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

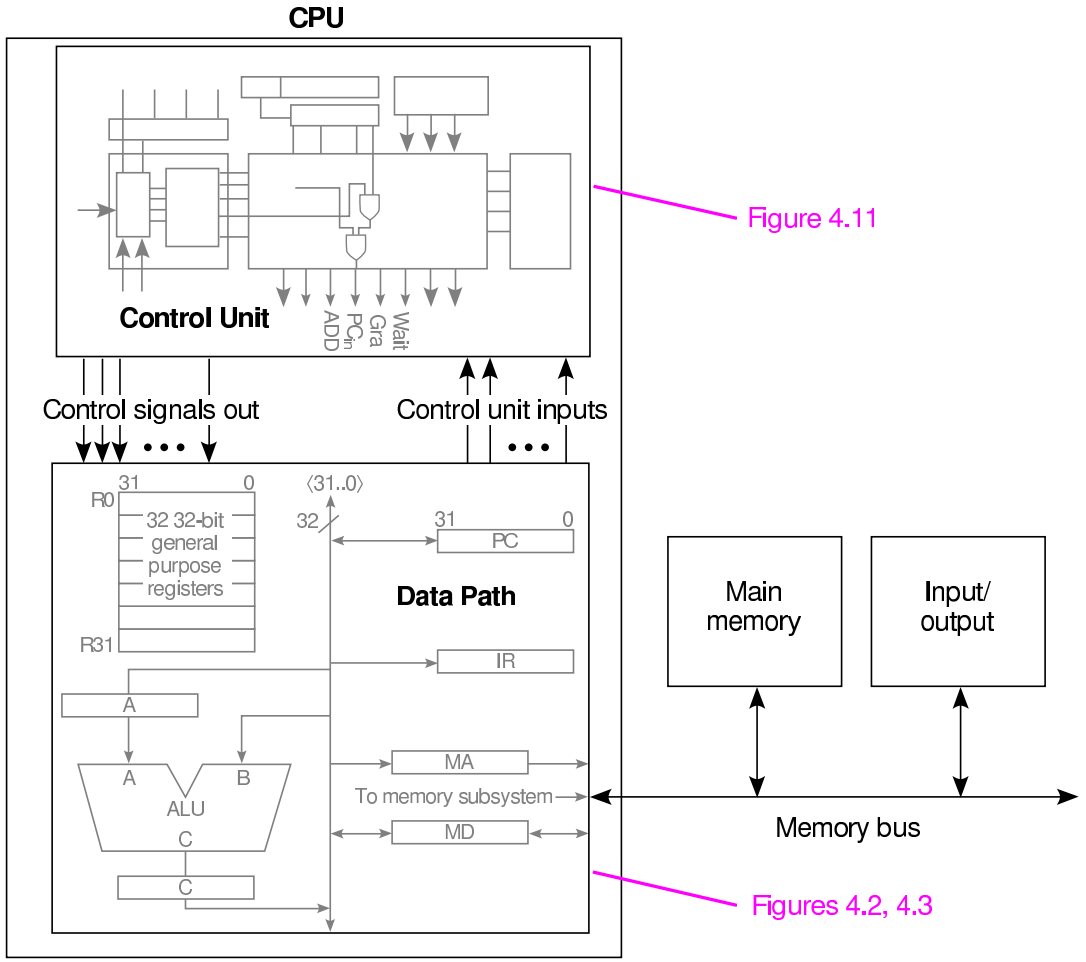
