ECE 2300
Digital Logic & Computer Organization

Spring 2025

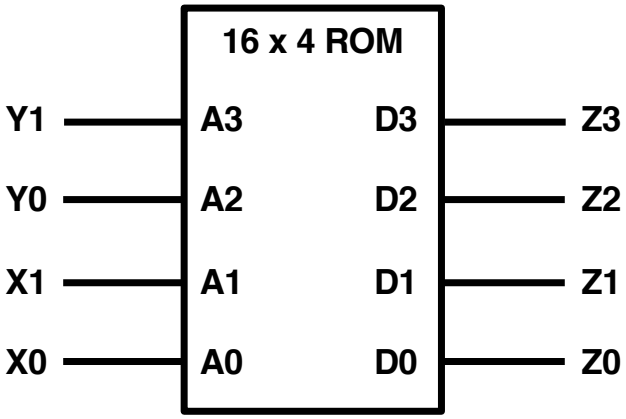
More Single Cycle Microprocessor

Cornell University

Announcements

- Lab 3b due tomorrow
- HW6 Q4 diagram updated

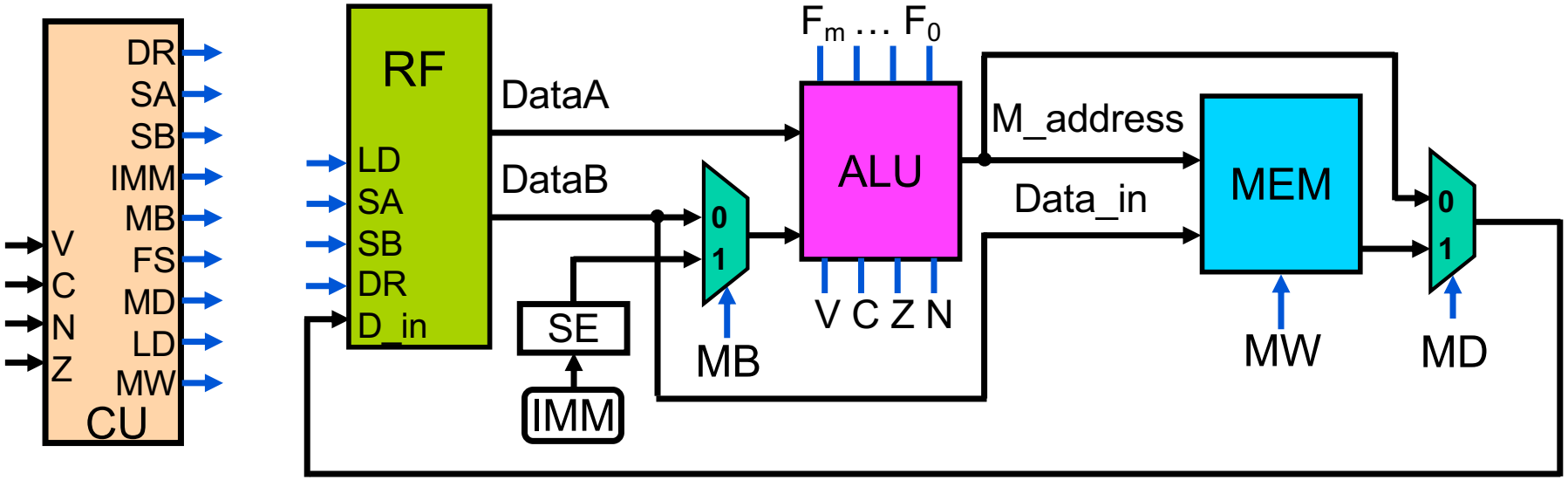
Exercise: Using Memory for Logic



Y1, Y0, X1, X0 (address)	Z3, Z2, Z1, Z0 (output)
0000	0000
0001	0000
0010	0000
0011	0000
0100	0000
0101	0001
0110	0010
0111	0011
1000	0000
1001	0010
1010	0100
1011	0110
1100	0000
1101	0011
1110	0110
1111	1001

What's the function of this ROM?

Review: Control Word



	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	x	0	1	ADD	1	1	0
M[R2] <= R0	x	010	000	0	1	ADD	x	0	1
R3 <= R1 – 3	011	001	x	3	1	SUB	0	1	0

Unsigned Multiplication

- Form each partial product by multiplying a single digit of the multiplier by the multiplicand
- Add shifted partial products to get result

multiplicand (= 5)	101
multiplier (= 3)	x 011
	101
partial products	101
	000
result (= 15)	01111

Iterative (Shift and Add) Multiplication

- Multiply $A_2A_1A_0$ by $B_2B_1B_0$

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

P = AxB

Assumptions:
(1) A, B are initially in memory at M[0] and M[1], respectively;
(2) P will be put back to memory to M[2]
(3) Registers in RF are 8-bit wide

Pseudo code

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

// M: main memory
// LSB: least significant bit
// MSB: most significant bit
```

Iterative Multiplication

- Multiply $A_2A_1A_0$ by $B_2B_1B_0$

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

P = AxB

Assumptions:
(1) A, B are initially in memory at M[0] and M[1], respectively;
(2) P will be put back to memory to M[2]
(3) Registers in RF are 8-bit wide

Refined Pseudo code

