ECE 2300
Digital Logic & Computer Organization

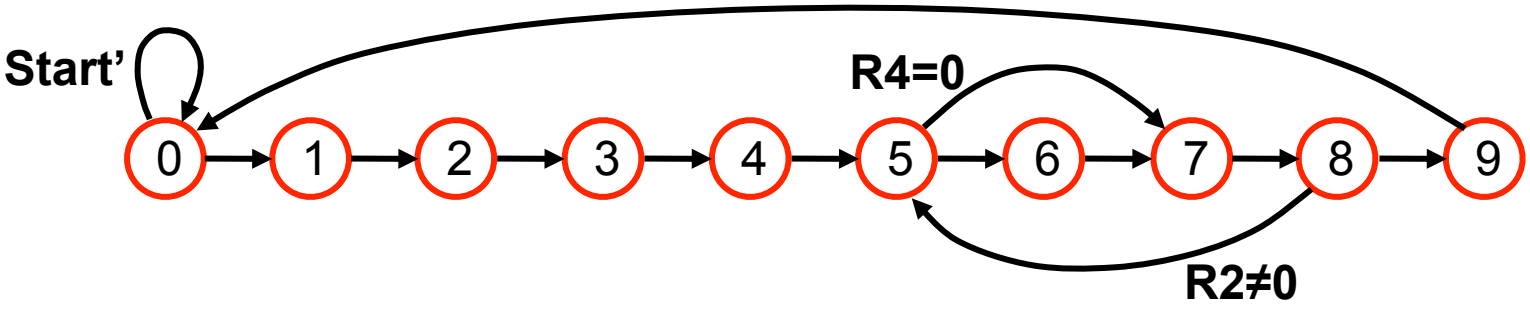
Fall 2016

More Single Cycle Microprocessor
Pipelined Microprocessor

Cornell University

Shift and Add 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	xxx	0	1	ADD	1	1	0
S3	R2 <= M[R0+1]	010	000	xxx	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	xxx	1	1	AND	0	1	0
S6	R3 <= R3 + R1	011	011	001	x	0	ADD	0	1	0
S7	R1 <= SLL(R1)	001	001	xxx	x	x	SLL	0	1	0
S8	R2 <= SRL(R2)	010	010	xxx	x	x	SRL	0	1	0
S9	M[R0+2] <= R3	xxx	000	011	2	1	ADD	x	0	1



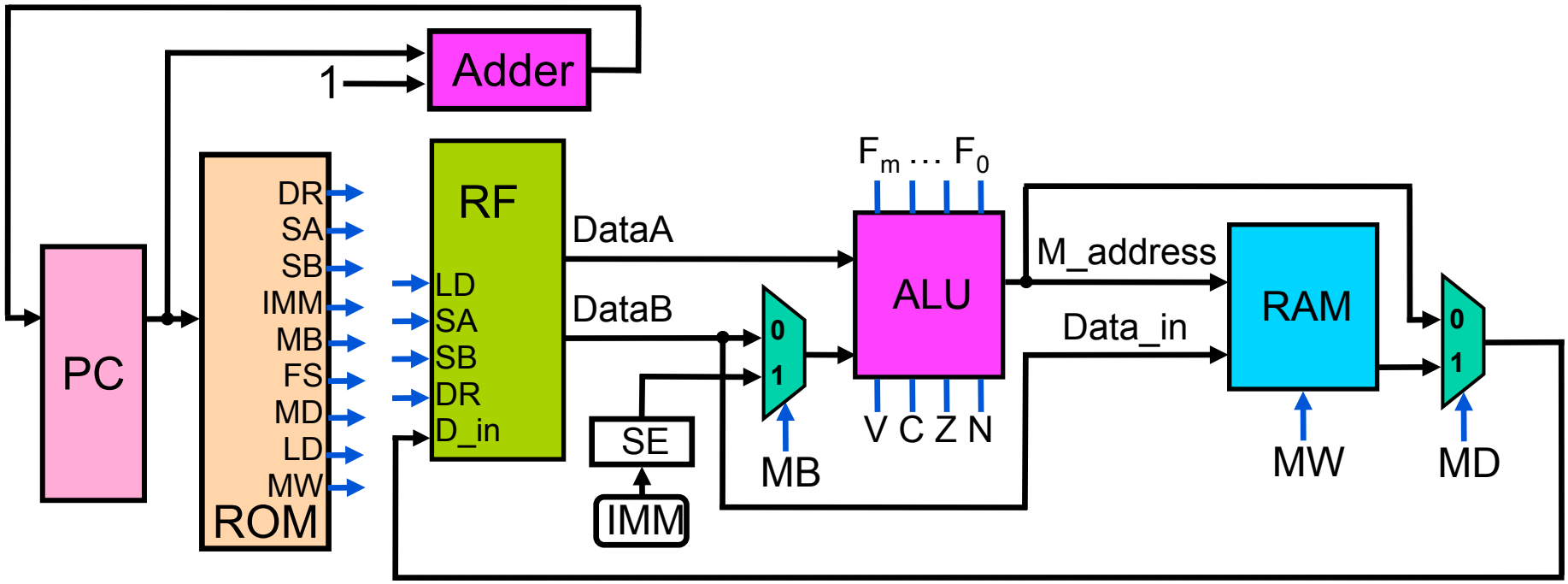
Programmable Microprocessor

- **Flexible datapath (RF, ALU, RAM, muxes)**
- **Flexible control via a programmable Control Unit**
  - **Control words stored in ROM**
  - **State machine replaced by a *Program Counter* plus *branch operations***
  - **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

