Computer Architecture

Reduced Instruction Set Computer

Dr. Masood Mir

Week 12

Reduced Instruction Set Computer (RISC)

- The first design we examine is for a reduced instruction set computer with a **pipelined datapath and pipelined control unit**.
- We begin by describing the RISC instruction set architecture, which is characterized by **load/store memory access**, **four addressing modes**, **a single instruction format length**, and **reduced instructions** that require only elementary operations.
- The operations, resembling those that can be performed by the single-cycle computer, can be performed by a single pass through the pipeline.

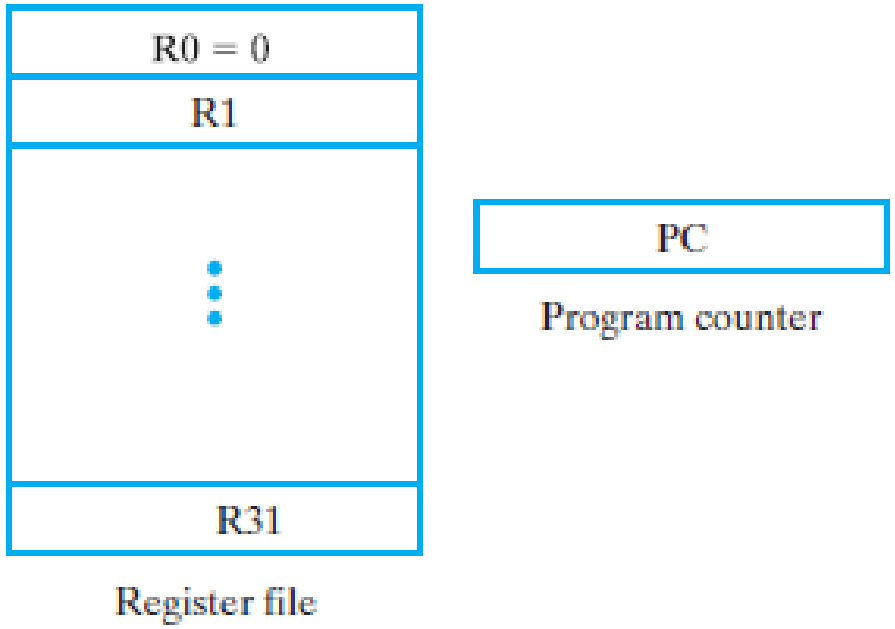


FIGURE 10-6
CPU Register Set Diagram for RISC

- The datapath for implementing the ISA is **based on the single-cycle datapath** initially described in Figure 8-11 and converted to a **pipeline** in Figure 10-2. In order to implement the RISC instruction set architecture, modifications are made to the register file and the function unit.

Instruction Set Architecture of RISC

- Figure 10-6 shows the CPU registers accessible to the programmer in this RISC.
- All registers are **32 bits**. The register file has **32 registers**, R0 through R31.
- The size of the programmer-accessible register file is comparatively large in the RISC because of the **load/store instruction set architecture**.
- Since the data-manipulation operations can use only register operands, many active operands need to be present in the register file. Otherwise, numerous stores and loads would be needed to temporarily save operands in the data memory between data-manipulation operations.
- **R0** is a special register that supplies the value zero when used as a source and discards the result when used as a destination.

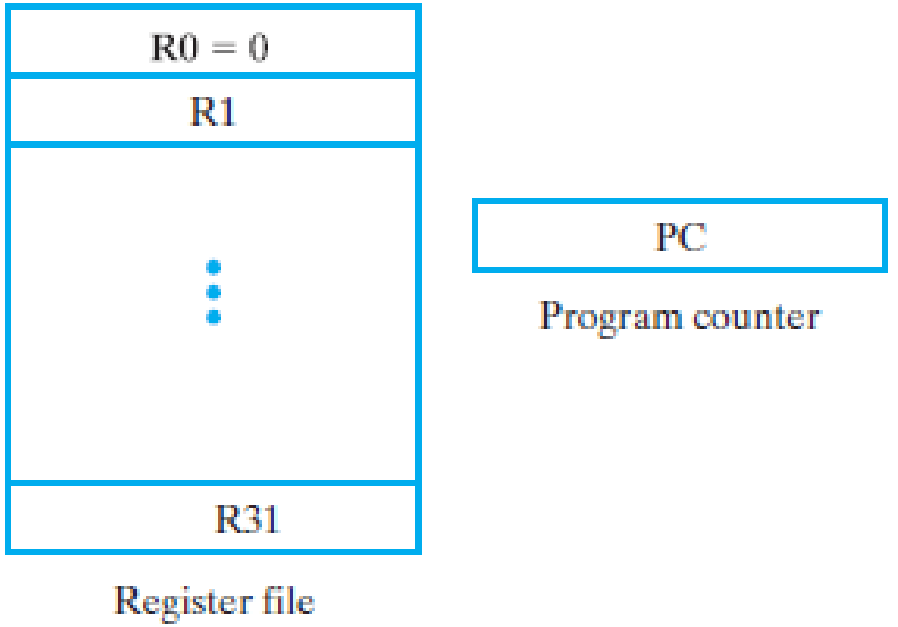


FIGURE 10-6
CPU Register Set Diagram for RISC

Instruction Set Architecture of RISC

- In addition to the register file, only a program counter, **PC**, is provided.
- If **stack pointer**-based or processor **status register**-based operations are required, they are simply implemented by sequences of instructions using registers.
- Figure 10-7 gives the **three instruction formats** for the RISC CPU. The formats use a single word of 32 bits. This **longer word length** is needed to hold realistic address values, since additional instruction words for holding addresses are difficult to accommodate in the RISC CPU.
- The first format specifies three registers. The two registers addressed by the 5-bit source register fields SA and SB contain the two operands. The third register, addressed by a 5-bit destination register field DR, specifies the register location for the result.