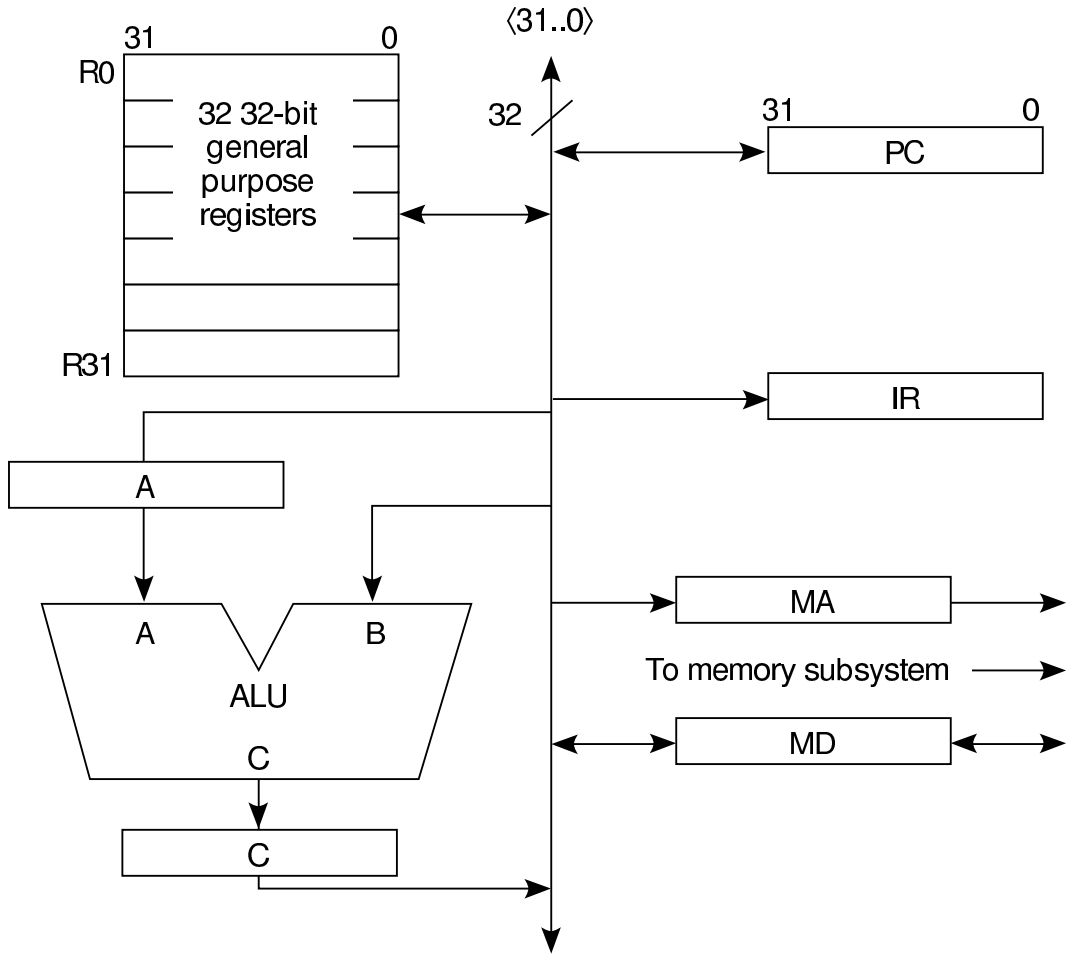
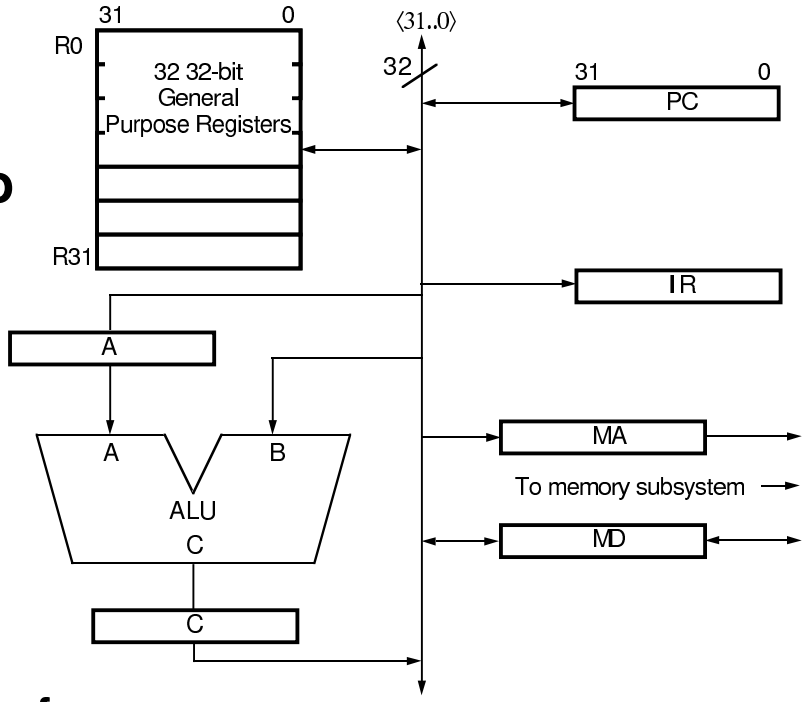