```
    R1 <= M[0]                  // Load A from M[0]
    R2 <= M[1]                  // Load B from M[1]
    R3 <= 0                     // Initialize P (R3) = 0

(*) R4 <= R2 & 1                // R4 = LSB(B)
    if (R4≠0) R3 <= R3+R1       // If LSB(B)=1, P=P+A
    R2 <= R2 >> 1               // Shift B right
    R1 <= R1 << 1               // Shift A left
    if (R2≠0) goto (*)          // Repeat until B=0

    M[2] <= R3                  // Store P to M[2]
```

Iterative Multiplication

- **Multiply $A_2A_1A_0$ by $B_2B_1B_0$**

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

Refined Pseudo code

```
R1 <= M[0]                  // Load A from M[0]
R2 <= M[1]                  // Load B from M[1]
R3 <= 0                     // Initialize P (R3) = 0

(*) R4 <= R2 & 1            // R4 = LSB(B)
    if (R4≠0) R3 <= R3+R1   // If LSB(B)=1, P=P+A
    R2 <= R2 >> 1           // Shift B right
    R1 <= R1 << 1           // Shift A left
    if (R2≠0) goto (*)      // Repeat until B=0

    M[2] <= R3              // Store P to M[2]
```

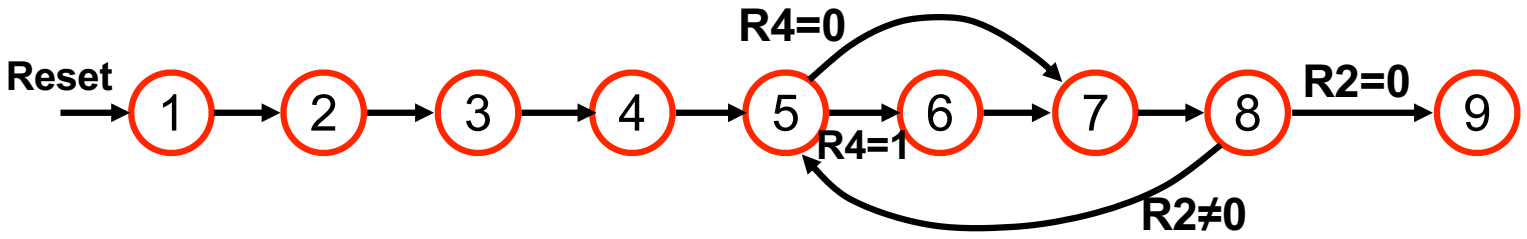
Multiply A=101 by B=011: Determine the values of R1~R4 at each iteration
(Here, a new iteration starts each time the line labeled * runs)

Iteration ↓

R1 (A)	R2 (B)	R3 (P)	R4 (LSB of B)
00000101	00000011	00000000	

Operation Sequence for Iterative Multiplication

		DR	SA	SB	IMM	MB	FS	MD	LD	MW
S1	R0 <= R0-R0	000	000	000	x	0	SUB	0	1	0
S2	R1 <= M[R0]	001	000	x	0	1	ADD	1	1	0
S3	R2 <= M[R0 + 1]	010	000	x	1	1	ADD	1	1	0
S4	R3 <= R3-R3	011	011	011	x	0	SUB	0	1	0
S5	R4 <= R2 & 1	100	010	x	1	1	AND	0	1	0
S6	R3 <= R3 + R1	011	011	001	x	0	ADD	0	1	0
S7	R2 <= R2 >> 1	010	010	x	x	x	SRL	0	1	0
