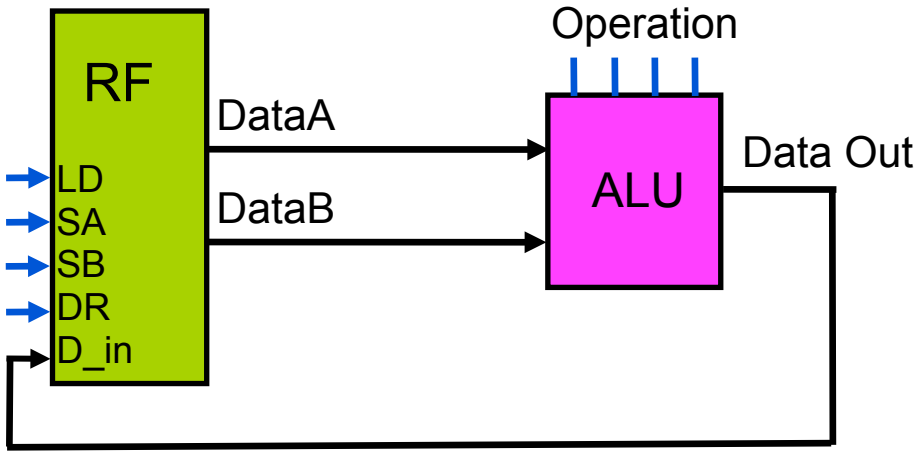
ECE 2300
Digital Logic & Computer Organization

Fall 2016

More Single Cycle Microprocessor

Cornell University

The Basic Processing Cycle

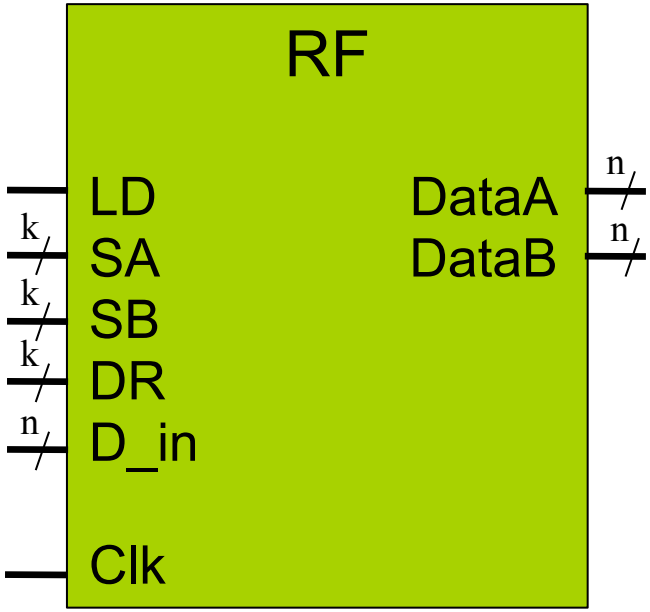


- Read data from two registers
- Perform an operation
- Place the result into a register

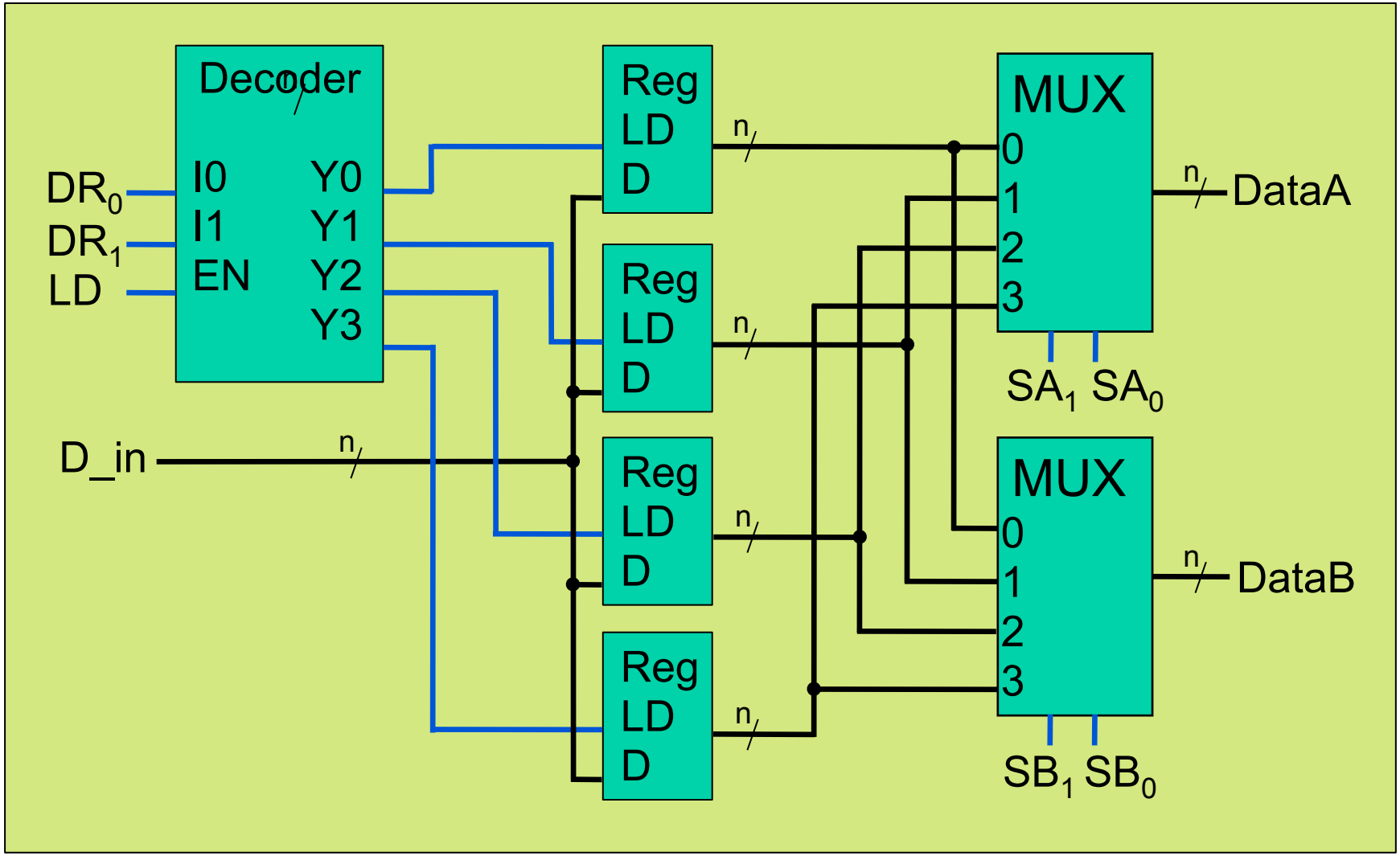
- All three steps performed in 1 clock cycle

Register File

- Collection of 2^k n-bit loadable registers
- Control inputs
  - SA – Source address A
  - SB – Source address B
  - DR – Destination address
  - LD – Load destination register with D_in
- Data inputs
  - D_in – Input data
- Data outputs
  - DataA – Output data A
  - DataB – Output data B

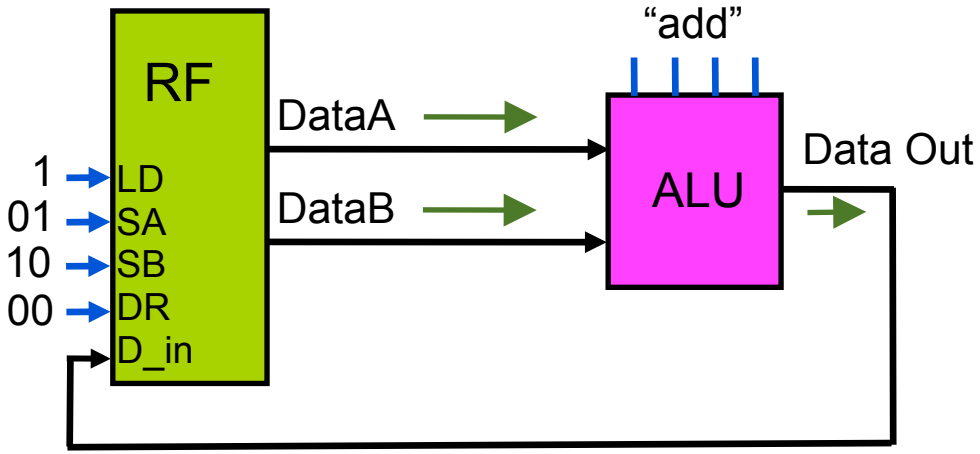


Register File



Example with 4 registers. Typically have 32 or more.