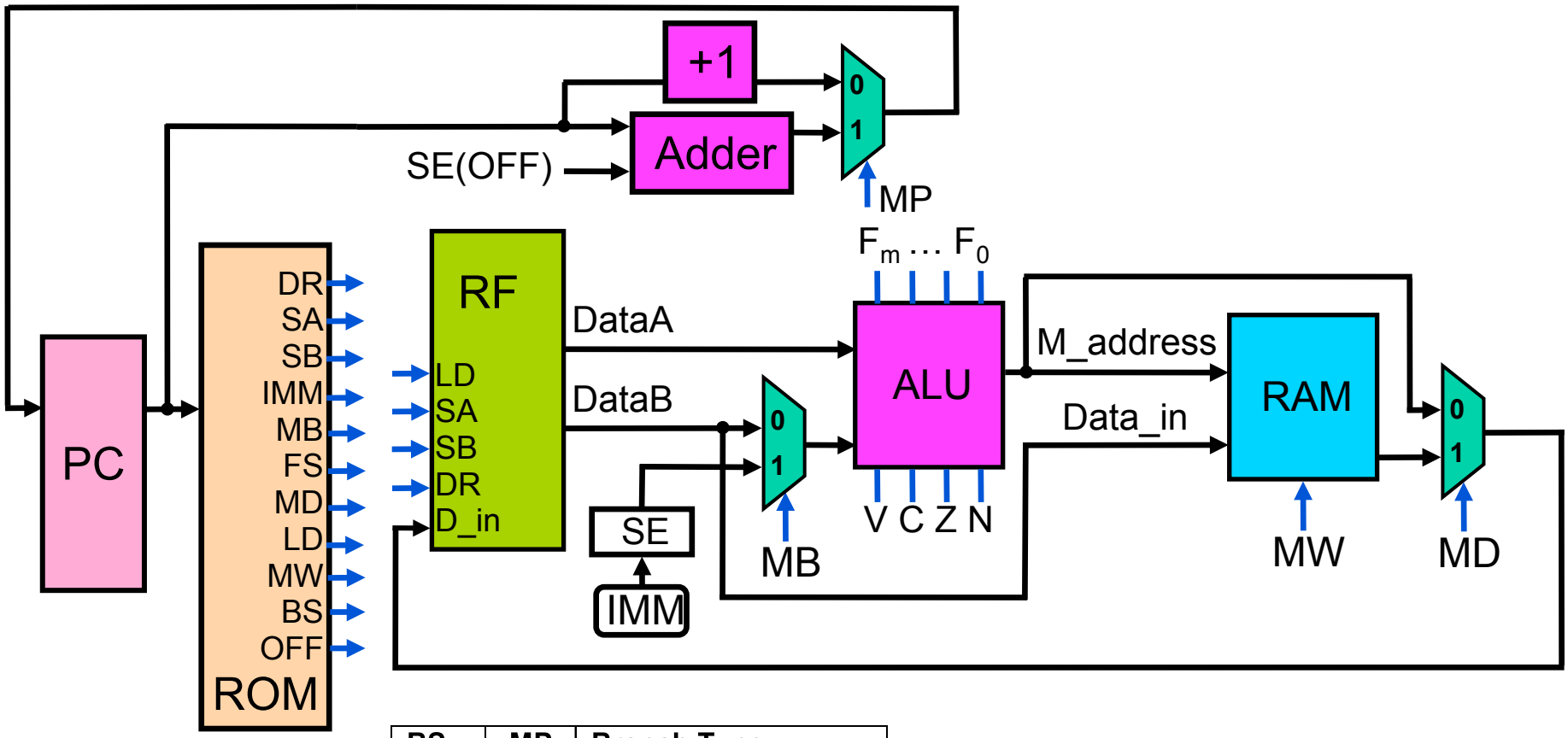


		DR	SA	SB	IMM	MB	FS	MD	LD	MW
R2 <= R0 + R1	a:	010	000	001	x	0	ADD	0	1	0
R1 <= M[R2]	a+1:	001	010	xxx	0	1	ADD	1	1	0
M[R2] <= R0	a+2:	xxx	010	000	0	1	ADD	x	0	1
R3 <= R0 + 3	a+3:	011	000	xxx	3	1	ADD	0	1	0