S8	R1 <= R1 << 1	001	001	x	x	x	SLL	0	1	0
S9	M[R0 + 2] <= R3	x	000	011	2	1	ADD	x	0	1

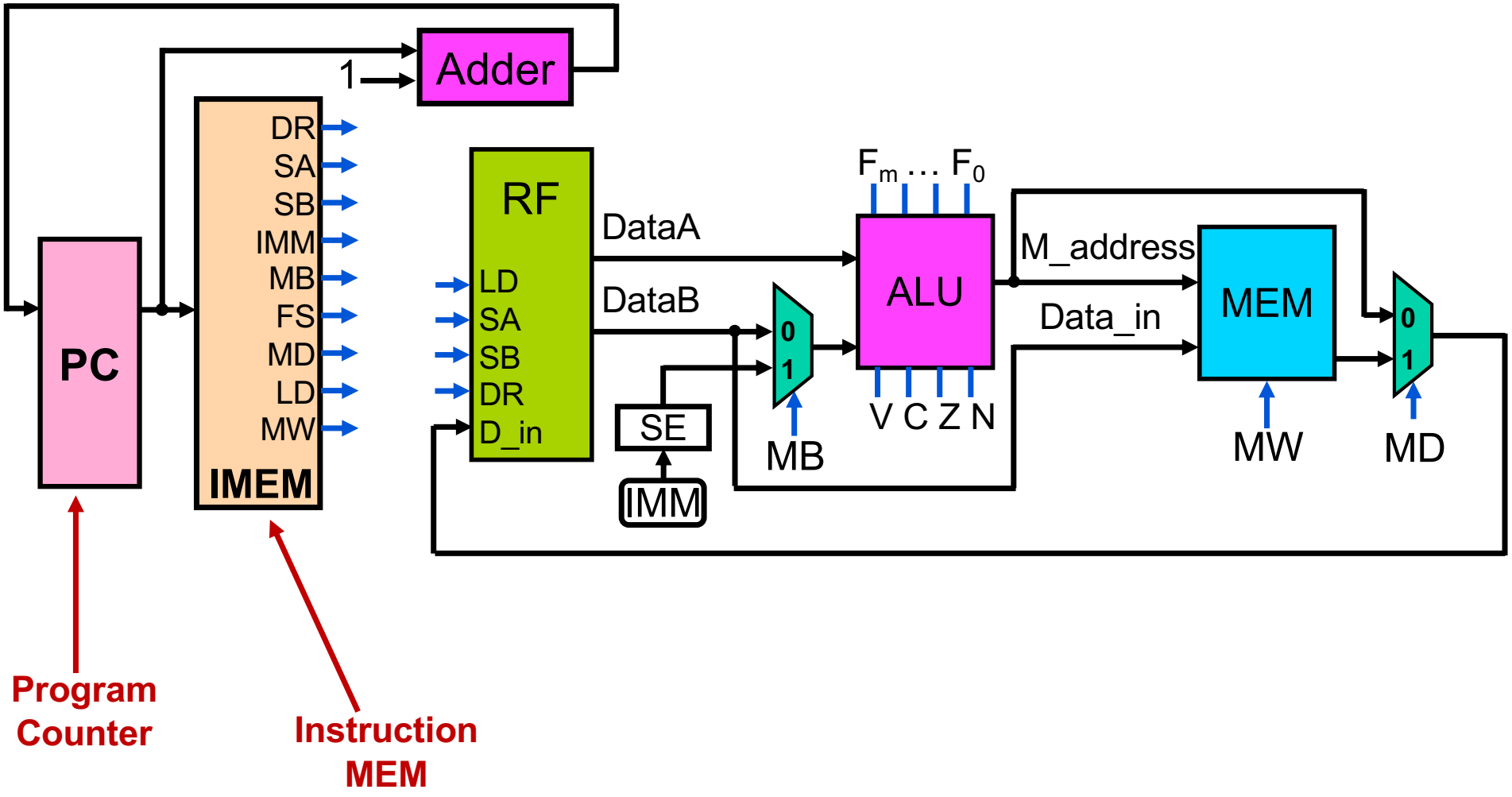


FSM-based control unit (hardwired)

Programmable Control Unit

- Datapath (RF, ALU, RAM, muxes) is flexible
- However, FSM-based multiplier control unit is "hardwired"
  - New operation sequence would require new state machine
- Programmable control unit
  - Control words stored in RAM
  - State machine replaced by a Program Counter (PC) plus *branch operations*
  - Flexible: Can run different sequences of control words (*programs*)

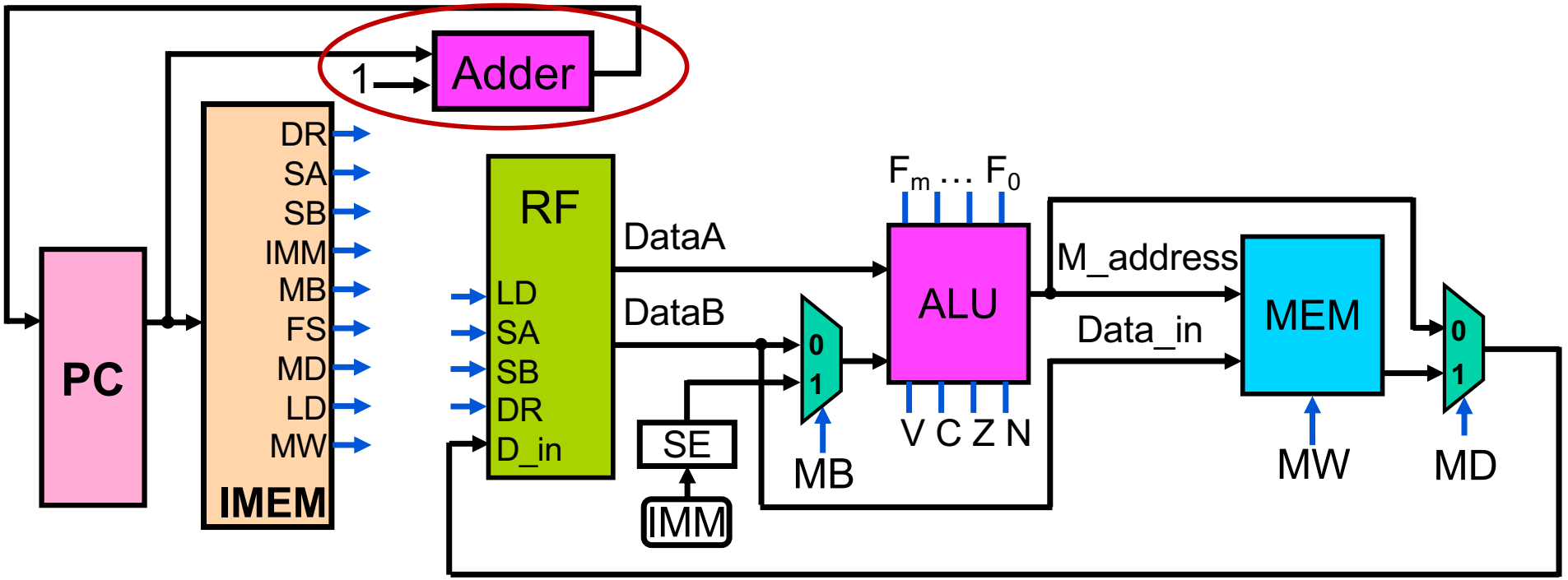
PC-Based Control Unit



Program Counter (PC)

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

PC-Based Control Unit



Increment PC

R0 <= R0 – R0

R1 <= M[R0]

R2 <= M[R0+1]

R3 <= R3 – R3