Instruction Execution



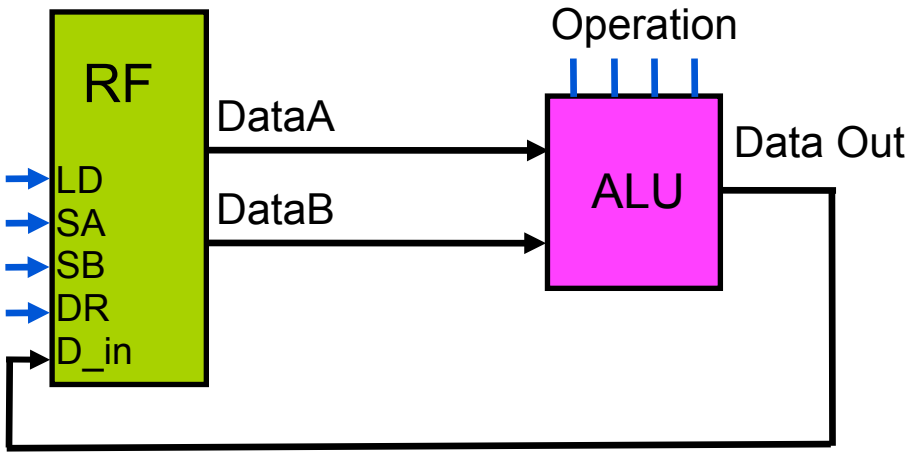
ADD R0, R1, R2

operation

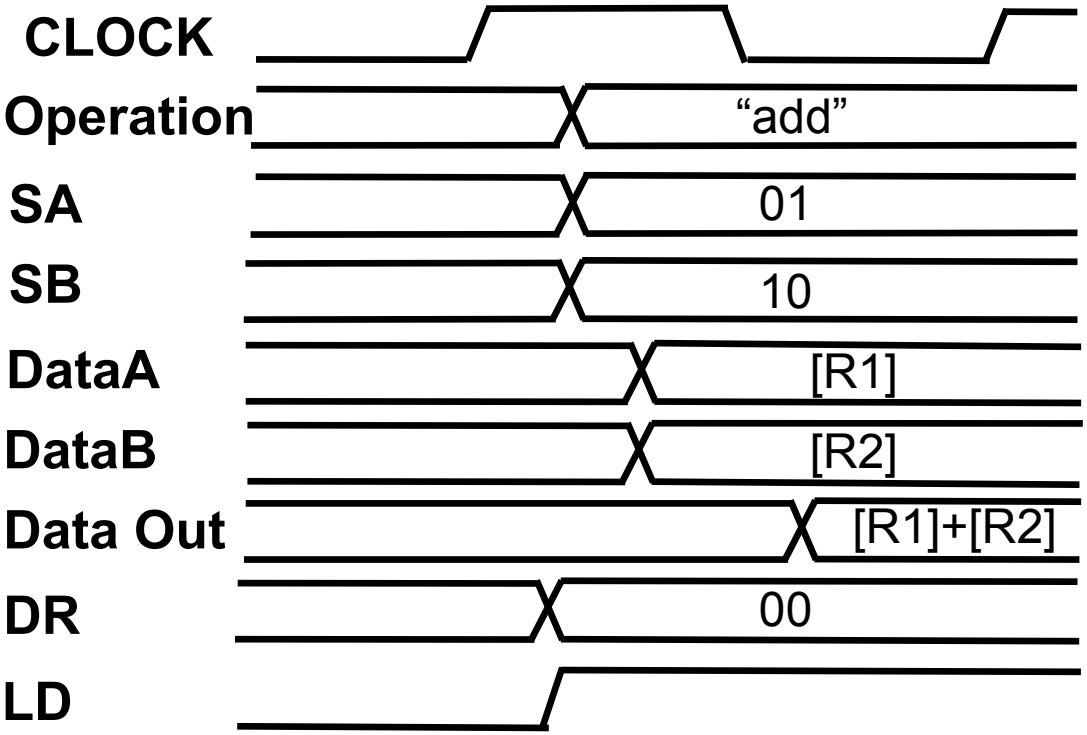
destination register

source registers

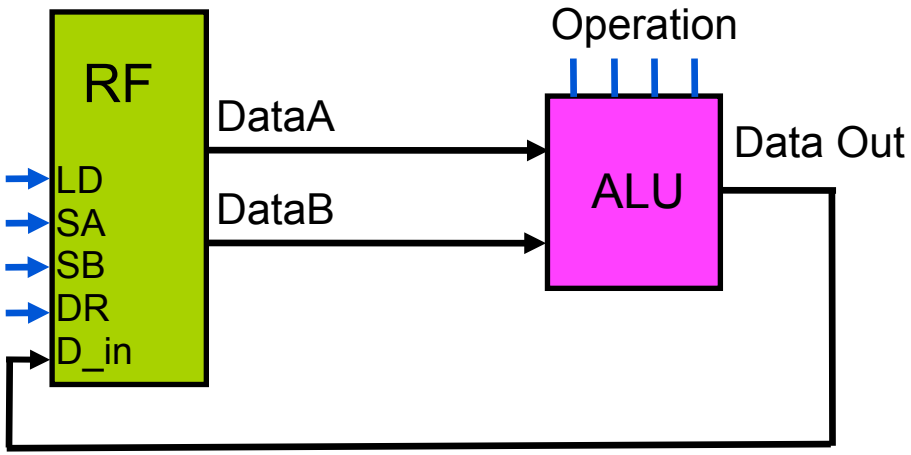
Instruction Execution



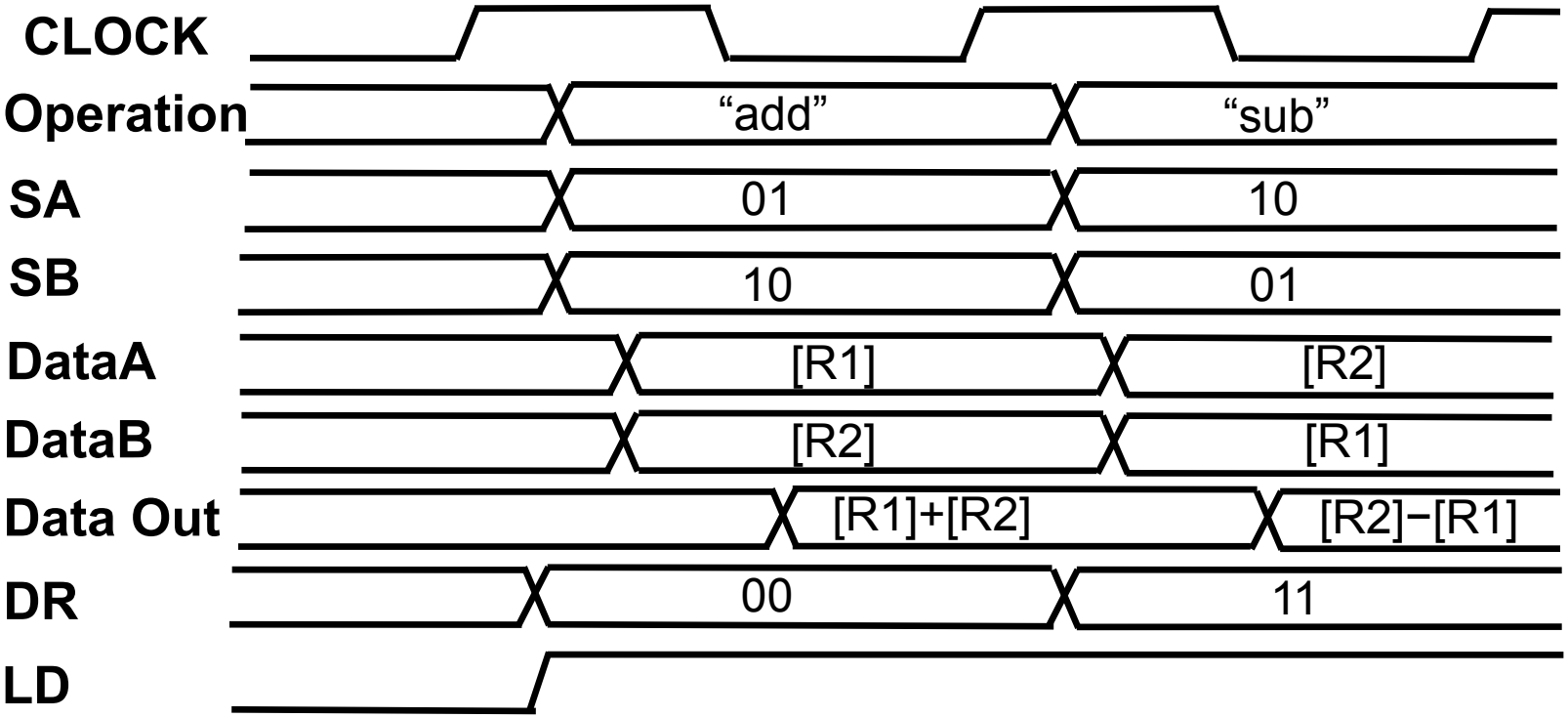
ADD R0, R1, R2



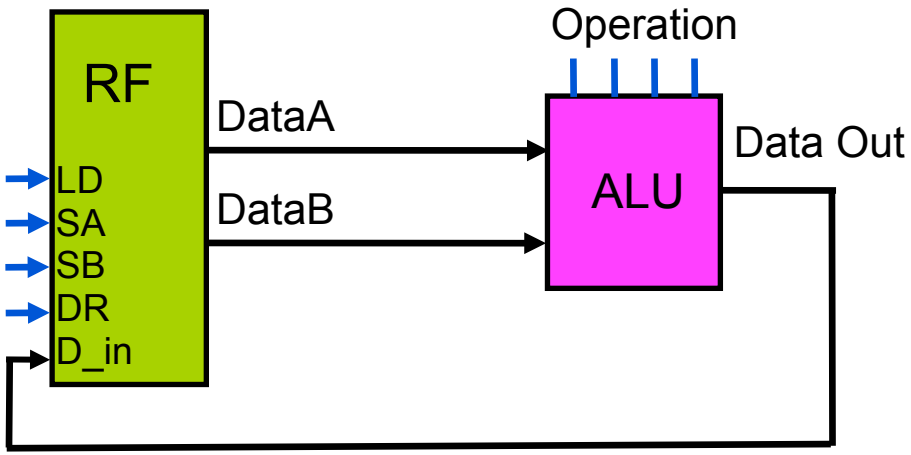
Instruction Execution



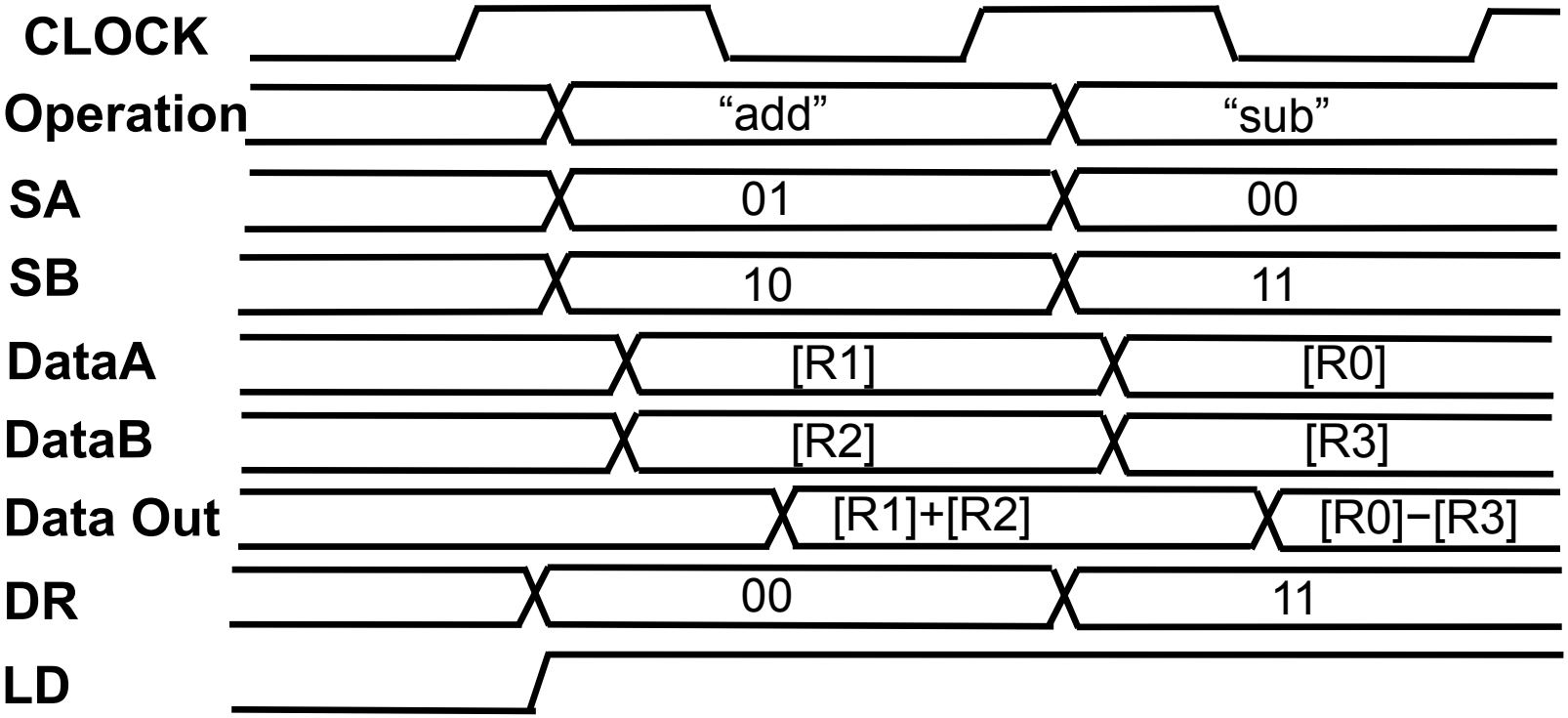
ADD R0, R1, R2
SUB R3, R2, R1



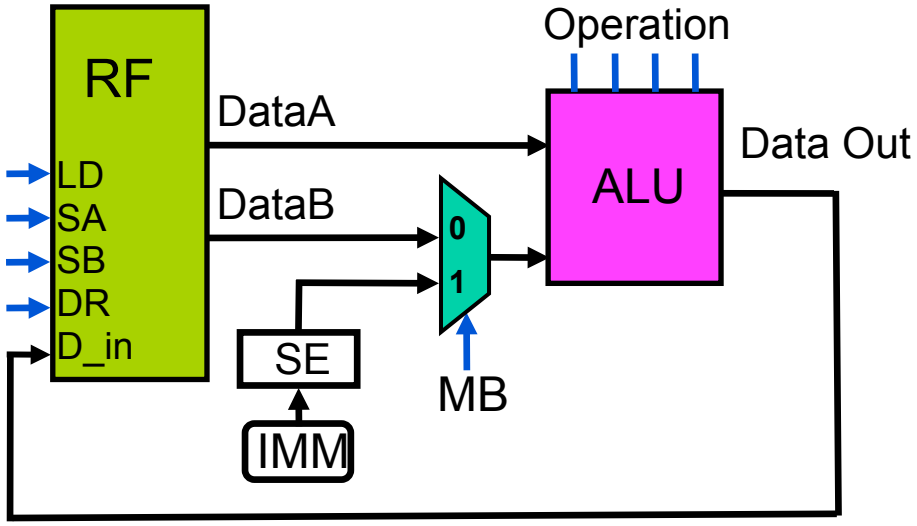
Instruction Execution



ADD R0, R1, R2
SUB R3, R0, R3