Fig 4.2 High-Level View of the 1-Bus SRC Design



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

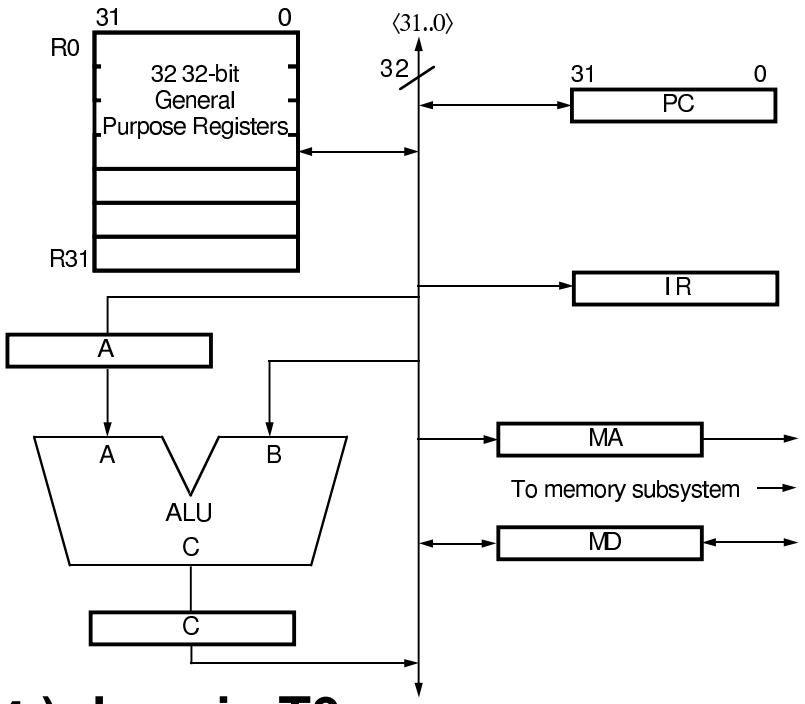


Abstract and Concrete RTN for SRC `add` Instruction

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

Step	RTN	
T0	MA ← PC: C ← PC + 4;	IF
T1	MD ← M[MA]: PC ← C;	
T2	IR ← MD;	
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)

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

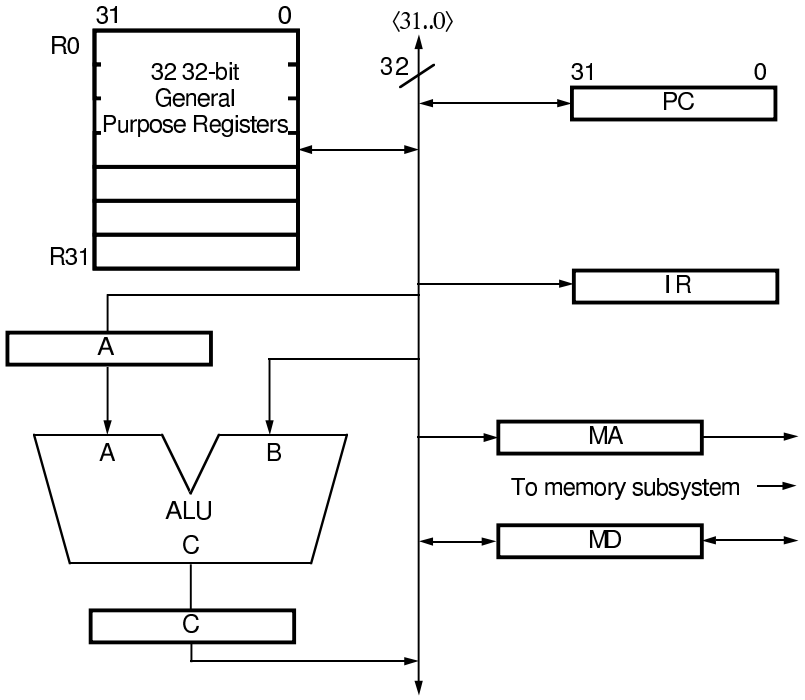
Concrete RTN for Arithmetic Instructions: addi

Abstract RTN:

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

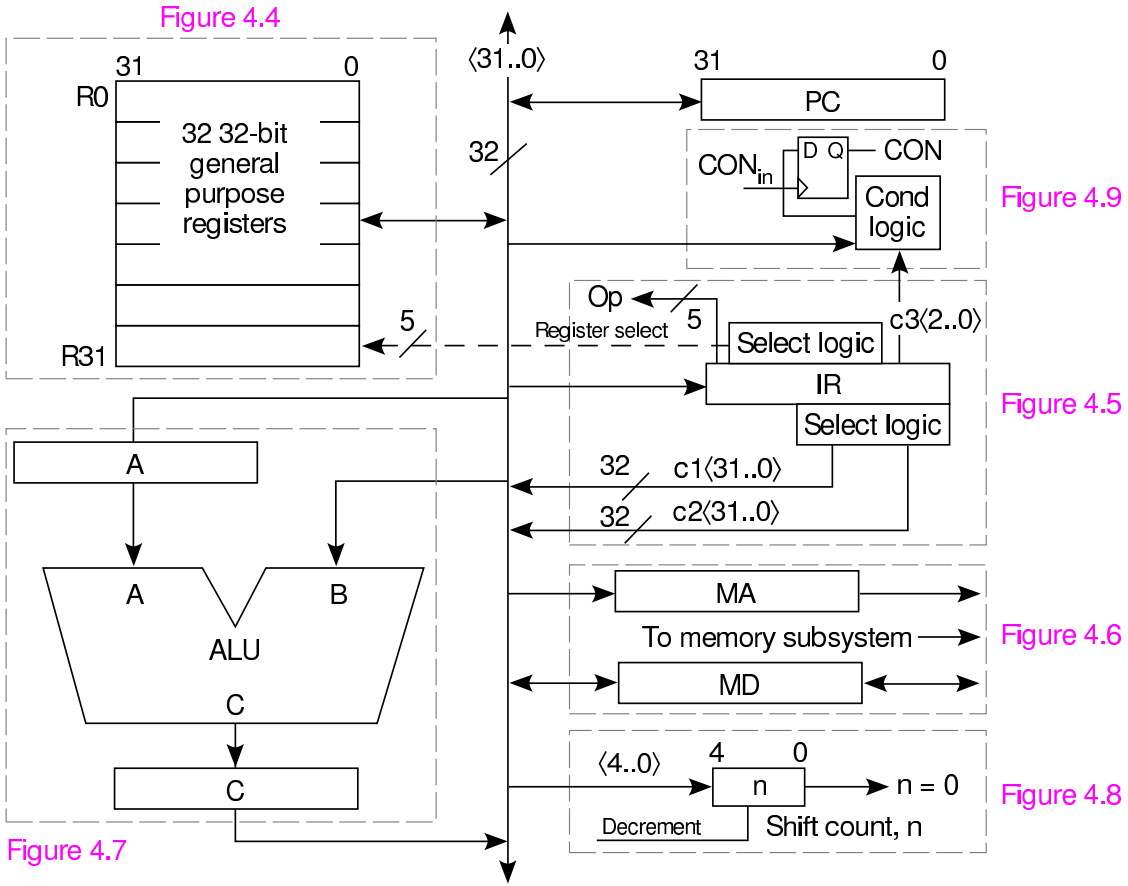
Concrete RTN for addi:

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

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

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 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⟨16..0⟩
- Memory read into MD occurs at T6 of ld
- Write of MD into memory occurs at T7 of st

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 The Branch Instruction, br

Step	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 register

Abstract and Concrete RTN for SRC Shift Right

shr (:= op = 26) → R[ra]⟨31..0⟩ ← (n @ 0) # R[rb]⟨31..n⟩ :
n := ((c3⟨4..0⟩ = 0) → R[rc]⟨4..0⟩ : Shift count in register
(c3⟨4..0⟩ ≠ 0) → c3⟨4..0⟩): or constant field of instruction

Tbl 4.5 The shr Instruction

Step	Concrete RTN
T0–T2	Instruction fetch
T3	n ← IR⟨4..0⟩;
T4	(n = 0) → (n ← R[rc]⟨4..0⟩);
T5	C ← R[rb];
T6	Shr (:= (n ≠ 0) → (C⟨31..0⟩ ← 0#C⟨31..1⟩: n ← n - 1; Shr));
T7	R[ra] ← C;

step T6 is repeated n times