R4 <= R2 & 1

0:
1:
2:
3:
4:

Instruction Address

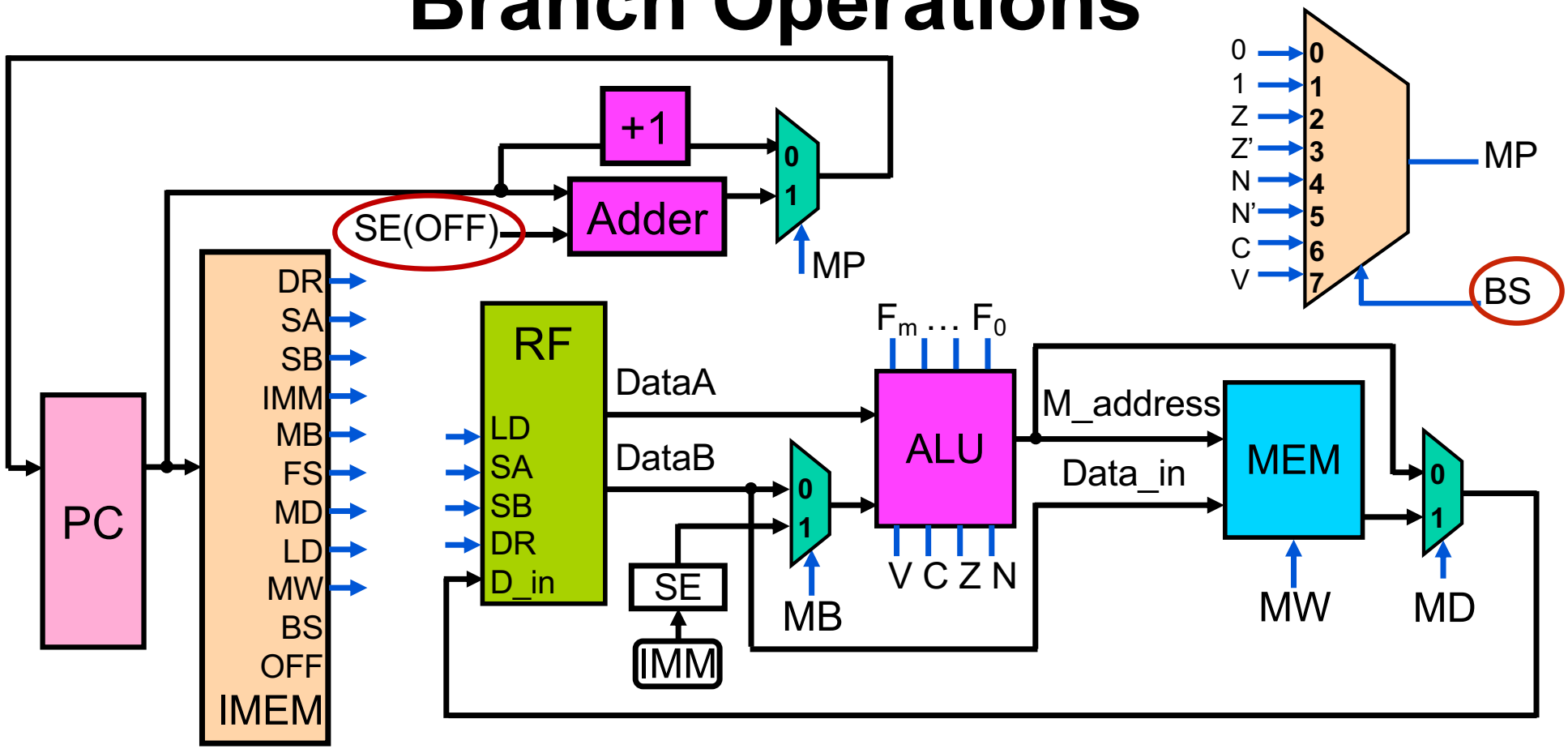
DR	SA	SB	IMM	MB	FS	MD	LD	MW
000	000	000	x	0	SUB	0	1	0
001	000	x	0	1	ADD	1	1	0
010	000	x	1	1	ADD	1	1	0
011	011	011	x	0	SUB	0	1	0
100	010	x	1	1	AND	0	1	0

Control words stored in a RAM

Branch Operations

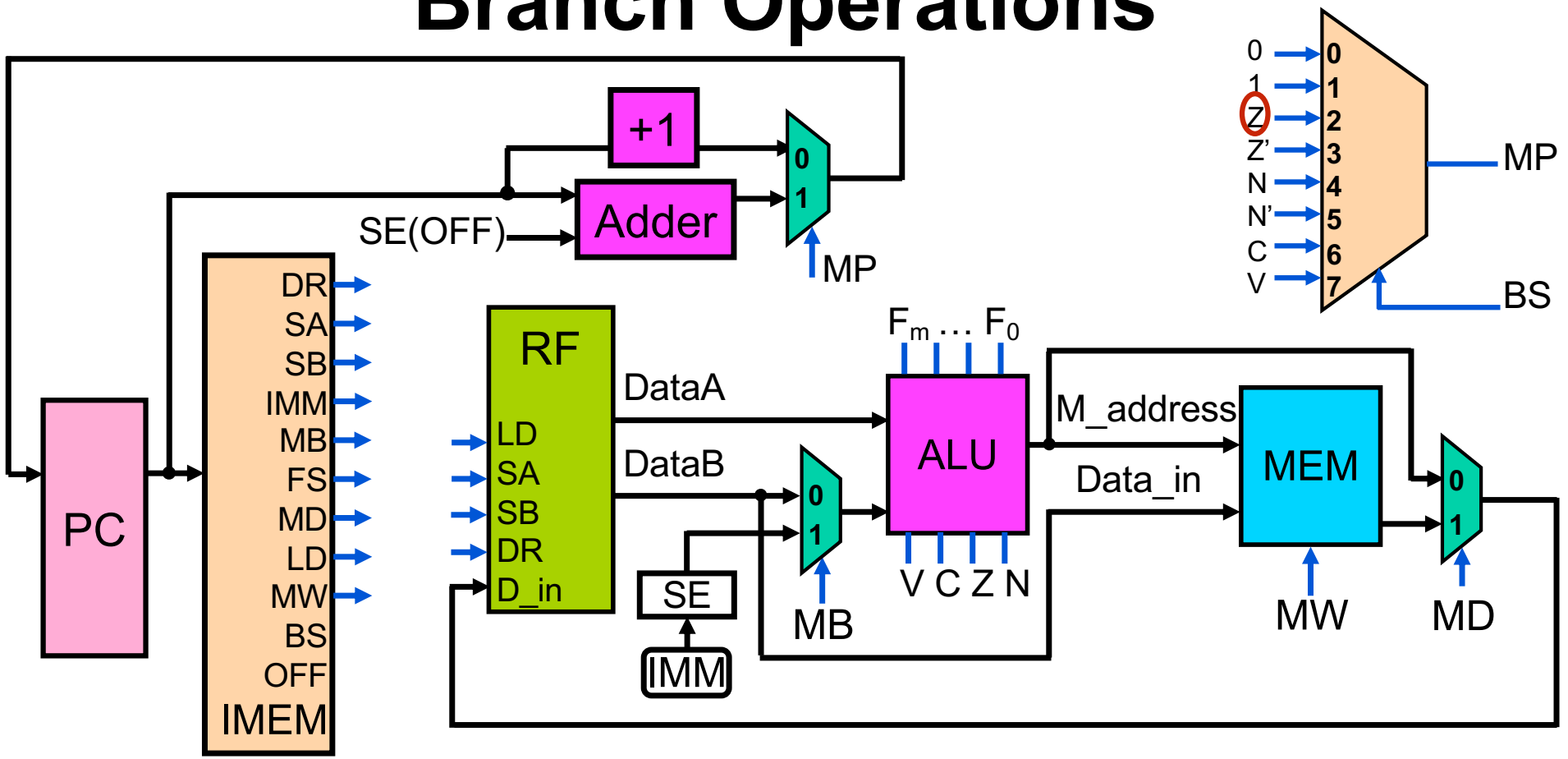
- Used to jump to a different part of the program
- Consists of a *condition* and an *offset*
- Condition
  - Checks to see whether the result of the last instruction met a specified criteria, such as
    - Zero (Z = 1), Negative (N = 1)
- Offset
  - How far ahead, or back, to jump within the program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

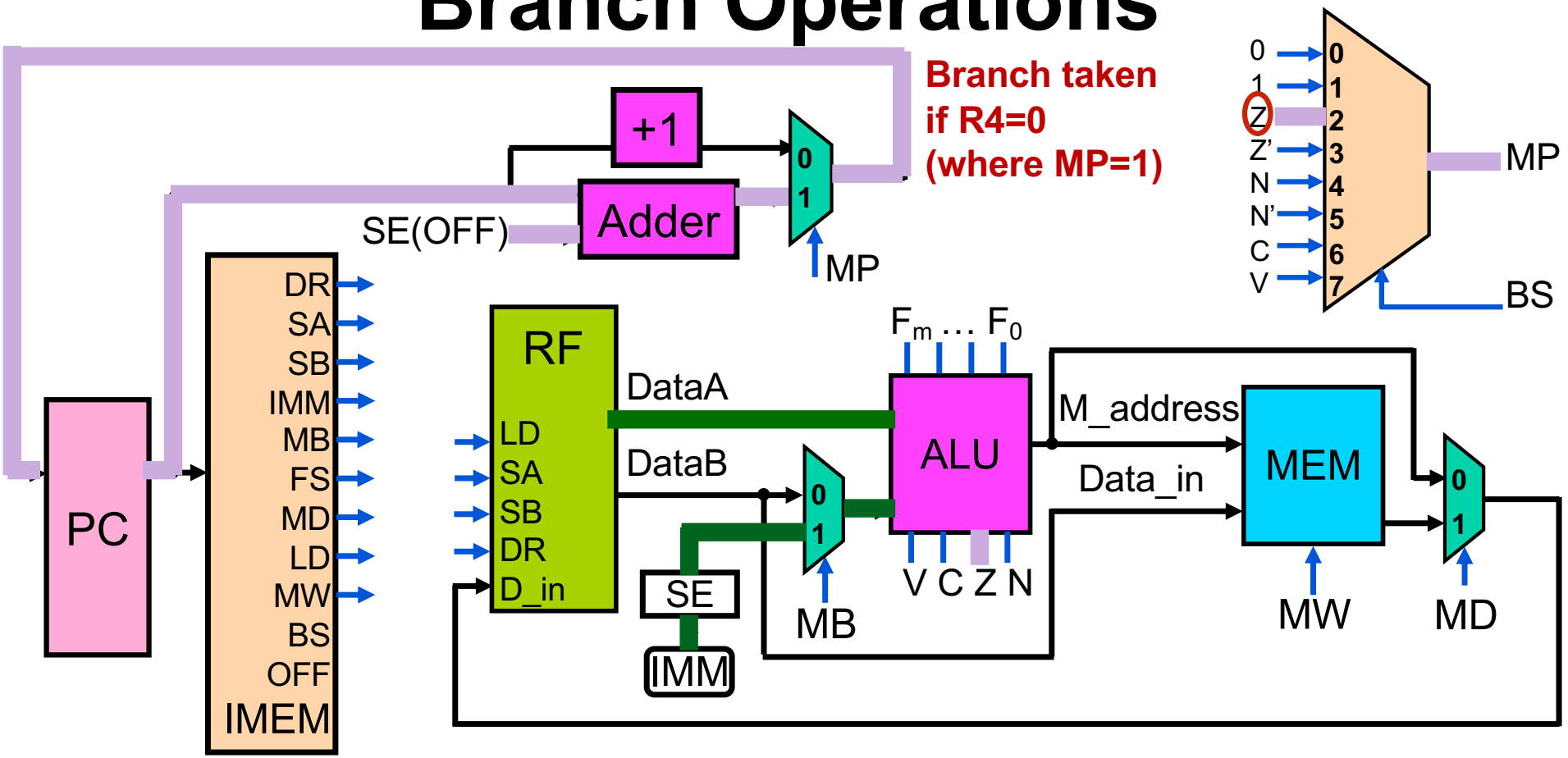
Branch Operations



		DR	SA	SB	IMM	MB	FS	MD	LD	MW	BS	OFF
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