Operations With Constants

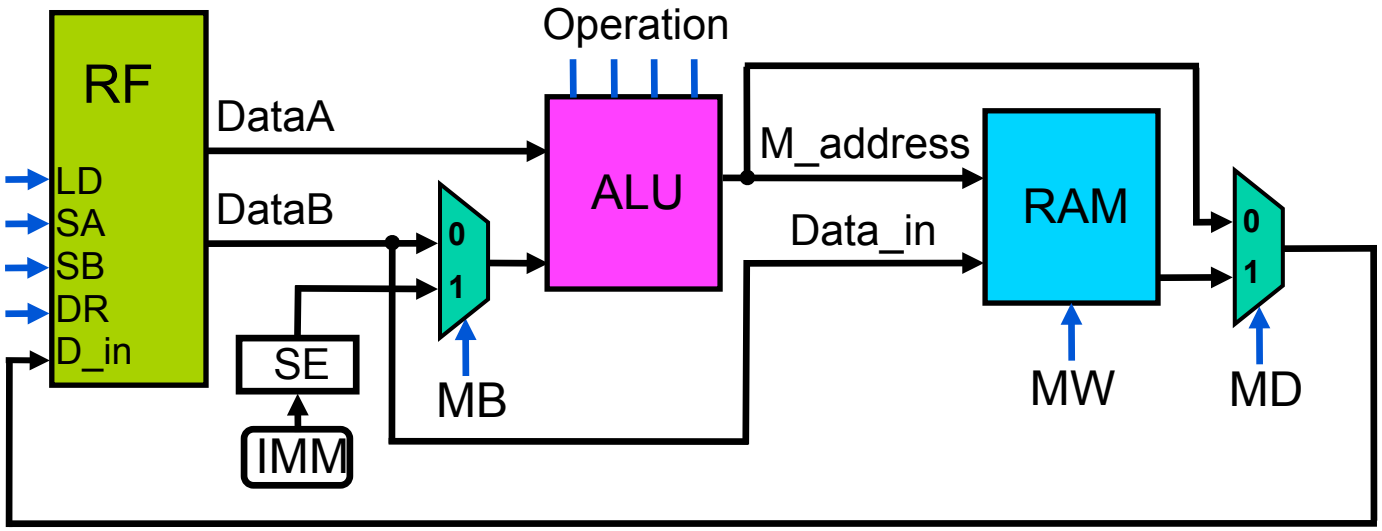


ADDI R1, R1, 1

- **Constants are called *immediate values***
- **Sign extend (SE) IMM to the width of DataA to perform correct two's complement operation**
  - **Why? May not have enough bits in instruction (later)**
  - **Assume IMM is 4 bits and DataA is 8 bits wide**

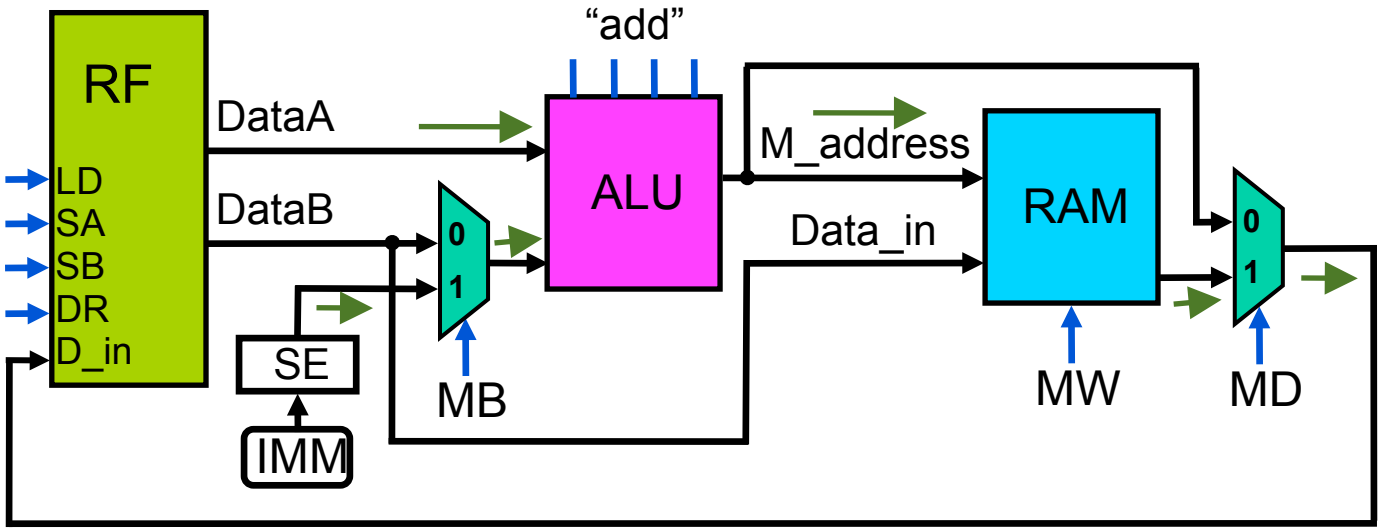
0101 → 00000101
1110 → 11111110

Reading and Writing Memory



- Most data is held in memory (RAM)
- Must be moved into a register in order to operate on it