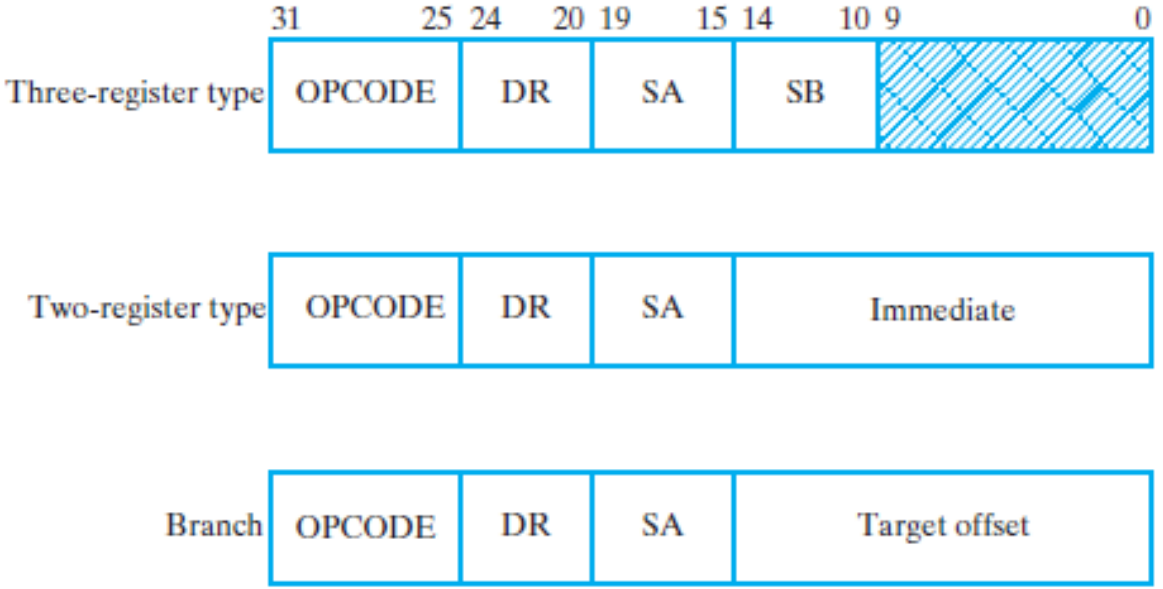


FIGURE 10-7
RISC CPU Instruction Formats

- **Three types of instruction formats** are used in RISC we are discussing.

Instruction Set Architecture of RISC

- A **7-bit OPCODE** provides for a maximum of 128 operations.
- The remaining two formats replace the second register with a **15-bit constant**.
- In the two-register format, the constant acts as an **immediate operand**, and in the branch format, the constant is a **target offset**.
- The **branch instructions** specify source register **SA**.
- Whether the branch or jump is taken is based on whether the source register contains zero. The DR field is used to specify the register in which to store the return address for the procedure call.
- Finally, the rightmost **5 bits of the 15-bit constant** are also used as the shift amount SH for multiple bit shifts.

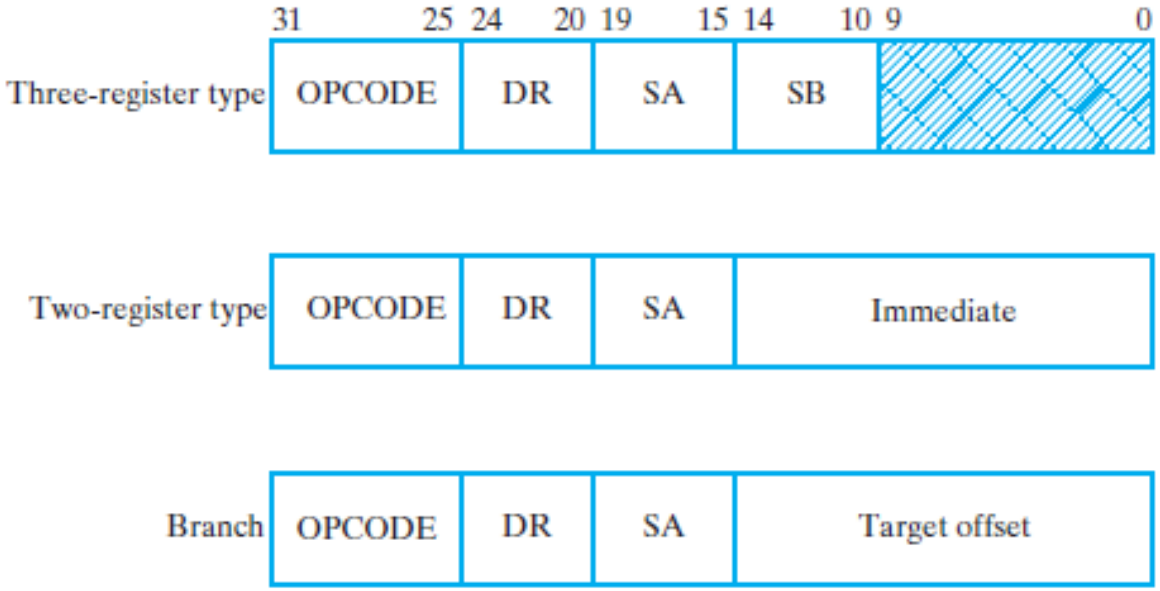


FIGURE 10-7
RISC CPU Instruction Formats

ISA of RISC

- Table 10-1 contains the **27 operations** to be performed by the instructions.
- A mnemonic, an opcode, and a register transfer description are given for each operation. All of the operations are elementary and can be described by a single register transfer statement. The only operations that can access memory are **Load and Store**.
- A significant number of **immediate instructions** help to reduce data memory accesses and speed up execution when constants are employed.
- Since the immediate field of the instruction is only **15 bits**, the leftmost 17 bits must be filled to form a 32-bit operand. In addition to using **zero fill** for logical operations, a second method used is called **sign extension**.