R2 <= R2 >> 1	7:	010	010	x	x	x	SRL	0	1	0	000	x

Branch if zero

Branch Operations



if (R4=0) goto 7

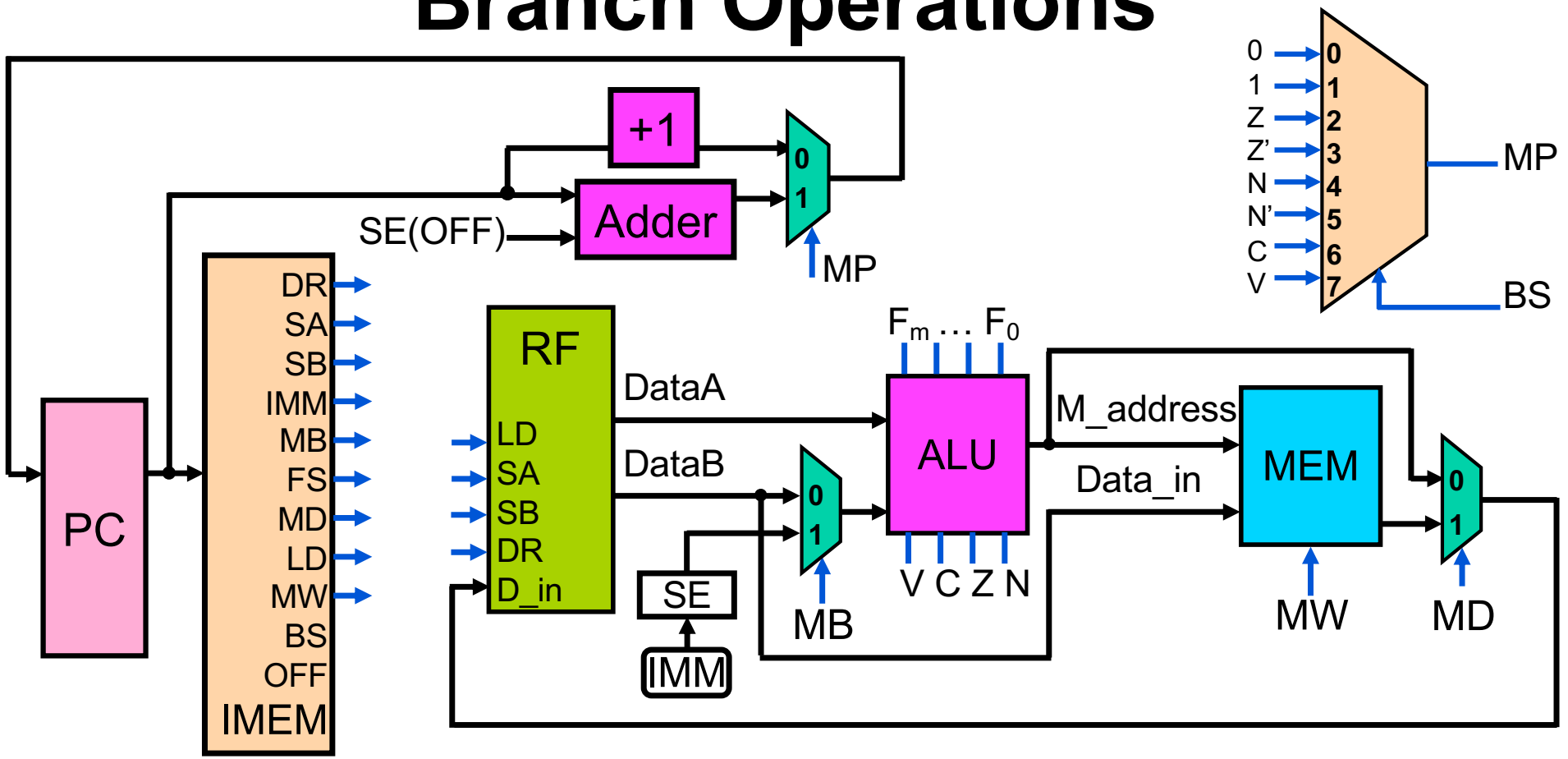
R3 <= R3 + R1

R2 <= R2 >> 1

	DR	SA	SB	IMM	MB	FS	MD	LD	MW	BS	OFF
5:	x	100	x	0	1	SUB	x	0	0	010	2
6:	011	011	001	x	0	ADD	0	1	0	000	x
7:	010	010	x	x	x	SRL	0	1	0	000	x

Branch if zero

Branch Operations



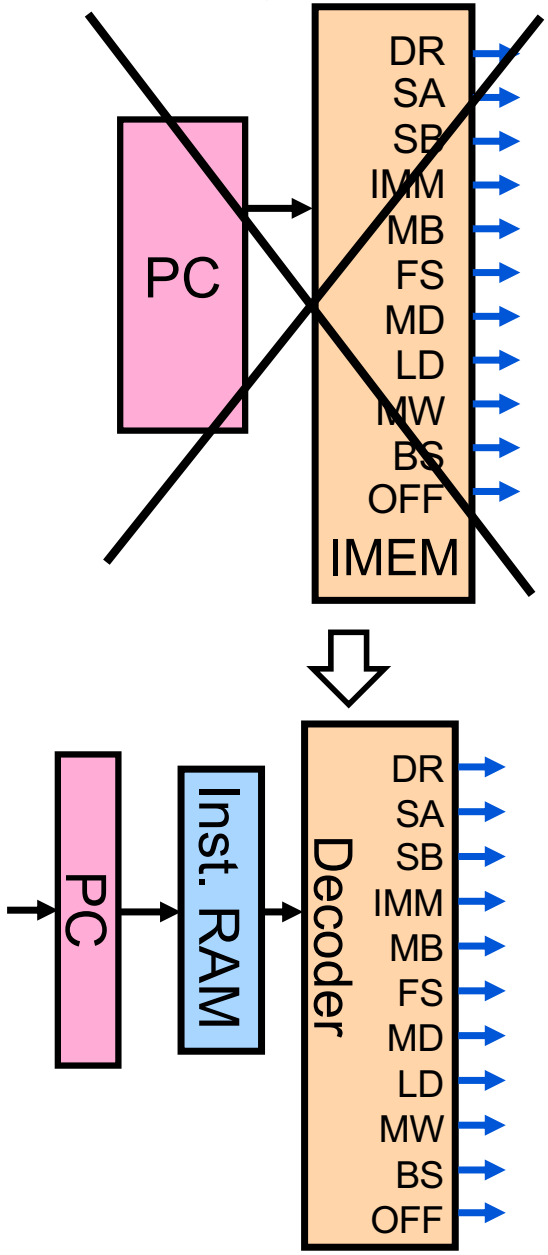
		DR	SA	SB	IMM	MB	FS	MD	LD	MW	BS	OFF
R1 <= R1 << 1	8:	001	001	x	x	x	SLL	0	1	0	000	x
if (R2≠0) goto 4	9:											
M[R0+2] <= R3	10:	x	000	011	2	1	ADD	x	0	1	000	x

Programmable Multiplication

	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: R2 <= R2 >> 1	010	010	x	x	x	SRL	0	1	0	000	x
8: R1 <= R1 << 1	001	001	x	x	x	SLL	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

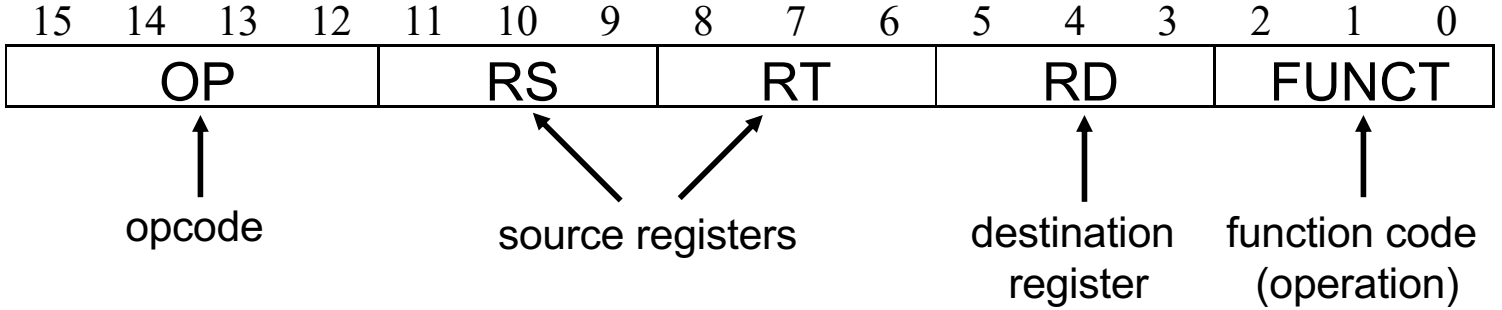