- Data is also moved out of registers into memory
  - To make room for other data
  - To move it to permanent storage (e.g., disk)

Reading Memory ("Load")

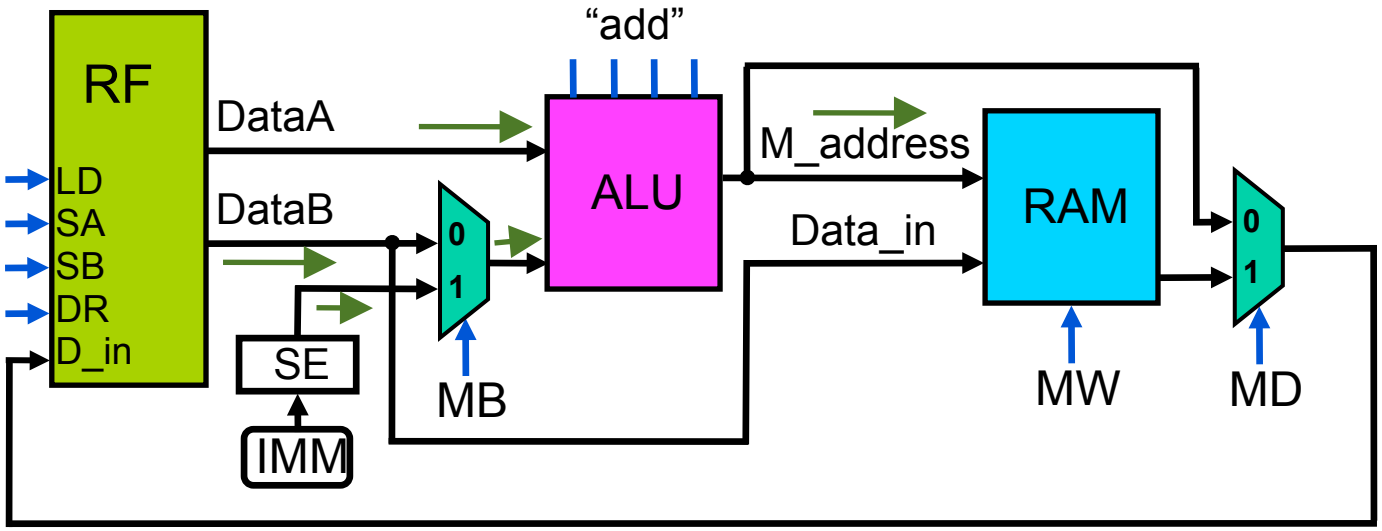


LOAD R3, 4(R1)

<u>Step 1</u>: Form the memory address by adding the value in R1 with the immediate (*offset*) 4

<u>Step 2</u>: Read the data at that address in RAM and place it in R3

Writing Memory ("Store")



STORE R2, 0(R0)

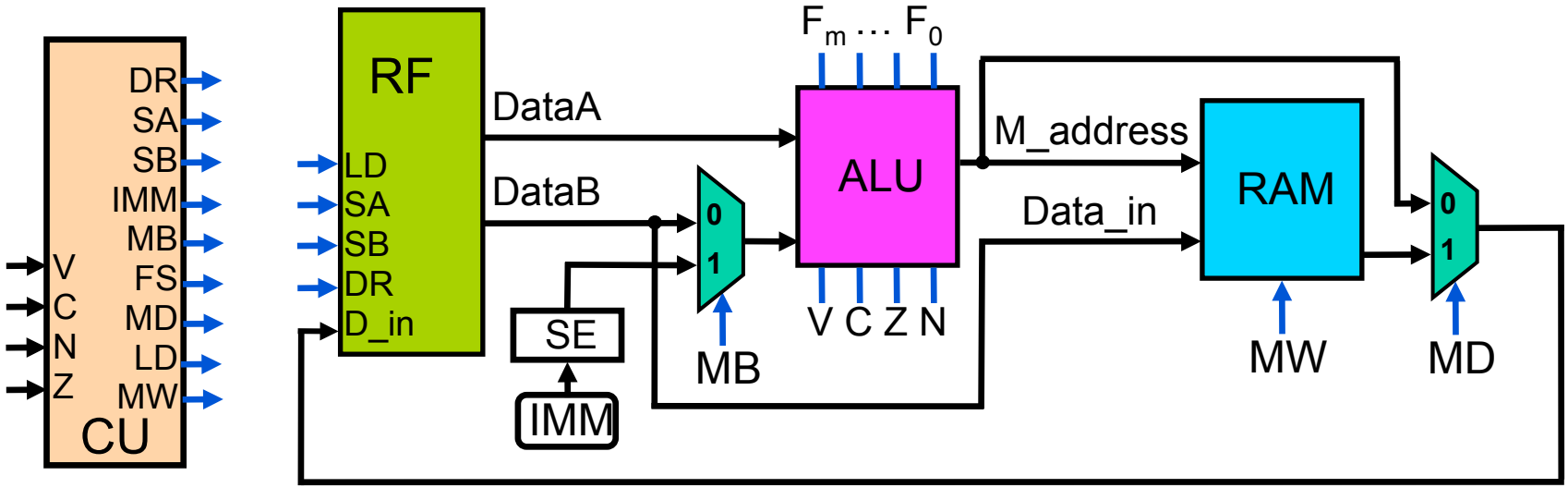
Step 1: Form the memory address by adding the value in R0 with the immediate 0

