגרים R48-R55 משותפים לשגרה A ולשגרה B. השגרה B פונה לאוגרים ע"פ R0-R31 ואלו ממופים בחומרה לכתובת האמיתית R48-R79. באותה צורה גם בעבור קריאה לשגרה C ע"י B ימופו האוגרים R72-R103 לשגרה C. <u>מבנה זה מאפשר קריאה מקוננת של שגרות בשלוש רמות</u>.
- לצורך התאמת המעבד ליכולת החדשה,הוסיפו את הפקודות הבאות:
- -הפקודה call, זהה ל brl, וגורמת גם לקו בקרה נוסף call_st לעלות ל 1 לוגי לצעד זמן אחד.
- -הפקודה ret , זהה ל br, וגורמת גם לקו בקרה נוסף ret_st לעלות ל 1 לוגי לצעד זמן אחד.

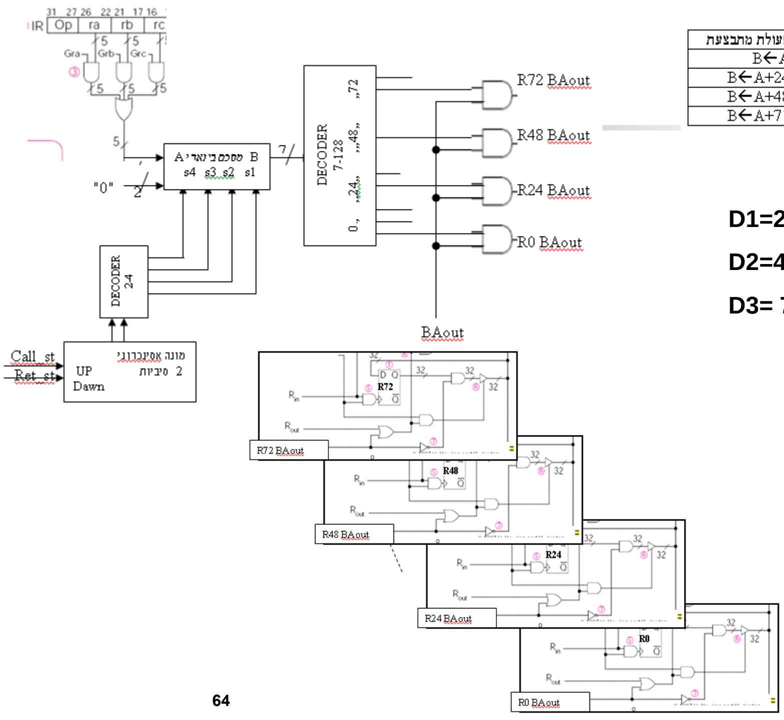
- **<u>מבלי לשנות את מיבנה הפקודות הקיימות ב- SRC,</u>** ובעזרת קווי הבקרה הנוספים. נדרש לתכנן **יחידה מתאמת** הממפה את מספרי האוגרים R0-R31 המצוינים בפקודות האסמבלי, לאוגרים הפיסיים בהתאם לרמת הקינון של השגרה המתבצעת (ראה איור).
- א. תן תכנון מפורט של היחידה המתאמת ל Register File המתוארת באיור (15 נק), ובכלל זה –תן את שינוי החומרה הנדרש לטיפול בסיגנל BAout בכל רמות הקינון (תכנן רמה אחת וציין במפורש באלו אוגרים מתבצע שינוי) (10 נק).
  - **במידת הצורך ניתן להשתמש ביחידה מסכמת המבצעת סיכום עם ערכים קבועים, אשר אופן פעולתה מתואר בטבלה. d1,d2,d3) קבועים לבחירתך).**
- **ב. כתוב תוכנית באסמבלי של ה- SRC המשודרג המאפסת את האוגרים R10,R34,R58,R82 (10 נק).**





$S_4S_3S_2S_1$	פעולת מתבצעת
0001	B←A
0010	B←A+d1
0100	B←A+d2
1000	B←A+d3

Ron Shmu 63



$S_4S_3S_2S_1$	פעולת מתבצעת
0001	B←A
0010	B←A+24
0100	B←A+48
1000	B←A+71

D1=24

D2=48

D3= 72

64

- **<u>שאלה 2 סעיף ב</u>**

```
main:
           la  r0, L1
          la  r10, 0
         call r24, r0
          :
L1:      la r0,L2
         la   r10,0
         call r24, r0

L2:    la r0,L3
       la   r10,0
       call r24, r0

L3:    la   r10,0
       ret  r0, r1
.end
```