Providing Even More Flexibility

- **Replace control words with instructions**
  - **More intuitive and not specific to particular hardware**
  - **Shorter than control words**
  - **Instruction decoder (hardwired logic) creates the control words from the instruction**

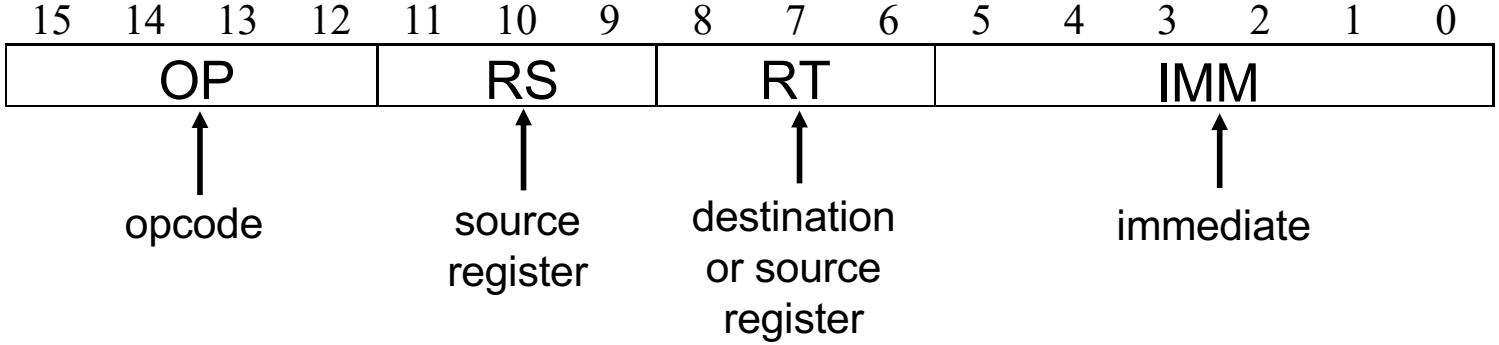


Our Instruction Formats

Register to Register (*R-Type*): ALU operations with 2 source registers



Immediate (*I-Type*): ALU operations with immediate, load/store, branch



ECE-2300 Instructions (A Subset)

Instruction	F	OP	FUNCT	DR	SA	SB	IMM	MB	FS	MD	LD	MW	BS	OFF
ADD rd,rs,rt	R	1111	000	RD	RS	RT	--	0	ADD	0	1	0	000	--
SUB rd,rs,rt	R	1111	001	RD	RS	RT	--	0	SUB	0	1	0	000	--
SRA rd,rs	R	1111	010	RD	RS	--	--	--	SRA	0	1	0	000	--
SRL rd,rs	R	1111	011	RD	RS	--	--	--	SRL	0	1	0	000	--
SLL rd,rs	R	1111	100	RD	RS	--	--	--	SLL	0	1	0	000	--