Step 2: Write the value in R2 into the RAM at that address

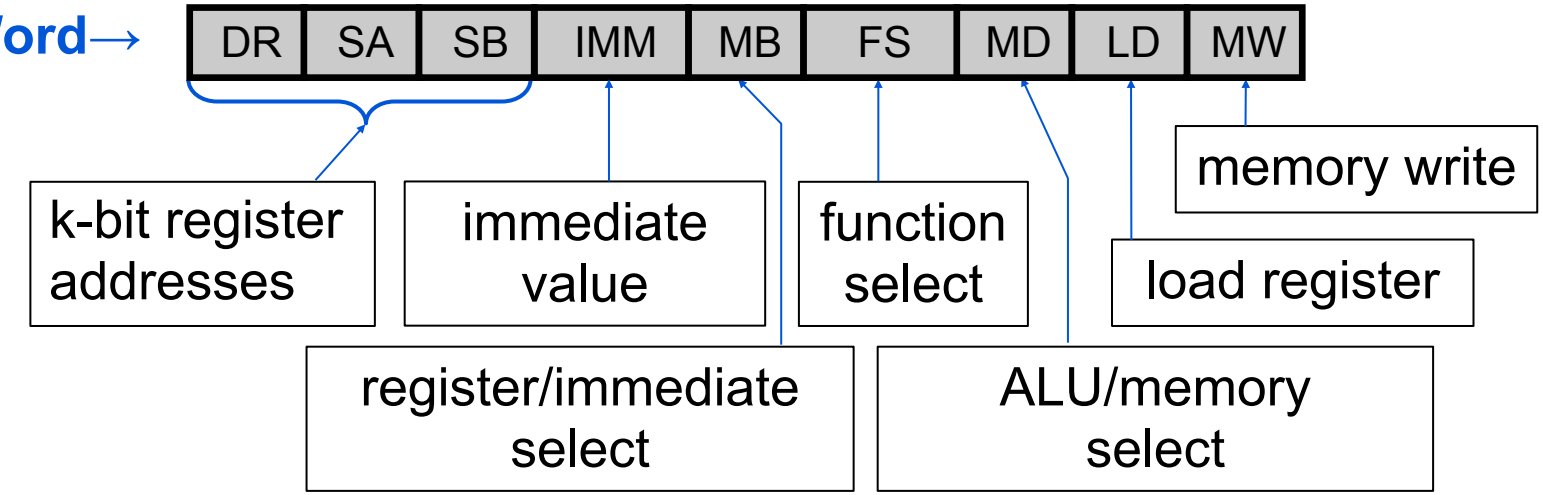
Control Unit

- Controls flow of data and operations on data
- Series of *control words* control the datapath to perform a sequence of operations
- The sequence of operations performed by the CU is affected by the ALU *Condition Codes*
  - Z: Zero
  - N: Negative
  - V: Overflow
  - C: Carry out

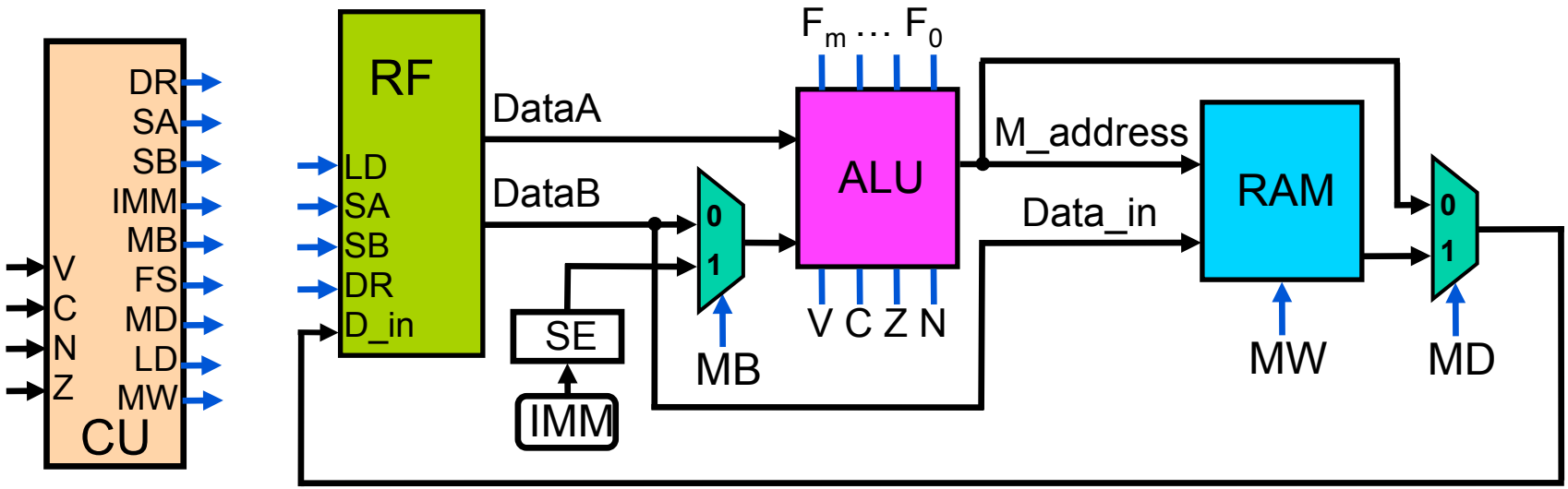
Datapath + Control Unit



Control Word→



Sequence of Operations



	DR	SA	SB	IMM	MB	FS	MD	LD	MW
R2 <= R0 + R1	010	000	001	x	0	ADD	0	1	0
R1 <= M[R2]	001	010	xxx	0	1	ADD	1	1	0
M[R2] <= R0	xxx	010	000	0	1	ADD	x	0	1
R3 <= R0 + 3	011	000	xxx	3	1	ADD	0	1	0

Shift and Add Multiplication

- Multiply $A_2A_1A_0$ by $B_2B_1B_0$

$$
\begin{array}{r}
A_2A_1A_0 \\
B_2B_1B_0 \\
\hline
(A_2A_1A_0)\times B_0 \\
(A_2A_1A_0)\quad \times B_1 \\
(A_2A_1A_0)\qquad \times B_2 \\
\hline
P_4P_3P_2P_1P_0
\end{array}
$$

- Load R1 with A from M[0]
- Load R2 with B from M[1]
- Initialize R3 (P) to 0
- If lsb of B = 1, P = P + A
- Shift A left one bit (0 into LSB)
- Shift B right one bit (0 into MSB)
- Repeat until B = 0
- Store R3 to M[2]

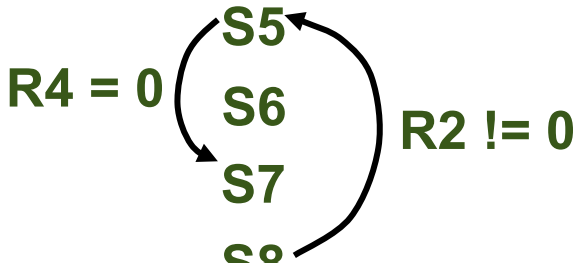
```
R1 <= M[0]              // load A
R2 <= M[1]              // load B
R3 <= 0                 // initialize P = 0

(*) R4 <= R2 & 1        // R4 = lsb(B)
    if (R4) R3 <= R3+R1 // if lsb(B)=1, add
    R1 <= SLL(R1)       // shift A left
    R2 <= SRL(R2)       // shift B right
    if (R2) goto (*)    // if B!=0, loop

    M[2] <= R3          // store P
```

Shift and Add Multiplication

R1 <= M[0]	R0 <= R0 – R0	S1
	R1 <= M[R0]	S2
R2 <= M[1]	R2 <= M[R0+1]	S3
R3 <= 0	R3 <= R3 – R3	S4
R4 <= R2 & 1	R4 <= R2 & 1	S5
R3 <= R3 + R1	R3 <= R3 + R1	S6
R1 <= SLL(R1)	R1 <= SLL(R1)	S7
R2 <= SRL(R2)	R2 <= SRL(R2)	S8
M[2] <= R3	M[R0+2] <= R3	S9