TABLE 10-1
RISC Instruction Operations

Operation	Symbolic Notation	Opcode	Action
No Operation	NOP	0000000	None
Move A	MOVA	1000000[1]	$R[DR] \leftarrow R[SA]$
Add	ADD	0000010	$R[DR] \leftarrow R[SA] + R[SB]$
Subtract	SUB	0000101	$R[DR] \leftarrow R[SA] + \overline{R[SB]} + 1$
AND	AND	0001000	$R[DR] \leftarrow R[SA] \wedge R[SB]$
OR	OR	0001001	$R[DR] \leftarrow R[SA] \vee R[SB]$
Exclusive-OR	XOR	0001010	$R[DR] \leftarrow R[SA] \oplus R[SB]$
Complement	NOT	0001011	$R[DR] \leftarrow \overline{R[SA]}$
Add Immediate	ADI	0100010	$R[DR] \leftarrow R[SA] + \text{se } IM$
Subtract Immediate	SBI	0100101	$R[DR] \leftarrow R[SA] + \overline{(\text{se } IM)} + 1$
AND Immediate	ANI	0101000	$R[DR] \leftarrow R[SA] \wedge (0 \parallel IM)$
OR Immediate	ORI	0101001	$R[DR] \leftarrow R[SA] \vee (0 \parallel IM)$
Exclusive-OR Immediate	XRI	0101010	$R[DR] \leftarrow R[SA] \oplus (0 \parallel IM)$
Add Immediate Unsigned	AIU	1000010	$R[DR] \leftarrow R[SA] + (0 \parallel IM)$
Subtract Immediate Unsigned	SIU	1000101	$R[DR] \leftarrow R[SA] + \overline{(0 \parallel IM)} + 1$
Move B	MOVB	0001100	$R[DR] \leftarrow R[SB]$
Logical Right Shift by SH Bits	LSR	0001101	$R[DR] \leftarrow \text{lsr } R[SA] \text{ by } SH$
Logical Left Shift by SH Bits	LSL	0001110	$R[DR] \leftarrow \text{lsl } R[SA] \text{ by } SH$
Load	LD	0010000	$R[DR] \leftarrow M[R[SA]]$
Store	ST	0100000	$M[R[SA]] \leftarrow R[SB]$
Jump Register	JMR	1110000	$PC \leftarrow R[SA]$
Set if Less Than[2]	SLT	1100101	If $R[SA] < R[SB]$ then $R[DR] = 1$
Branch if Zero	BZ	1100000	If $R[SA] = 0$, then $PC \leftarrow PC + 1 + \text{se } IM$
Branch if Nonzero	BNZ	1001000	If $R[SA] \neq 0$, then $PC \leftarrow PC + 1 + \text{se } IM$
Jump	JMP	1101000	$PC \leftarrow PC + 1 + \text{se } IM$
Jump and Link	JML	0110000	$PC \leftarrow PC + 1 + \text{se } IM, R[DR] \leftarrow PC + 1$

ISA of RISC

- The absence of stored versions of status bits is handled by the use of three instructions: **Branch if Zero (BZ), Branch if Nonzero (BNZ), and Set if Less Than (SLT)**.
- **BZ and BNZ** are single instructions that determine whether a register operand is zero or nonzero and branch accordingly.
- **SLT** stores a value in register R[DR] that acts like a negative status bit. If R[SA] is less than R[SB], a 1 is placed in register R[DR]. If R[SA] is greater than or equal to R[SB], a 0 is placed in R[DR]. The register R[DR] can then be examined by a subsequent instruction to see whether it is zero (0) or nonzero (1). Thus, using two instructions, the relative values of two operands or the sign of one operand (by letting R[SB] equal R0) can be determined.

TABLE 10-1
RISC Instruction Operations

Operation	Symbolic Notation	Opcode	Action
No Operation	NOP	0000000	None
Move A	MOVA	1000000[1]	$R[DR] \leftarrow R[SA]$
Add	ADD	0000010	$R[DR] \leftarrow R[SA] + R[SB]$
Subtract	SUB	0000101	$R[DR] \leftarrow R[SA] + \overline{R[SB]} + 1$
AND	AND	0001000	$R[DR] \leftarrow R[SA] \wedge R[SB]$
OR	OR	0001001	$R[DR] \leftarrow R[SA] \vee R[SB]$
Exclusive-OR	XOR	0001010	$R[DR] \leftarrow R[SA] \oplus R[SB]$
Complement	NOT	0001011	$R[DR] \leftarrow \overline{R[SA]}$
Add Immediate	ADI	0100010	$R[DR] \leftarrow R[SA] + \text{se } IM$
Subtract Immediate	SBI	0100101	$R[DR] \leftarrow R[SA] + \overline{(\text{se } IM)} + 1$
AND Immediate	ANI	0101000	$R[DR] \leftarrow R[SA] \wedge (0 \parallel IM)$
OR Immediate	ORI	0101001	$R[DR] \leftarrow R[SA] \vee (0 \parallel IM)$
Exclusive-OR Immediate	XRI	0101010	$R[DR] \leftarrow R[SA] \oplus (0 \parallel IM)$
Add Immediate Unsigned	AIU	1000010	$R[DR] \leftarrow R[SA] + (0 \parallel IM)$
Subtract Immediate Unsigned	SIU	1000101	$R[DR] \leftarrow R[SA] + \overline{(0 \parallel IM)} + 1$
Move B	MOVB	0001100	$R[DR] \leftarrow R[SB]$
Logical Right Shift by SH Bits	LSR	0001101	$R[DR] \leftarrow \text{lsr } R[SA] \text{ by } SH$
Logical Left Shift by SH Bits	LSL	0001110	$R[DR] \leftarrow \text{lsl } R[SA] \text{ by } SH$
Load	LD	0010000	$R[DR] \leftarrow M[R[SA]]$
Store	ST	0100000	$M[R[SA]] \leftarrow R[SB]$
Jump Register	JMR	1110000	$PC \leftarrow R[SA]$
Set if Less Than[2]	SLT	1100101	If $R[SA] < R[SB]$ then $R[DR] = 1$
Branch if Zero	BZ	1100000	If $R[SA] = 0$, then $PC \leftarrow PC + 1 + \text{se } IM$
Branch if Nonzero	BNZ	1001000	If $R[SA] \neq 0$, then $PC \leftarrow PC + 1 + \text{se } IM$
Jump	JMP	1101000	$PC \leftarrow PC + 1 + \text{se } IM$
Jump and Link	JML	0110000	$PC \leftarrow PC + 1 + \text{se } IM, R[DR] \leftarrow PC + 1$