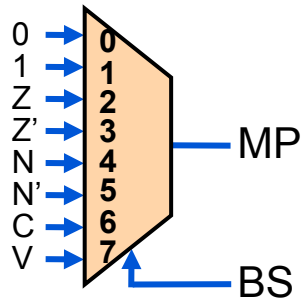
Branch Operations

- 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

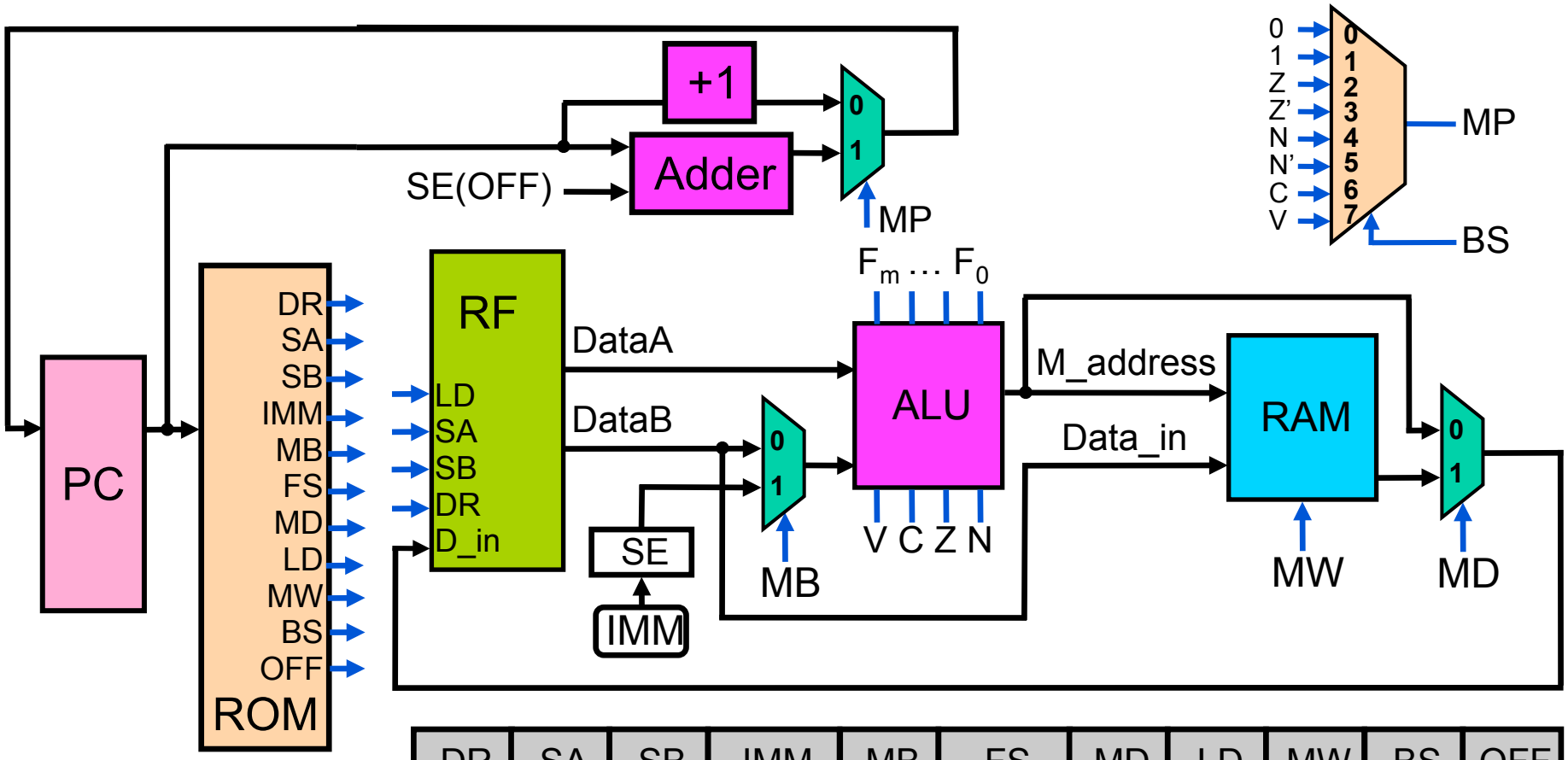
Branch Operations



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*

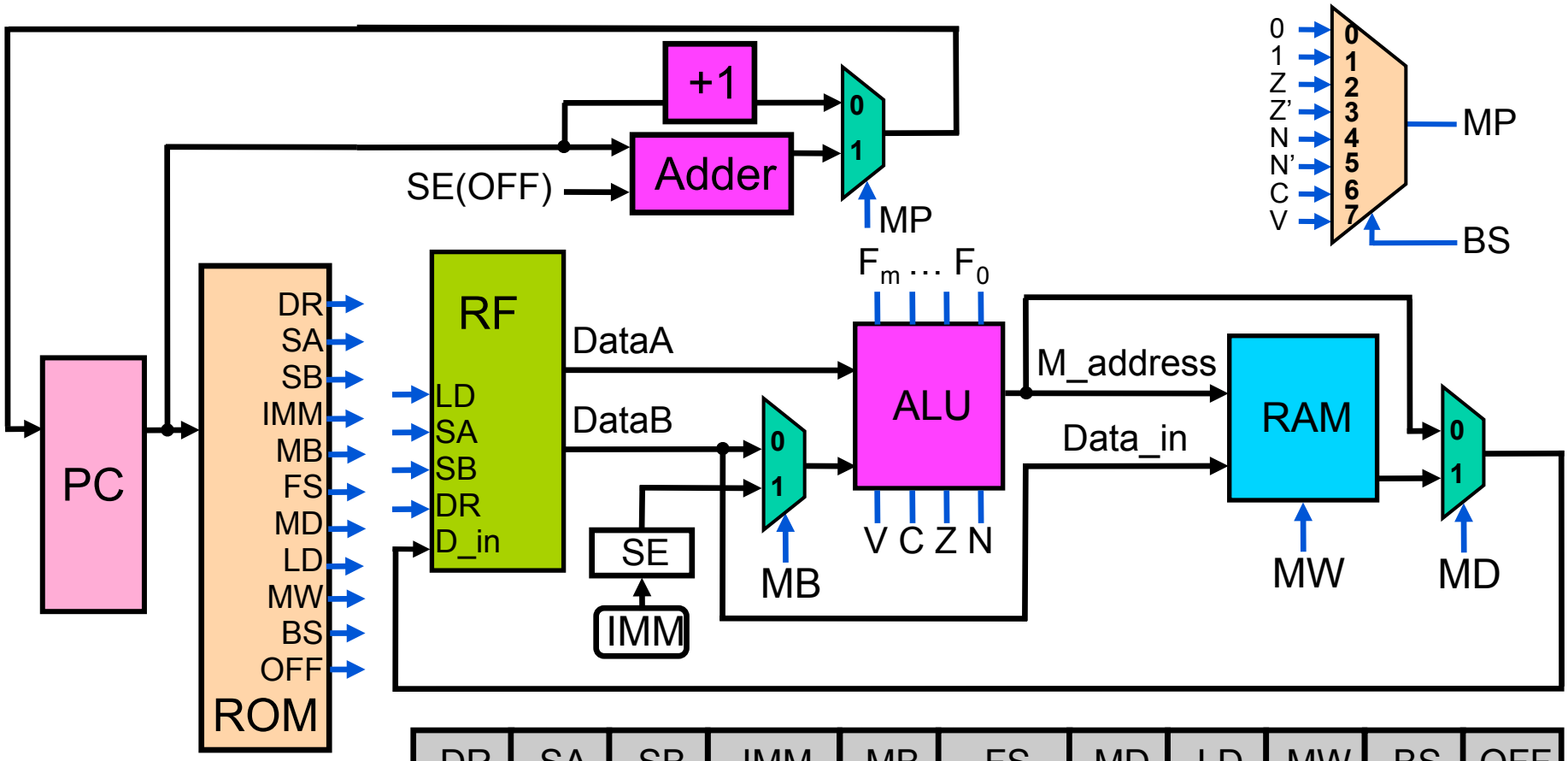


Branch Operations



	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



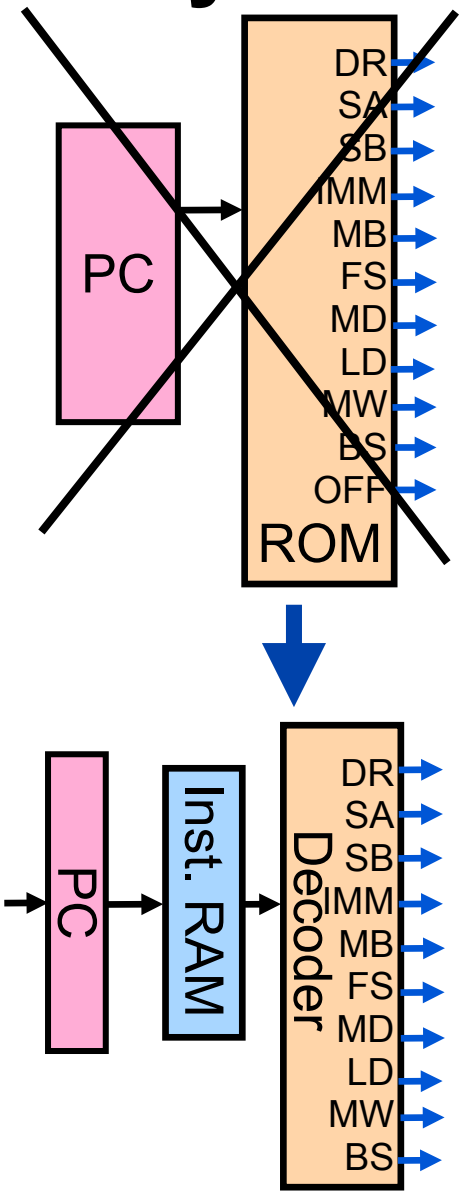
	DR	SA	SB	IMM	MB	FS	MD	LD	MW	BS	OFF
R2 <= SRL(R2) 8:	010	010	x	x	x	SRL	0	1	0	000	x
if (R2≠0) goto 4 9:	x	010	x	0	1	SUB	x	0	0	011	-5
M[R0+2] <= R3 10:	x	000	011	2	1	ADD	x	0	1	000	x

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

Providing Even More Flexibility

- Replace the ROM with a RAM into which the system loads the program