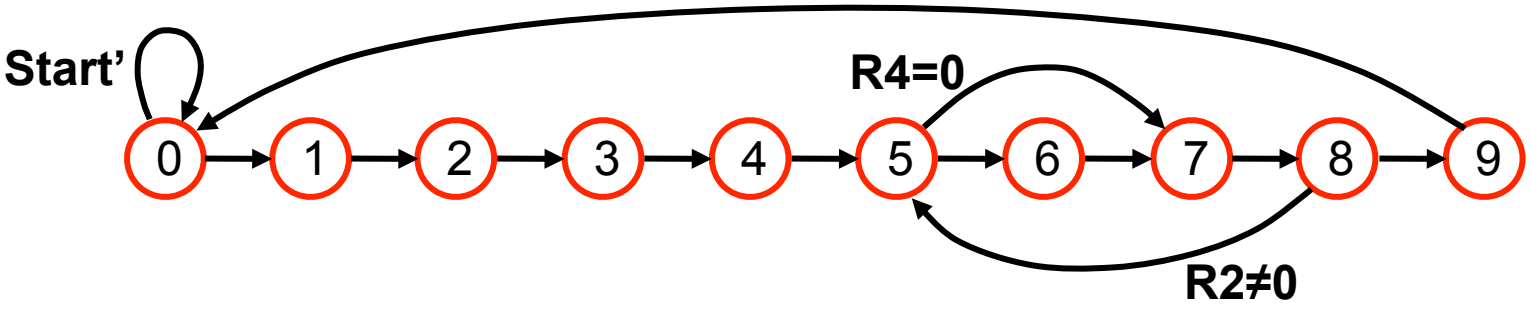


- **Can implement with a 9 state FSM**
  - Inputs are the values of R4 and R2
  - Outputs are the control words that carry out the above operations

Shift and Add Multiplication

S1	R0 <= R0 – R0
S2	R1 <= M[R0]
S3	R2 <= M[R0+1]
S4	R3 <= R3 – R3
S5	R4 <= R2 & 1
S6	R3 <= R3 + R1
S7	R1 <= SLL(R1)
S8	R2 <= SRL(R2)
S9	M[R0+2] <= R3

DR	SA	SB	IMM	MB	FS	MD	LD	MW
000	000	000	x	0	SUB	0	1	0
001	000	xxx	0	1	ADD	1	1	0
010	000	xxx	1	1	ADD	1	1	0
011	011	011	x	0	SUB	0	1	0
100	010	xxx	1	1	AND	0	1	0
011	011	001	x	0	ADD	0	1	0
001	001	xxx	x	x	SLL	0	1	0
010	010	xxx	x	x	SRL	0	1	0
xxx	000	011	2	1	ADD	x	0	1



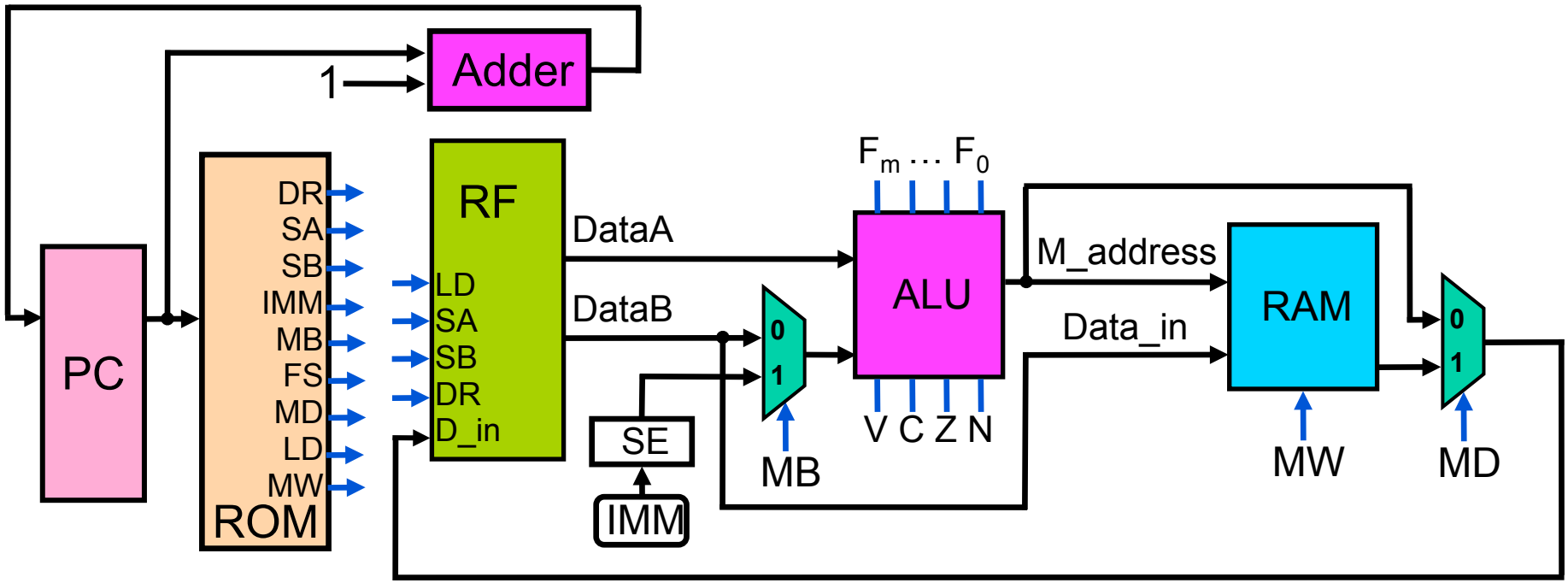
Programmable Control Unit

- Datapath (RF, ALU, RAM, muxes) is flexible
- However, Multiplier Control Unit is “hardwired”
  - New operation requires new state machine
- Programmable Control Unit
  - Control words still stored in ROM
  - State machine replaced by a *Program Counter* plus *branch operations*
  - Flexible: Can run multiple sequences of control words (*programs*) stored in ROM

Program Counter (PC)

- Special register that points to the location (address) in ROM of the next control word
- Updated every clock cycle
- Sequential execution
  - Control words read from sequential ROM locations
  - PC is increased by 1 after each control word read
- Branch operations
  - Special control flow operations
  - *Condition code* determines whether to branch or not
  - If so, next ROM address is *PC* + *branch offset*