ISA of RISC

- The **Jump and Link (JML)** instruction provides a mechanism for implementing procedures.
- PC <- PC + 1 + se IM, **$R[DR]$ <- PC + 1**
- The value in the PC after updating is stored in register R[DR], and then the sum of the PC and the sign-extended target offset from the instruction is placed in the PC.
- The return from a called procedure can use the Jump Register instruction with SA equal to DR for the calling procedure.
- If a procedure is to be called from within a called procedure, then each successive procedure that is called will need its own register for storing the return value. A software stack that moves return addresses from R[DR] to memory at the beginning of a called procedure and restores them to R[SA] before the return can also be used.

TABLE 10-1
RISC Instruction Operations

Operation	Symbolic Notation	Opcode	Action
No Operation	NOP	0000000	None
Move A	MOVA	1000000[1]	$R[DR] \leftarrow R[SA]$
Add	ADD	0000010	$R[DR] \leftarrow R[SA] + R[SB]$
Subtract	SUB	0000101	$R[DR] \leftarrow R[SA] + \overline{R[SB]} + 1$
AND	AND	0001000	$R[DR] \leftarrow R[SA] \wedge R[SB]$
OR	OR	0001001	$R[DR] \leftarrow R[SA] \vee R[SB]$
Exclusive-OR	XOR	0001010	$R[DR] \leftarrow R[SA] \oplus R[SB]$
Complement	NOT	0001011	$R[DR] \leftarrow \overline{R[SA]}$
Add Immediate	ADI	0100010	$R[DR] \leftarrow R[SA] + \text{se } IM$
Subtract Immediate	SBI	0100101	$R[DR] \leftarrow R[SA] + \overline{(\text{se } IM)} + 1$
AND Immediate	ANI	0101000	$R[DR] \leftarrow R[SA] \wedge (0 \parallel IM)$
OR Immediate	ORI	0101001	$R[DR] \leftarrow R[SA] \vee (0 \parallel IM)$
Exclusive-OR Immediate	XRI	0101010	$R[DR] \leftarrow R[SA] \oplus (0 \parallel IM)$
Add Immediate Unsigned	AIU	1000010	$R[DR] \leftarrow R[SA] + (0 \parallel IM)$
Subtract Immediate Unsigned	SIU	1000101	$R[DR] \leftarrow R[SA] + \overline{(0 \parallel IM)} + 1$
Move B	MOVB	0001100	$R[DR] \leftarrow R[SB]$
Logical Right Shift by SH Bits	LSR	0001101	$R[DR] \leftarrow \text{lsr } R[SA] \text{ by } SH$
Logical Left Shift by SH Bits	LSL	0001110	$R[DR] \leftarrow \text{lsl } R[SA] \text{ by } SH$
Load	LD	0010000	$R[DR] \leftarrow M[R[SA]]$
Store	ST	0100000	$M[R[SA]] \leftarrow R[SB]$
Jump Register	JMR	1110000	$PC \leftarrow R[SA]$
Set if Less Than[2]	SLT	1100101	If $R[SA] < R[SB]$ then $R[DR] = 1$
Branch if Zero	BZ	1100000	If $R[SA] = 0$, then $PC \leftarrow PC + 1 + \text{se } IM$
Branch if Nonzero	BNZ	1001000	If $R[SA] \neq 0$, then $PC \leftarrow PC + 1 + \text{se } IM$
Jump	JMP	1101000	$PC \leftarrow PC + 1 + \text{se } IM$
Jump and Link	JML	0110000	$PC \leftarrow PC + 1 + \text{se } IM, R[DR] \leftarrow PC + 1$