- Replace control words with *instructions*
  - More intuitive and not specific to particular hardware
  - Shorter than control words
  - *Instruction decoder* creates the control words from the instruction
- How do we know what instructions to run on the machine?

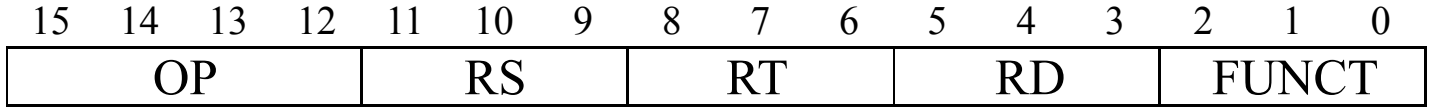


Instruction Set Architecture

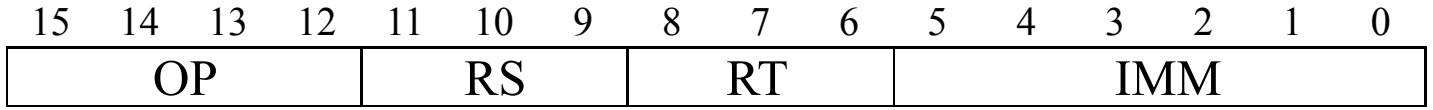
- The ISA describes a set of instructions supported by a family of machines
- Examples: x86, ARM, MIPS, POWER, SPARC
- The ISA specification tells hardware and software (compiler and operating system) developers
  - Instruction formats
  - Operation of each instruction
  - Ways to form memory addresses
  - Data formats
  - Lots of other stuff