PC-Based Control Unit

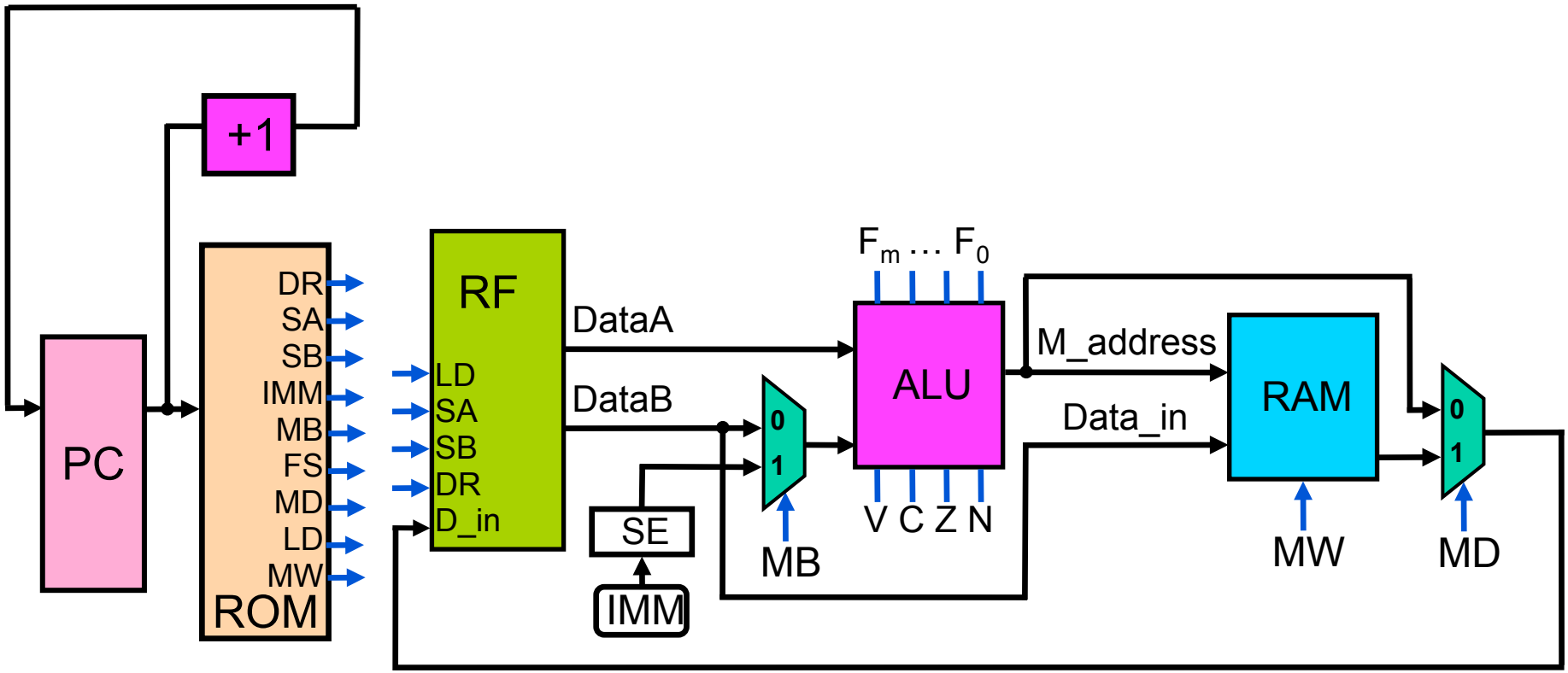


		DR	SA	SB	IMM	MB	FS	MD	LD	MW
R2 <= R0 + R1	a:	010	000	001	x	0	ADD	0	1	0
R1 <= M[R2]	a+1:	001	010	xxx	0	1	ADD	1	1	0
M[R2] <= R0	a+2:	xxx	010	000	0	1	ADD	x	0	1
R3 <= R0 + 3	a+3:	011	000	xxx	3	1	ADD	0	1	0

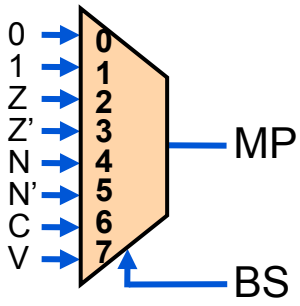
Branch Operations

- Used to jump to a different part of the program
- Consists of a *condition* and an *offset*
- Condition
  - Checks to see whether the result of the last instruction met a specified criteria, such as
    - Zero (Z = 1)
    - Negative (N = 1)
- Offset
  - How far ahead, or back, to jump within the program if the condition is met

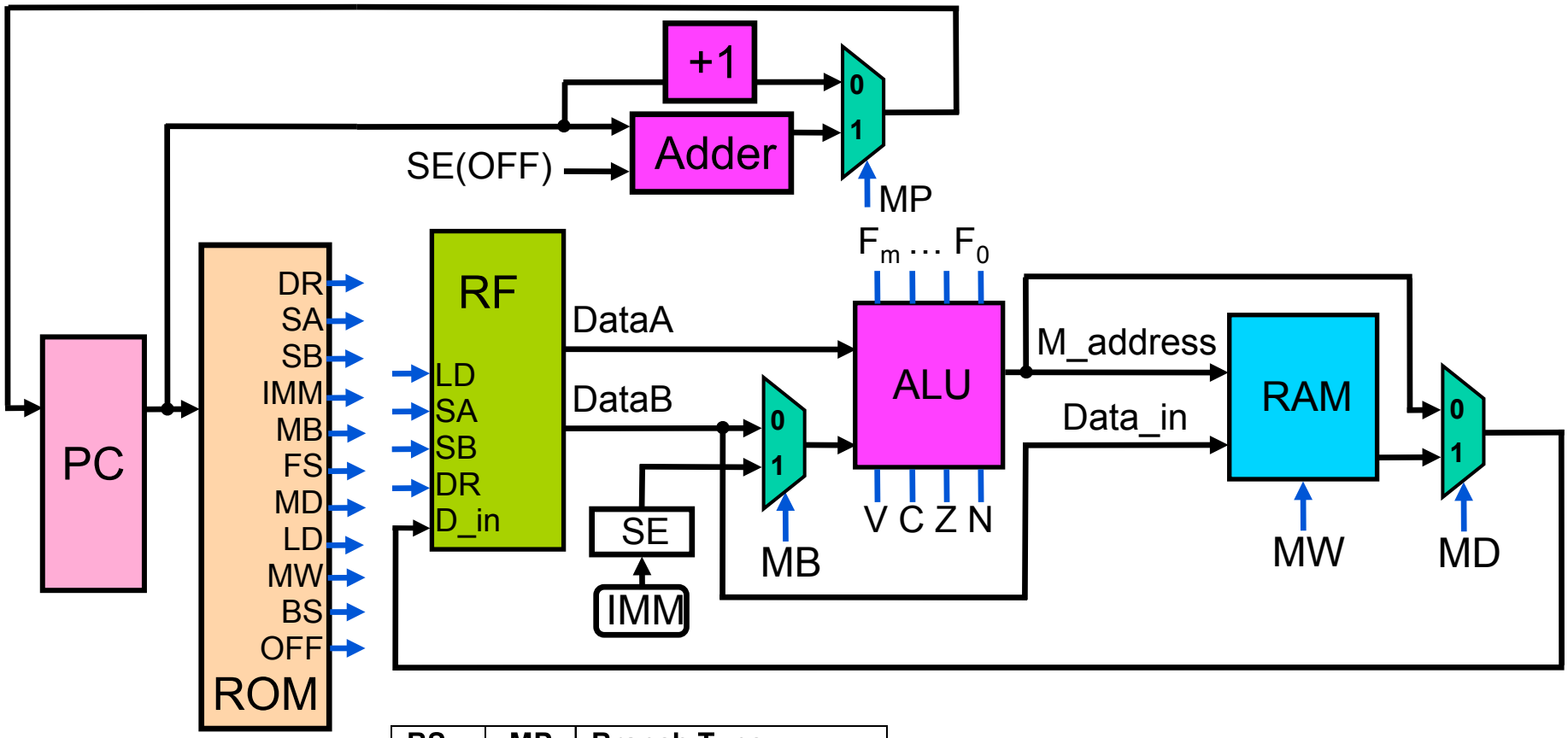
Branch Operations



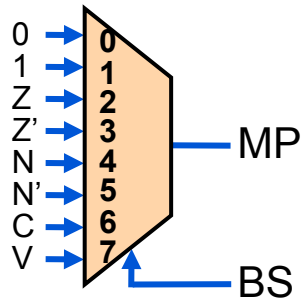
BS	MP	Branch Type
000	0	*Branch Never*
001	1	*Branch Always*
010	Z	*Branch if Zero*
011	Z'	*Branch if Not Zero*
100	N	*Branch if < Zero*
101	N'	*Branch if >= Zero*
110	C	*Branch if Carry Out*
111	V	*Branch if Overflow*