Addressing Modes in RISC

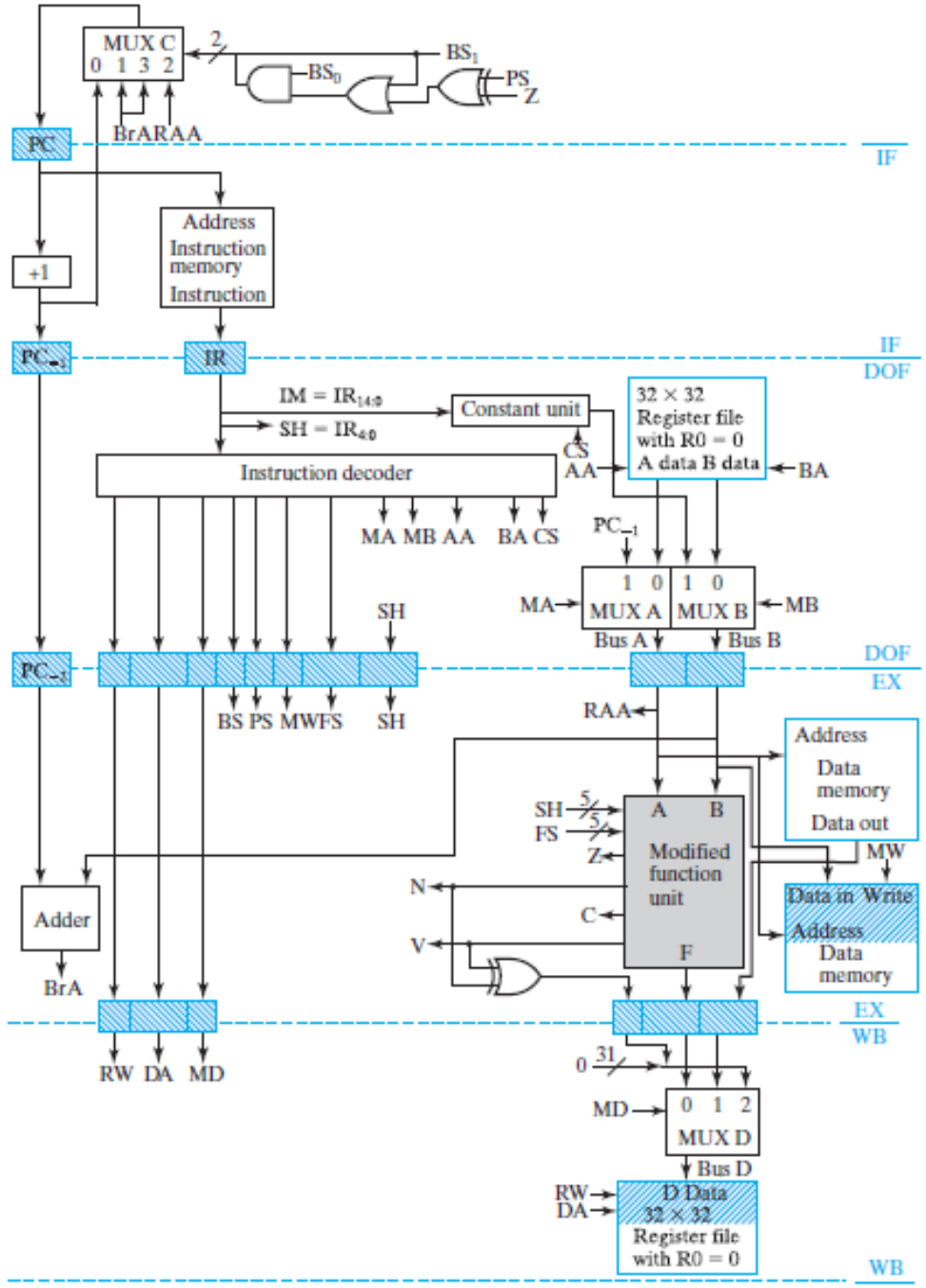
- The four addressing modes in the RISC are register, register indirect, immediate, and relative.
- The mode is specified by the operation code, rather than by separate mode field (fixed addressing mode).
- The three-operand data-manipulation instructions use **register-mode addressing**.
- **Register indirect**, applies only to the load and store instructions. In the indirect addressing mode, the address field of the instruction gives the address at which the effective address is stored in memory.
- Instructions using the two-register format have an **immediate** value that replaces register address SB.
- **Relative addressing** applies exclusively to branch and jump instructions and so produces addresses only for the instruction memory (offset to PC is in register).