Example Instruction Formats

Register to Register

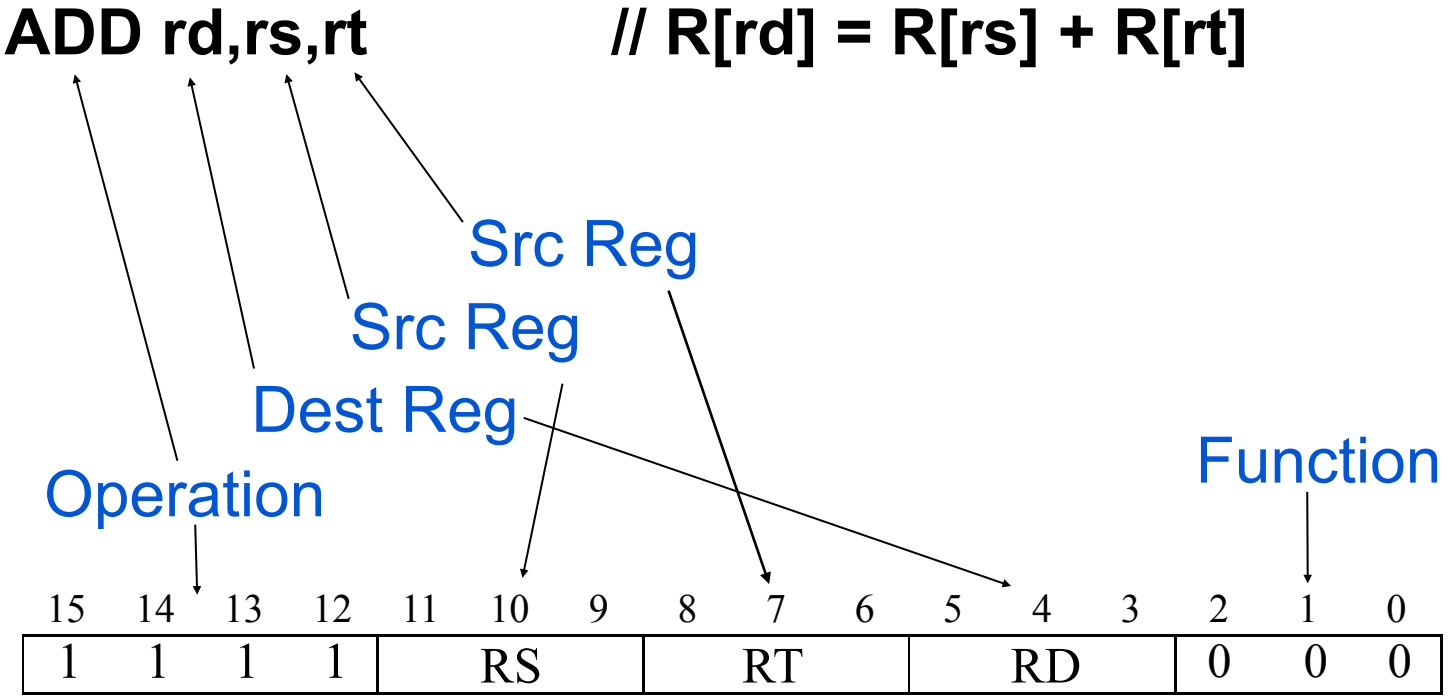


Immediate



Assembly Language

- **Human readable version of instructions**



Register to Register Format

Instruction	Format	OP	FUNCT	Operation
ADD rd,rs,rt	R	1111	000	R[rd] = R[rs] + R[rt]
SUB rd,rs,rt	R	1111	001	R[rd] = R[rs] - R[rt]
SRA rd, rs	R	1111	010	R[rd] = R[rs] $>>_A$ 1
SRL rd, rs	R	1111	011	R[rd] = R[rs] $>>_L$ 1
SLL rd,rs	R	1111	100	R[rd] = R[rs] $<<_L$ 1
AND rd,rs,rt	R	1111	101	R[rd] = R[rs] & R[rt]
OR rd,rs,rt	R	1111	110	R[rd] = R[rs] | R[rt]
unused	R	1111	111	
NOP	R	0000	000	Null Operation
HALT	R	0000	001	Halt the processor

- ALU operations with two source registers
- Shift operations
  - Logical: Shift in a 0
  - Arithmetic: Shift in the sign bit
- NOP and HALT

Immediate Format

Instruction	Format	OP	Operation
LW rt,*offset*(rs)	I	0001	R[rt] = M[R[rs]+sext(offset)]
LB rt,*offset*(rs)	I	0010	R[rt] = sext(M[R[rs]+sext(offset)])
SW rt,*offset*(rs)	I	0011	M[R[rs]+sext(offset)] = R[rt]
SB rt,*offset*(rs)	I	0100	M[R[rs]+sext(offset)] = R[rt](7:0)
ADDI rt,rs,*imm*	I	0101	R[rt] = R[rs] + sext(imm)
ANDI rt,rs,*imm*	I	0110	R[rt] = R[rs] & zext(imm)
ORI rt,rs,*imm*	I	0111	R[rt] = R[rs] | zext(imm)
BEQ rt,rs,*target*	I	1000	if(R[rs] == R[rt]) PC = PC + sext({imm,1'b0})
BNE rt,rs,*target*	I	1001	if(R[rs] ≠ R[rt]) PC = PC + sext({imm,1'b0})
BGEZ rs,*target*	I	1010	if(R[rs] >= 0) PC = PC + sext({imm,1'b0})
BLTZ rs,*target*	I	1011	if(R[rs] < 0) PC = PC + sext({imm,1'b0})

- ALU instructions with an immediate operand
- Load and Store instructions
  - LB, SB: Load or store a byte (8 bits)
  - LW, SW: Load or store a word (16 bits)
- Branch instructions

Memory Addresses

- The smallest addressable quantity in memory is a byte
- A memory address points to a particular byte location
- LB = “give me the byte at this address”
- LW = “give me the byte at this address and the next one too”

Branch Instructions

Instruction	Format	OP	Operation
BEQ rt,rs,*target*	I	1000	if(R[rs] == R[rt]) PC = PC + sext({imm,1'b0})
BNE rt,rs,*target*	I	1001	if(R[rs] ≠ R[rt]) PC = PC + sext({imm,1'b0})
BGEZ rs,*target*	I	1010	if(R[rs] >= 0) PC = PC + sext({imm,1'b0})
BLTZ rs,*target*	I	1011	if(R[rs] < 0) PC = PC + sext({imm,1'b0})

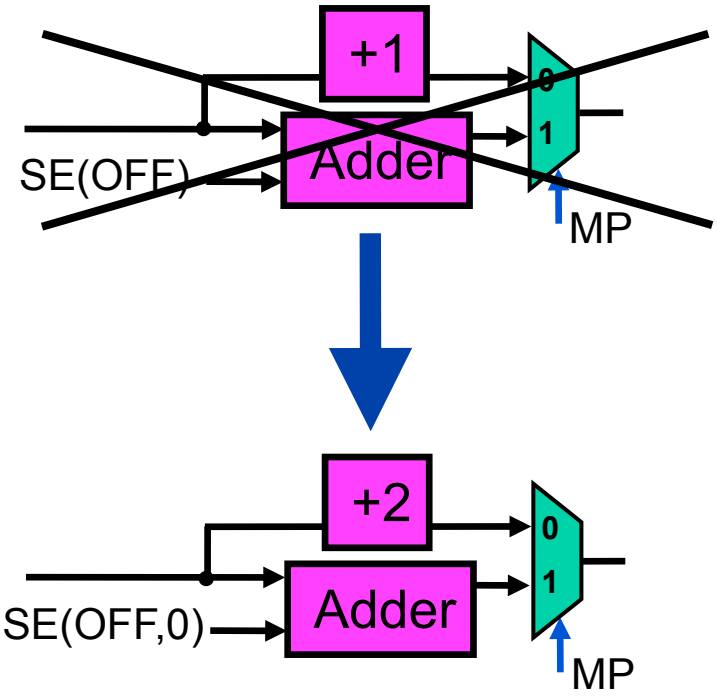
- **Condition**
  - Test two registers for equality (BEQ, BNE)
  - Test if a register is positive or negative (BGEZ, BLTZ)

- **Target**
  - Where to jump in the program if the condition is true
  - Formed by adding the offset to the PC

Incrementing the PC, Forming the Target

Instruction	Format	OP	Operation
BEQ rt,rs,*target*	I	1000	if(R[rs] == R[rt]) PC = PC + sext({imm,1'b0})
BNE rt,rs,*target*	I	1001	if(R[rs] ≠ R[rt]) PC = PC + sext({imm,1'b0})
BGEZ rs,*target*	I	1010	if(R[rs] >= 0) PC = PC + sext({imm,1'b0})
BLTZ rs,*target*	I	1011	if(R[rs] < 0) PC = PC + sext({imm,1'b0})