AND rd,rs,rt	R	1111	101	RD	RS	RT	--	0	AND	0	1	0	000	--
OR rd,rs,rt	R	1111	110	RD	RS	RT	--	0	OR	0	1	0	000	--
NOP	R	0000	000	--	--	--	--	--	--	-	0	0	000	--
HALT	R	0000	001	--	--	--	--	--	--	-	0	0	000	--
LB rt,*imm*(rs)	I	0010		RT	RS	--	*imm*	1	ADD	1	1	0	000	--
SB rt,*imm*(rs)	I	0100		--	RS	RT	*imm*	1	ADD	-	0	1	000	--
ADDI rt,rs,*imm*	I	0101		RT	RS	--	*imm*	1	ADD	0	1	0	000	--
ANDI rt,rs,*imm*	I	0110		RT	RS	--	*imm*	1	AND	0	1	0	000	--
ORI rt,rs,*imm*	I	0111		RT	RS	--	*imm*	1	OR	0	1	0	000	--
BEQ rt,rs,*target*	I	1000		--	RS	RT	--	0	SUB	-	0	0	010	*imm*
BNE rt,rs,*target*	I	1001		--	RS	RT	--	0	SUB	-	0	0	011	*imm*
BGEZ rs,*target*	I	1010		--	RS	--	0	1	SUB	-	0	0	101	*imm*
BLTZ rs,*target*	I	1011		--	RS	--	0	1	SUB	-	0	0	100	*imm*

Next Class

Instruction Set Architecture
Pipelined Microprocessor
(H&H 7.5.1-7.5.2)