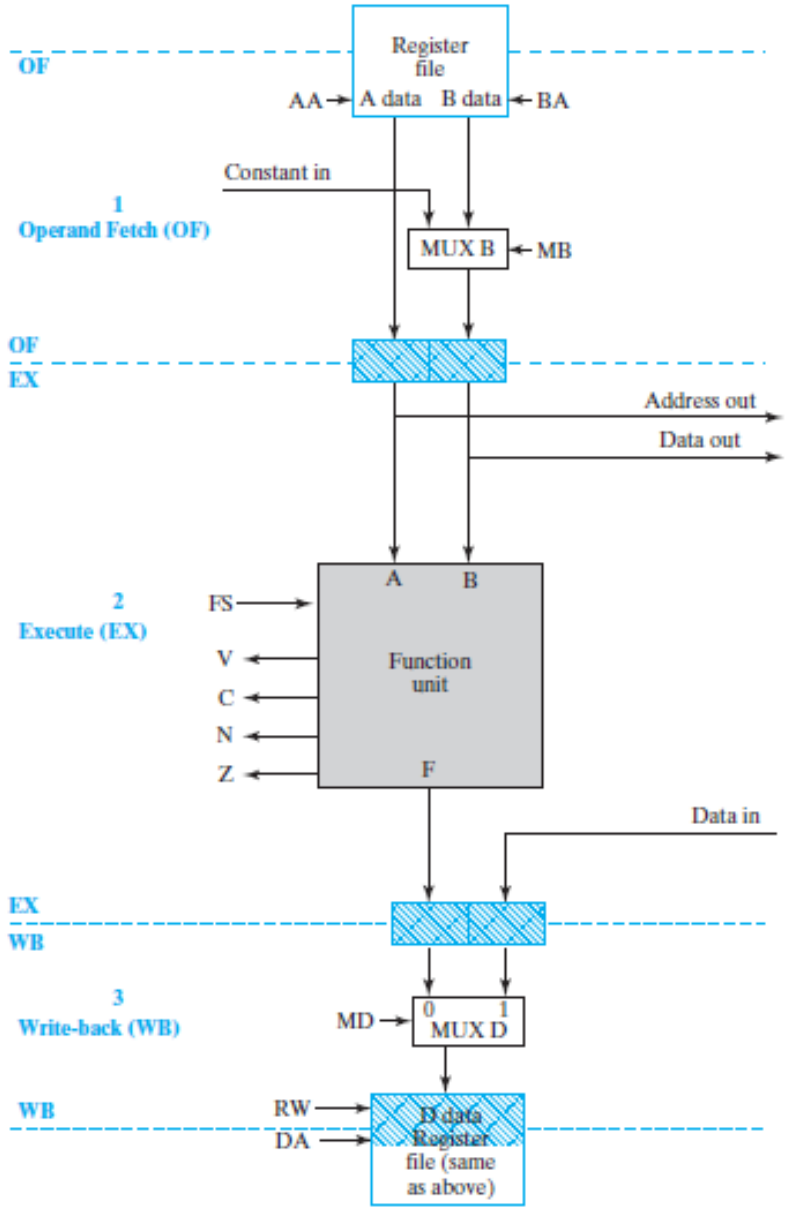
TABLE 10-1
RISC Instruction Operations

Operation	Symbolic Notation	Opcode	Action
No Operation	NOP	0000000	None
Move A	MOVA	1000000[1]	$R[DR] \leftarrow R[SA]$
Add	ADD	0000010	$R[DR] \leftarrow R[SA] + R[SB]$
Subtract	SUB	0000101	$R[DR] \leftarrow R[SA] + \overline{R[SB]} + 1$
AND	AND	0001000	$R[DR] \leftarrow R[SA] \wedge R[SB]$
OR	OR	0001001	$R[DR] \leftarrow R[SA] \vee R[SB]$
Exclusive-OR	XOR	0001010	$R[DR] \leftarrow R[SA] \oplus R[SB]$
Complement	NOT	0001011	$R[DR] \leftarrow \overline{R[SA]}$
Add Immediate	ADI	0100010	$R[DR] \leftarrow R[SA] + \text{se } IM$
Subtract Immediate	SBI	0100101	$R[DR] \leftarrow R[SA] + \overline{(\text{se } IM)} + 1$
AND Immediate	ANI	0101000	$R[DR] \leftarrow R[SA] \wedge (0 \parallel IM)$
OR Immediate	ORI	0101001	$R[DR] \leftarrow R[SA] \vee (0 \parallel IM)$
Exclusive-OR Immediate	XRI	0101010	$R[DR] \leftarrow R[SA] \oplus (0 \parallel IM)$
Add Immediate Unsigned	AIU	1000010	$R[DR] \leftarrow R[SA] + (0 \parallel IM)$
Subtract Immediate Unsigned	SIU	1000101	$R[DR] \leftarrow R[SA] + \overline{(0 \parallel IM)} + 1$
Move B	MOVB	0001100	$R[DR] \leftarrow R[SB]$
Logical Right Shift by SH Bits	LSR	0001101	$R[DR] \leftarrow \text{lsr } R[SA] \text{ by } SH$
Logical Left Shift by SH Bits	LSL	0001110	$R[DR] \leftarrow \text{lsl } R[SA] \text{ by } SH$
Load	LD	0010000	$R[DR] \leftarrow M[R[SA]]$
Store	ST	0100000	$M[R[SA]] \leftarrow R[SB]$
Jump Register	JMR	1110000	$PC \leftarrow R[SA]$
Set if Less Than[2]	SLT	1100101	If $R[SA] < R[SB]$ then $R[DR] = 1$
Branch if Zero	BZ	1100000	If $R[SA] = 0$, then $PC \leftarrow PC + 1 + \text{se } IM$
Branch if Nonzero	BNZ	1001000	If $R[SA] \neq 0$, then $PC \leftarrow PC + 1 + \text{se } IM$
Jump	JMP	1101000	$PC \leftarrow PC + 1 + \text{se } IM$
Jump and Link	JML	0110000	$PC \leftarrow PC + 1 + \text{se } IM, R[DR] \leftarrow PC + 1$