- A memory address points to a byte location



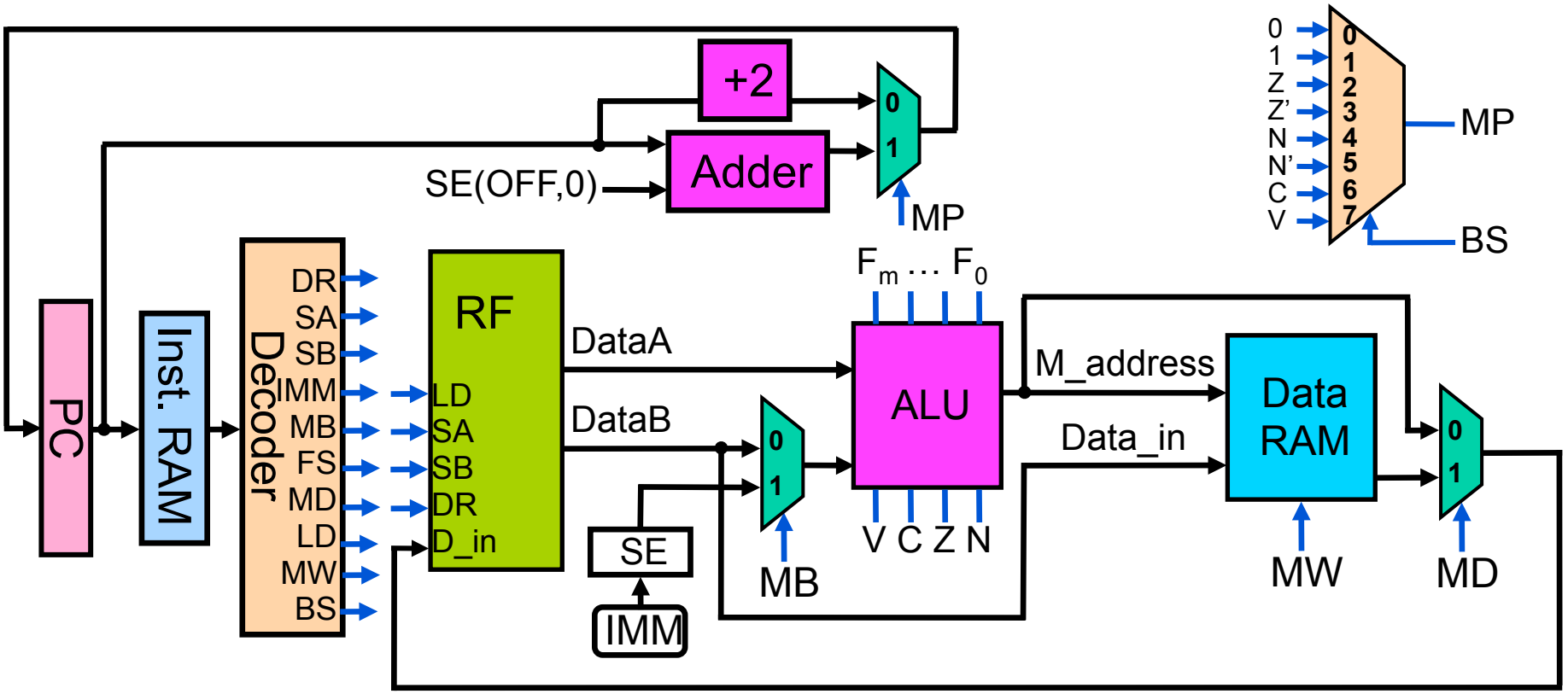
- If instructions are 2 bytes wide
  - Instructions are located at even locations (0, 2, 4, ...)
  - The PC must be incremented by 2 (PC+2)
  - Since the offset refers to the number of *instructions* (not bytes) to jump, we must append a 0 to it

Instructions → Control Words

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*

- *Instruction decoder* derives the control word from the instruction fields
- Combinational circuit → instruction input, CW output

Programmable Processor



Multiplication Revisited

0: SUB R0,R0,R0
1: LB R1,0(R0)
2: LB R2,1(R0)
3: SUB R3,R3,R3
4: ANDI R4,R2,1
5: BEQ R4,R0,7
6: ADD R3,R3,R1
7: SLL R1,R1
8: SRL R2,R2
9: BNE R2,R0,4
10: SW R3,2(R0)

OP	RS	RT	RD	FUNCT
1111	000	000	000	001
0010	000	001	000000	
0010	000	010	000001	
1111	011	011	011	001
0110	010	100	000001	
1000	000	100	000010	
1111	011	001	011	000
1111	001	xxx	001	100
1111	010	xxx	010	011
1001	000	010	111011	
0011	000	010	000010	
OP	RS	RT	IMM	

Jump Instructions

- "GoTo" instructions
- Jump unconditionally to the target
- Target is formed and loaded into the PC register
- Jump
  - Target formed by adding offset to the PC
- Jump Register
  - Register contains the target address

Single Cycle Microprocessor Performance

- Hardware steps: instruction fetch, register read, ALU, memory, register write
- Assume each step takes 1ns
- Different instructions use different steps

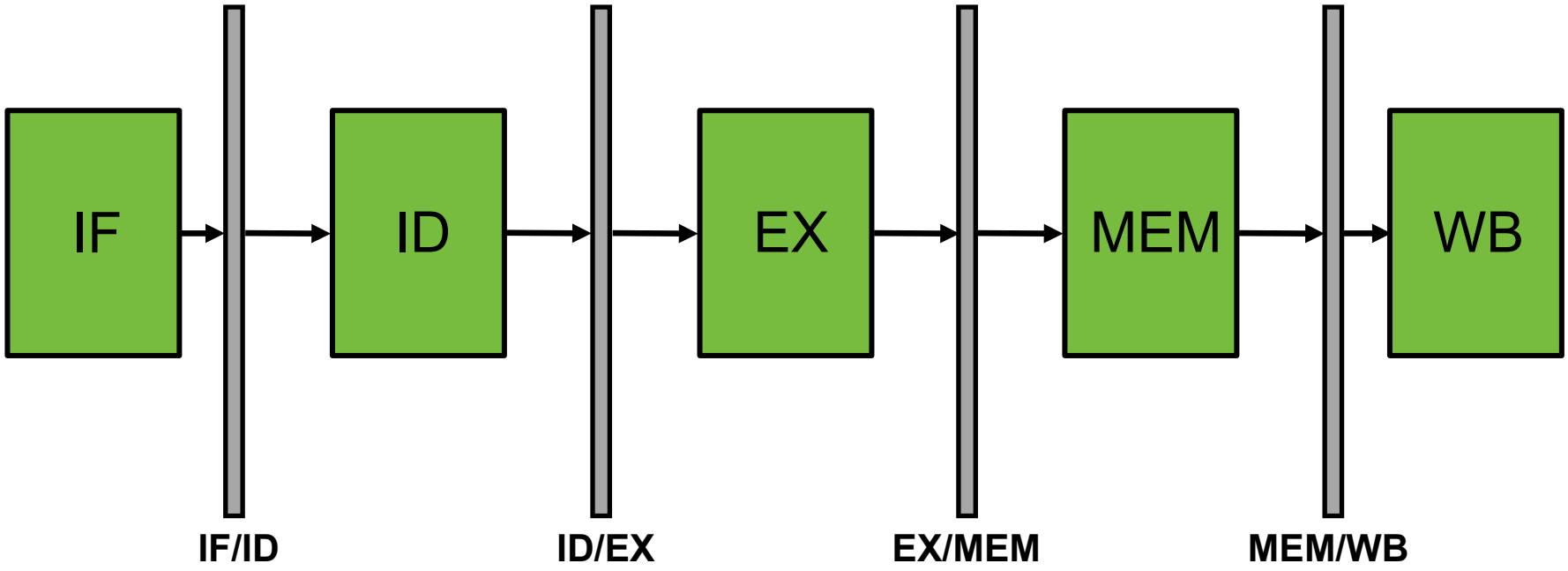
Instruction Type	IF	RR	ALU	MEM	RW	Total
Register to Register	1ns	1ns	1ns	-	1ns	4ns
Load from Memory	1ns	1ns	1ns	1ns	1ns	5ns
Store to Memory	1ns	1ns	1ns	1ns	-	4ns
Jump	1ns	1ns	-	-	-	2ns