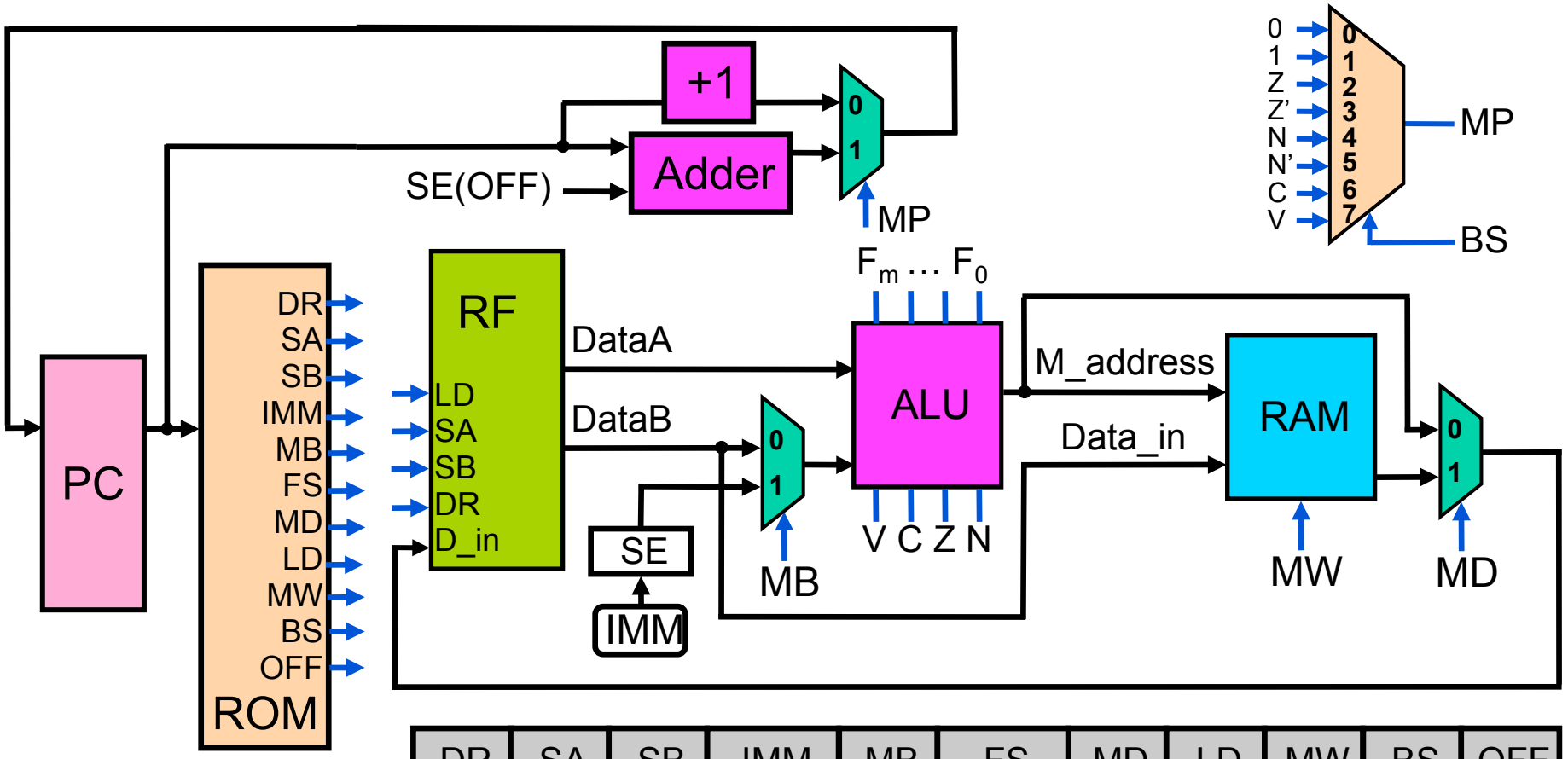
Branch Operations



BS	MP	Branch Type
000	0	*Branch Never*
001	1	*Branch Always*
010	Z	*Branch if Zero*
011	Z'	*Branch if Not Zero*
100	N	*Branch if < Zero*
101	N'	*Branch if >= Zero*
110	C	*Branch if Carry Out*
111	V	*Branch if Overflow*

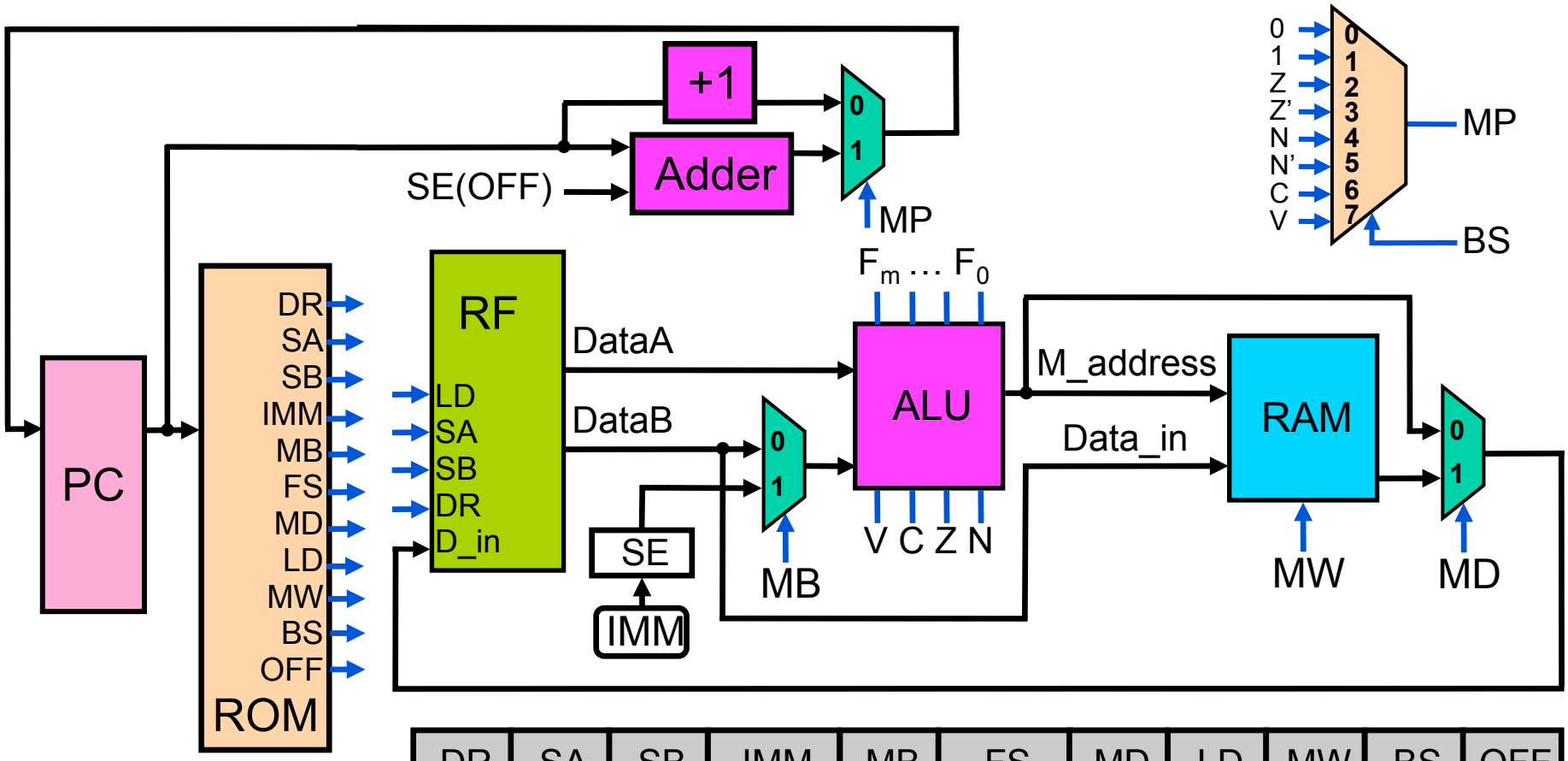


Branch Operations



	DR	SA	SB	IMM	MB	FS	MD	LD	MW	BS	OFF
R4 <= R2 & 1 — 4:	100	010	x	1	1	AND	0	1	0	000	x
if (R4=0) goto 7 — 5:	x	100	x	0	1	SUB	x	0	0	010	2
R3 <= R3 + R1 — 6:	011	011	001	x	0	ADD	0	1	0	000	x
R1 <= SLL(R1) — 7:	001	001	x	x	x	SLL	0	1	0	000	x

Branch Operations



	DR	SA	SB	IMM	MB	FS	MD	LD	MW	BS	OFF
R2 <= SRL(R2) 8:	010	010	x	x	x	SRL	0	1	0	000	x
if (R2≠0) goto 4 9:	x	010	x	0	1	SUB	x	0	0	011	-5
M[R0+2] <= R3 10:	x	000	011	2	1	ADD	x	0	1	000	x

Programmable Multiplication ROM

	DR	SA	SB	IMM	MB	FS	MD	LD	MW	BS	OFF
0: R0 <= R0 – R0	000	000	000	x	0	SUB	0	1	0	000	x
1: R1 <= M[R0]	001	000	x	0	1	ADD	1	1	0	000	x
2: R2 <= M[R0+1]	010	000	x	1	1	ADD	1	1	0	000	x
3: R3 <= R3 – R3	011	011	011	x	0	SUB	0	1	0	000	x
4: R4 <= R2 & 1	100	010	x	1	1	AND	0	1	0	000	x
5: if (R4=0) goto 7	x	100	x	0	1	SUB	x	0	0	010	2
6: R3 <= R3 + R1	011	011	001	x	0	ADD	0	1	0	000	x
7: R1 <= SLL(R1)	001	001	x	x	x	SLL	0	1	0	000	x
8: R2 <= SRL(R2)	010	010	x	x	x	SRL	0	1	0	000	x
9: if (R2≠0) goto 4	x	010	x	0	1	SUB	x	0	0	011	-5
10: M[R0+2] <= R3	x	000	011	2	1	ADD	x	0	1	000	x

Before Next Class

- H&H 7.5

Next Time

More Single Cycle Microprocessor

Pipelined Microprocessor