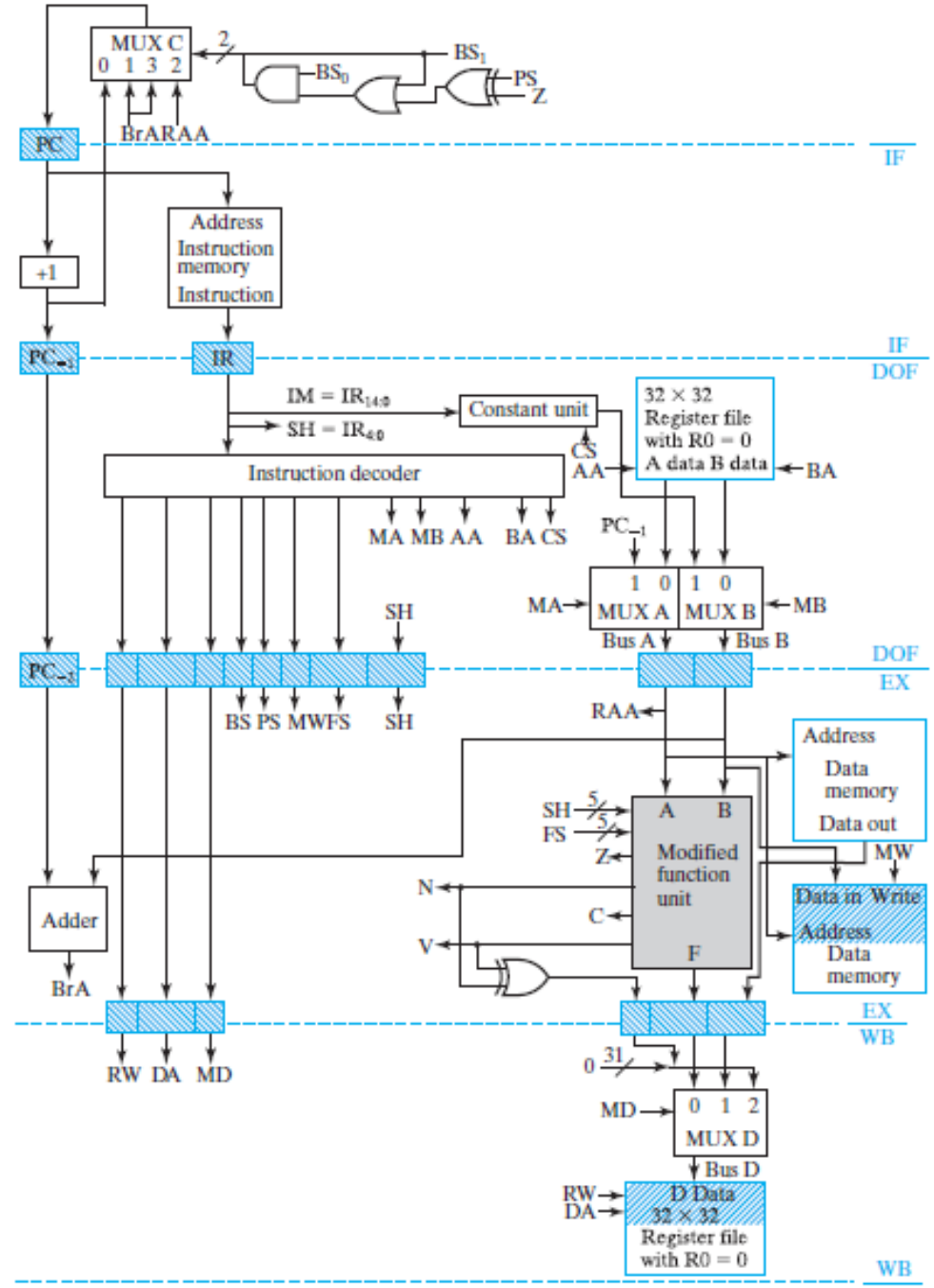
Datapath Modification for RISC

- The modifications for datapath unit affect the register file, the function unit, and the bus structure in figure.
- Figure 10-8 (for RISC) can be compared with fig 10-2.
- In Figure 10-2, there are 16 16-bit registers, and all registers are identical in function. In the new datapath, there are 32 32-bit registers. Also, reading register R0 gives a constant value of zero. If a write is attempted into R0, the data will be lost.
- All data inputs and the data output are 32 bits. To correspond to the 32 registers, the address inputs are five bits.
- The fixed value of 0 in R0 is implemented by replacing the storage elements for R0 with open circuits on the lines that were their inputs.
- These changes are implemented in the register file.



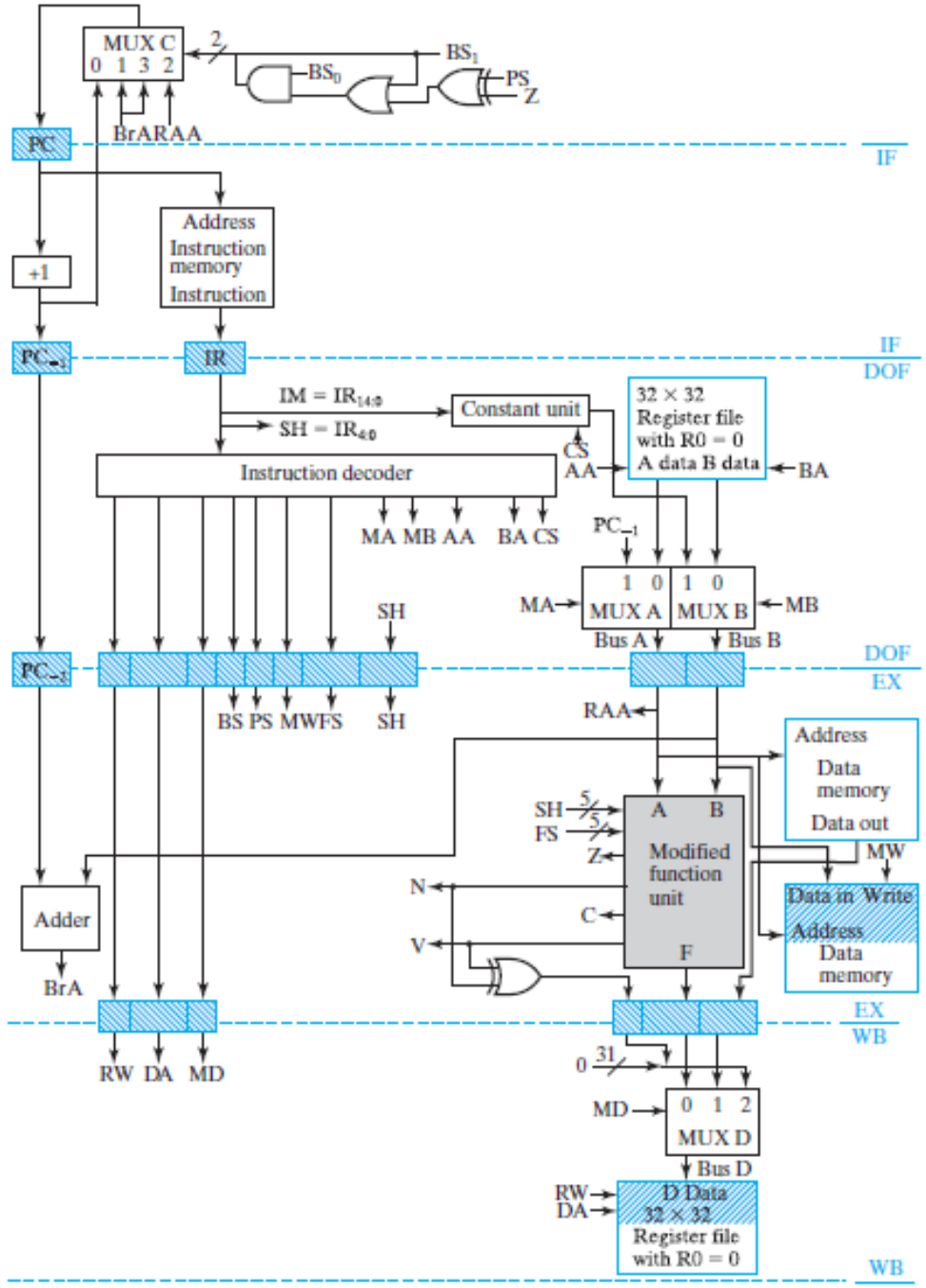


□ **FIGURE 10-2**
Block Diagram of Pipelined Datapath



Datapath Modification for RISC

- A second major modification to the datapath is the replacement of the single-bit position shifter with a barrel shifter to permit multiple-position shifting.
- This barrel shifter can perform a logical right or logical left shift of from 0 to 31 positions.
- The data input is 32-bit operand A, and the output is 32-bit result G. Left/right', a control signal decoded from OPCODE, selects a left or right shift. The shift amount field SH = IR (4:0) specifies the number of bit positions to shift the data input and takes on values from 0 through 31.
- A logical shift of p bit positions involves inserting p zeros into the result. In order to provide these zeros and simplify the design of the shifter, we will perform both the left and right shift by using a right rotate.



Barrel Shifter in RISC

- The data input is 32-bit operand A, and the output is 32-bit result G. Left/right', a control signal decoded from OPCODE, selects a left or right shift.
- The shift amount field SH = IR (4:0) specifies the number of bit positions to shift the data input and takes on values from 0 through 31.
- A logical shift of "p" bit positions involves inserting "p" zeros into the result.
- In order to provide these zeros and simplify the design of the shifter, we will perform both the left and right shift by using a right rotate.
- The input to this rotate will be the input data A with 32 zeros concatenated to its left.
- A right shift is performed by rotating the input p positions to the right; a left shift is performed by rotating 64 - p positions to the right.
- This number of positions can be obtained by taking the 2s complement of the 6-bit value of 0 || SH.