- Clock period is limited by longest instruction (5ns)
- Pipelining: Overlap instruction execution by performing each step in successive clock cycles

5 Steps in Instruction Execution

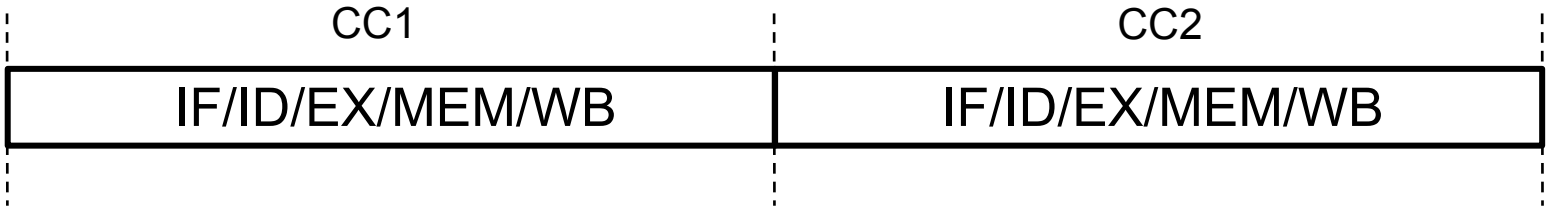
- Instruction Fetch (IF)
  - Update PC; Fetch instruction
- Instruction Decode (ID)
  - Decode instruction; Read register file
- Execute (EX)
  - Perform ALU operation
- Memory (MEM)
  - Perform memory operation
- Write Back (WB)
  - Store result into register file