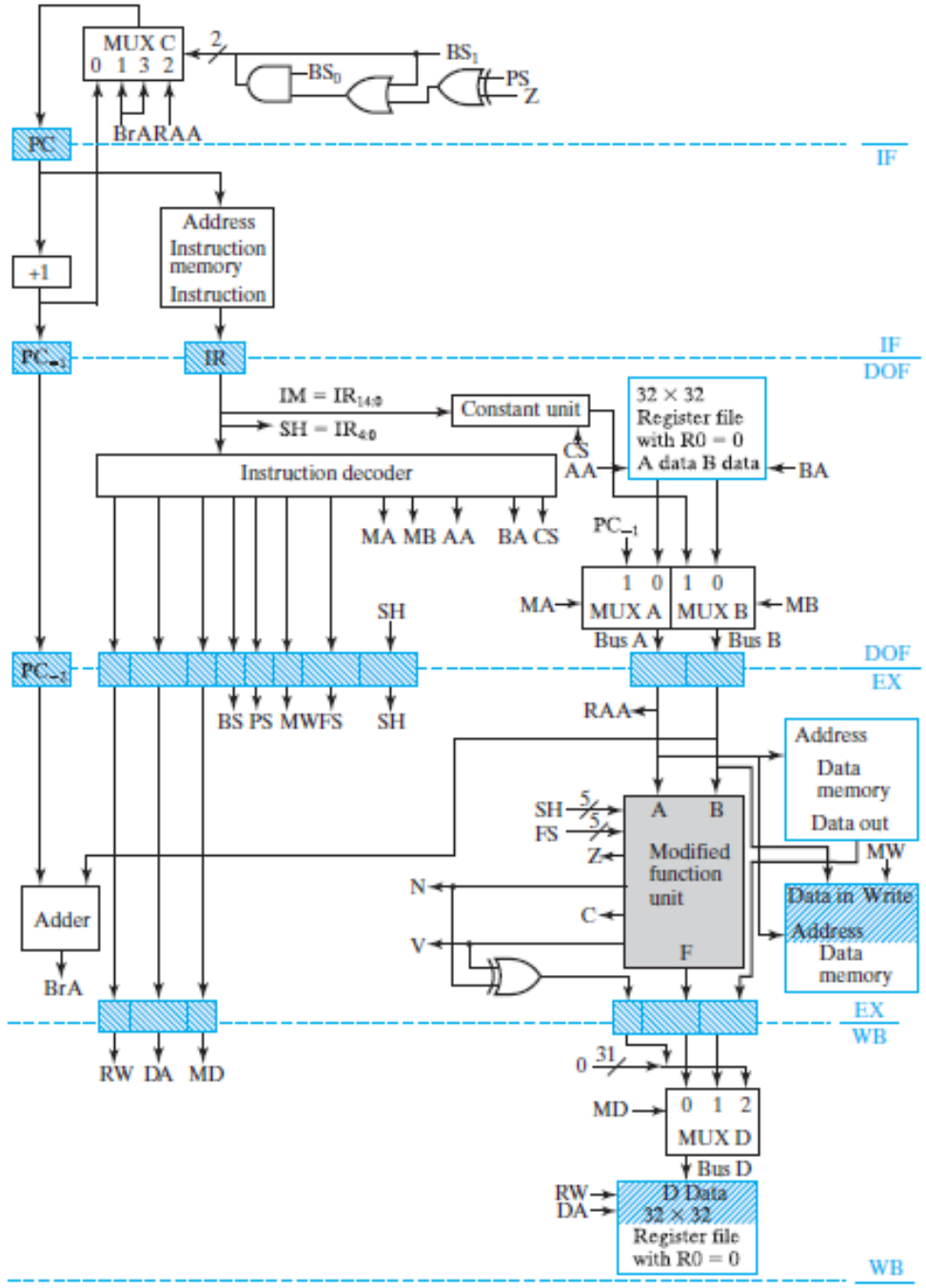
Answer of Rotate or Shift -> Shift achieved by Rotation?

Datapath Modification for RISC

- Role of BS0, BS1, and PS.

TABLE 10-2
Definition of Control Fields BS and PS

Register Transfer	BS Code	PS Code	Comments
$PC \leftarrow PC + 1$	00	X	Increment PC
Z: $PC \leftarrow$ BrA, $\overline{Z}$: $PC \leftarrow PC + 1$	01	0	Branch on Zero
$\overline{Z}$: $PC \leftarrow$ BrA, Z: $PC \leftarrow PC + 1$	01	1	Branch on Nonzero
$PC \leftarrow R[AA]$	10	X	Jump to Contents of $R[AA]$
$PC \leftarrow$ BrA	11	X	Unconditional Branch



Datapath Modification for RISC

- Beginning at the top of the datapath, zero fill has been replaced by the constant unit. The constant unit performs zero fill for CS = 0 and sign extension for CS = 1.
- MUX A is added to provide a path for the updated PC, PC-1, to the register file for implementation of the Jump and Link (JML) instruction.
- One other change in the figure helps implement the Set if Less Than (SLT) instruction. This logic provides a 1 to be loaded into R[DA] if R[AA] - R[BA] < 0 and a 0 to be loaded into R[DA] if R[AA] - R[BA] > 0.
- It is implemented by adding an additional input to MUX D. The leftmost 31 bits of the input are 0; the rightmost bit is 1 if N is 1 and V is 0 (i.e., if the result of the subtraction is negative and there is no overflow).
- It is also 1 if N is 0 and V is 1 (i.e., if the result of the subtraction is positive and there is an overflow).
- These represent all cases in which R[AA] is greater than R[BA] and can be implemented using an exclusive-OR of N and V. **(Why EXORing N & V)**