Pipelining: Basic Idea

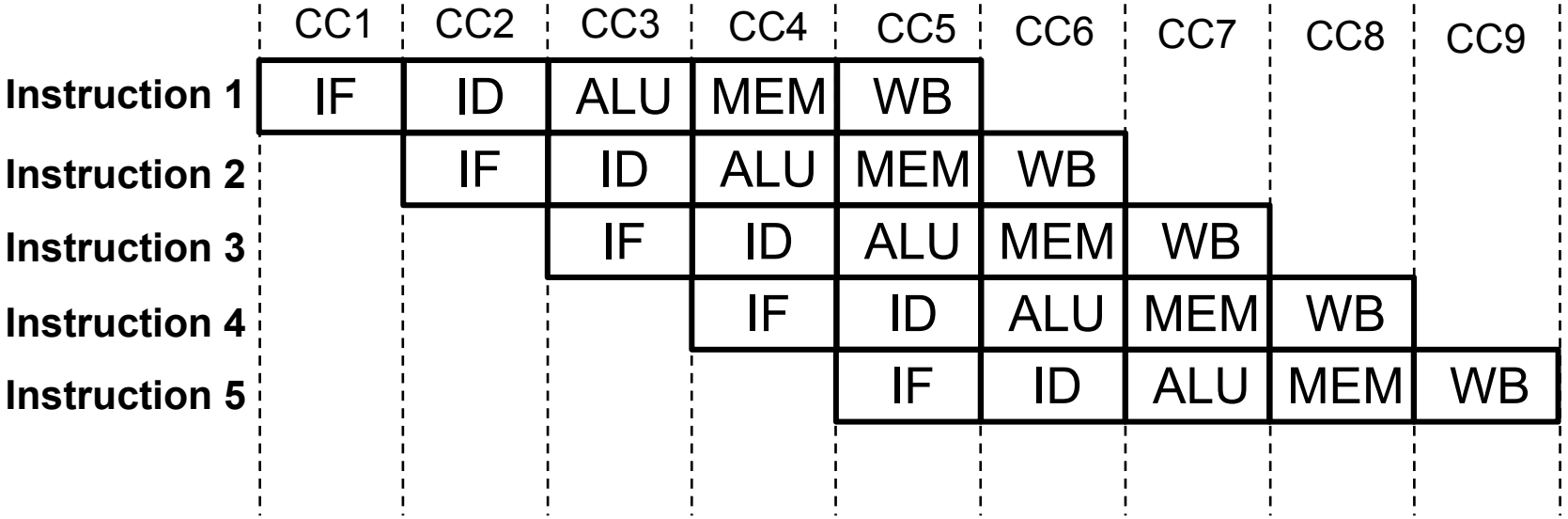


Pipelining: Overlapped Instructions

- **Single-cycle execution**



- **Pipelined execution**

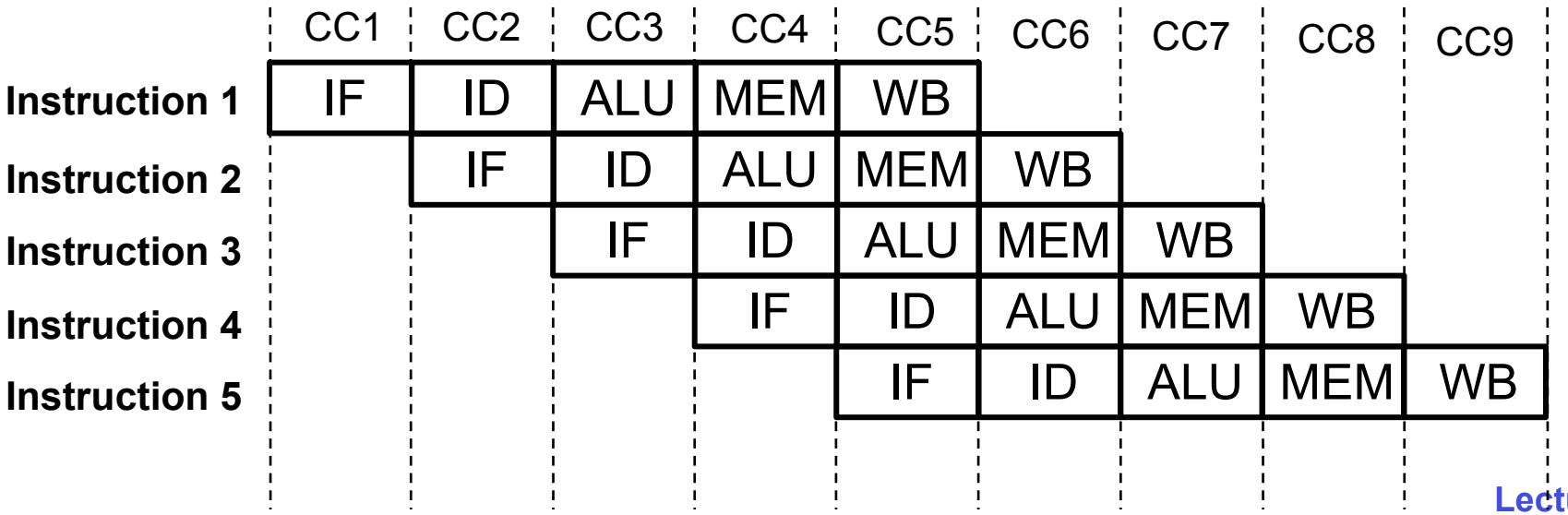


Pipelining: Performance

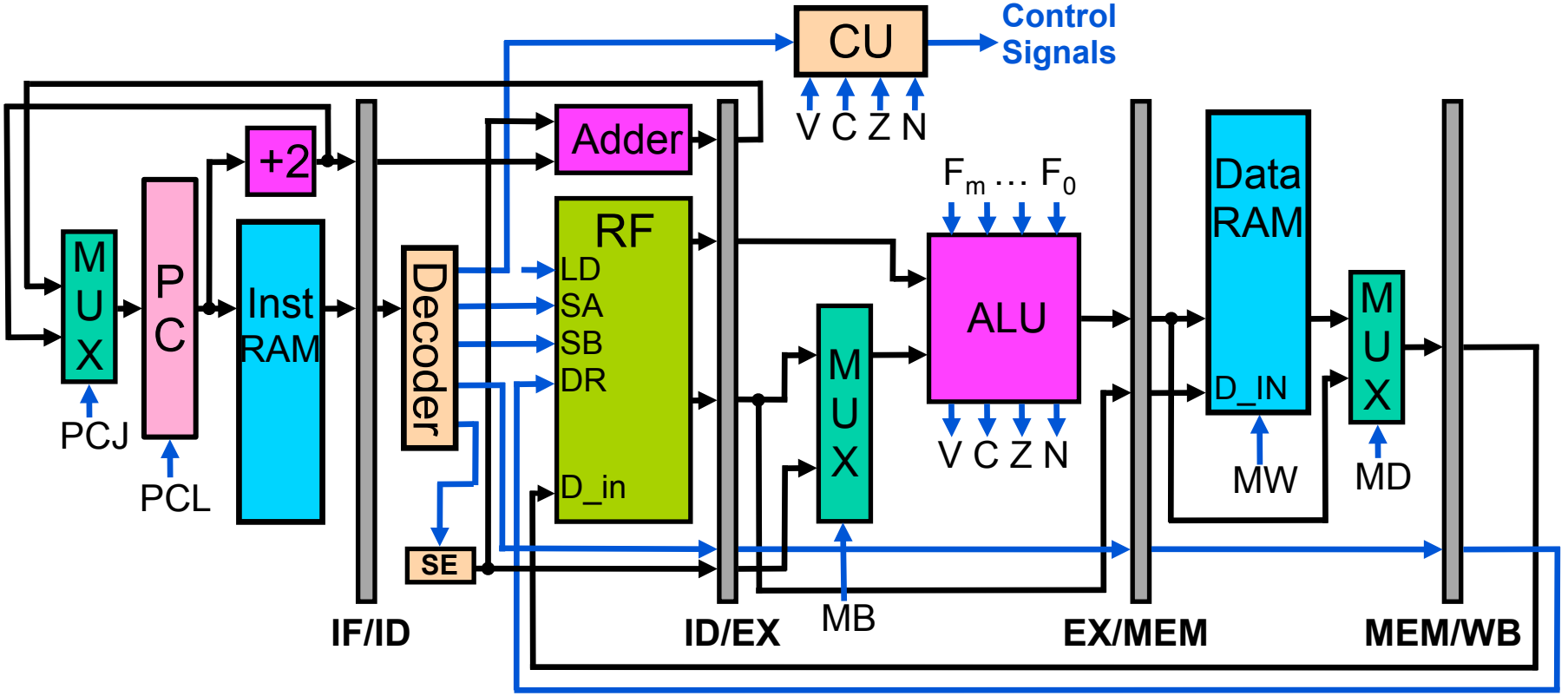
- **Faster clock frequency than single cycle processor**
- **Each instruction takes 5 cycles**
- **Average number of cycles per instruction (CPI)**

$$\frac{\text{Number of cycles for } N \text{ instructions}}{N} = \frac{N+4}{N} \approx 1$$

- **~1 instruction completed every cycle (ideally)**



Pipelined Processor



Next Time

More Pipelined Microprocessor