Logic and Computer Design Fundamentals

Chapter 9 – Computer Design Basics

Part 2 – A Simple Computer

Charles Kime & Thomas Kaminski

(Hyperlinks are active in View Show mode)

Overview

- Part 1 – Datapaths
- Part 2 – A Simple Computer
  - Instruction Set Architecture (ISA)
  - Single-Cycle Hardwired Control
    - PC Function
    - Instruction Decoder
    - Example Instruction Execution
- Part 3 – Multiple Cycle Hardwired Control

Instruction Set Architecture (ISA) for Simple Computer (SC)

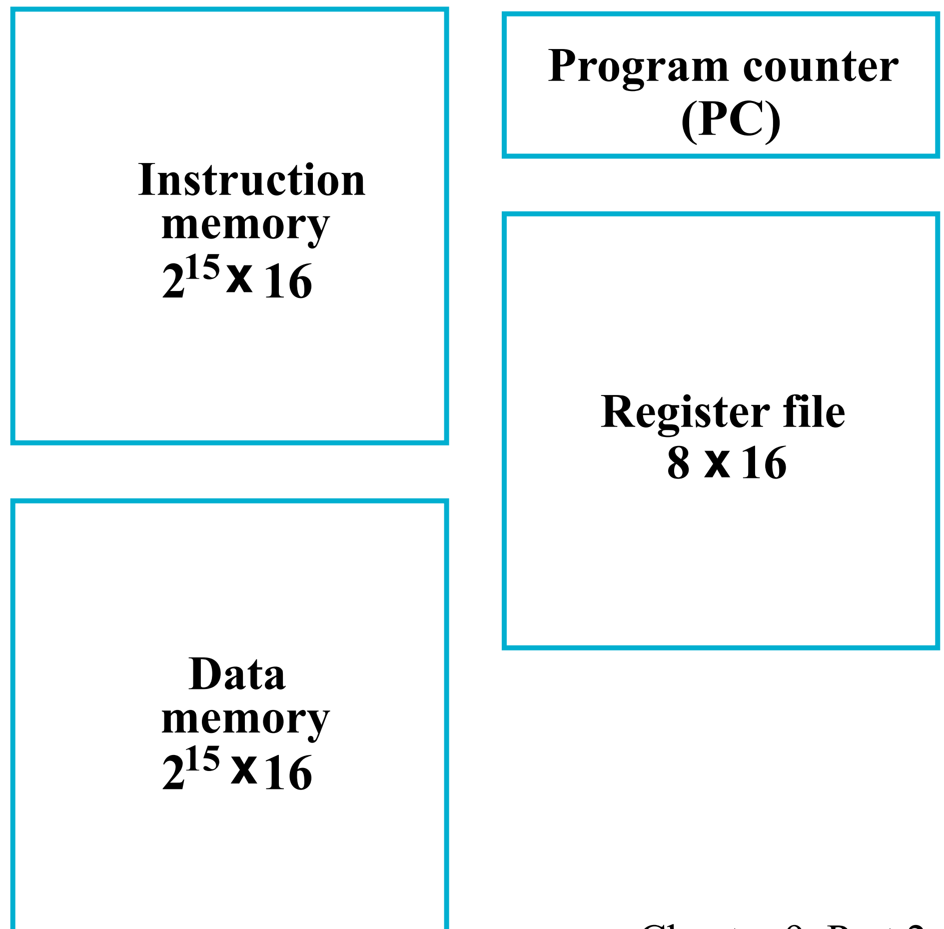
- A programmable system uses a sequence of *instructions* to control its operation
- An typical instruction specifies:
  - Operation to be performed
  - Operands to use, and
  - Where to place the result, or
  - Which instruction to execute next
- Instructions are stored in RAM or ROM as a *program*
- The addresses for instructions in a computer are provided by a *program counter (PC)* that can
  - Count up
  - Load a new address based on an instruction and, optionally, status information

Instruction Set Architecture (ISA) (continued)

- **The PC and associated control logic are part of the Control Unit**
- **Executing an instruction - activating the necessary sequence of operations specified by the instruction**
- **Execution is controlled by the control unit and performed:**
  - **In the datapath**
  - **In the control unit**
  - **In external hardware such as memory or input/ output**

ISA: Storage Resources

- The storage resources are "visible" to the programmer at the lowest software level (typically, machine or assembly language)
- Storage resources for the SC =>
- Separate instruction and data memories imply "Harvard architecture"
- Done to permit use of single clock cycle per instruction implementation
- Due to use of "cache" in modern computer architectures, is a fairly realistic model



ISA: Instruction Format

- **A instruction consists of a bit vector**
- **The *fields* of an instruction are subvectors representing specific functions and having specific binary codes defined**
- **The *format* of an instruction defines the subvectors and their function**
- **An ISA usually contains multiple formats**
- **The SC ISA contains the three formats presented on the next slide**

ISA: Instruction Format

15 9	8 6	5 3	2 0
Opcode	Destination register (DR)	Source register A (SA)	Source register B (SB)

(a) Register

15 9	8 6	5 3	2 0
Opcode	Destination register (DR)	Source register A (SA)	Operand (OP)

(b) Immediate

15 9	8 6	5 3	2 0
Opcode	Address (AD) (Left)	Source register A (SA)	Address (AD) (Right)

(c) Jump and Branch

- The three formats are: Register, Immediate, and Jump and Branch
- All formats contain an Opcode field in bits 9 through 15.
- The Opcode specifies the operation to be performed
- More details on each format are provided on the next three slides

ISA: Instruction Format (continued)

15 — 9	8 — 6	5 — 3	2 — 0
Opcode	Destination register (DR)	Source register A (SA)	Source register B (SB)

(a) Register

- This format supports instructions represented by:
  - R1 ← R2 + R3
  - R1 ← sl R2
- There are three 3-bit register fields:
  - DR - specifies destination register (R1 in the examples)
  - SA - specifies the A source register (R2 in the first example)
  - SB - specifies the B source register (R3 in the first example and R2 in the second example)
- Why is R2 in the second example SB instead of SA?
  - The source for the shifter in our datapath to be used in

ISA: Instruction Format (continued)

15 — 9	8 — 6	5 — 3	2 — 0
Opcode	Destination register (DR)	Source register A (SA)	Operand (OP)

(b) Immediate

- **This format supports instructions described by:**
  - **R1 ← R2 + 3**
- **The B Source Register field is replaced by an Operand field OP which specifies a constant.**
- **The Operand:**
  - **3-bit constant**
  - **Values from 0 to 7**
- **The constant:**
  - **Zero-fill (on the left of) the Operand to form 16-bit constant**
  - **16-bit representation for values 0 through 7**

ISA: Instruction Format (continued)

15 … 9	8 … 6	5 … 3	2 … 0
Opcode	Address (AD) (Left)	Source register A (SA)	Address (AD) (Right)

(c) Jump and Branch

- This instruction supports changes in the sequence of instruction execution by adding an extended, 6-bit, signed 2s-complement *address offset* to the PC value
- The 6-bit Address (AD) field replaces the DR and SB fields
  - Example: Suppose that a jump is specified by the Opcode and the PC contains 45 (0…0101101) and Address contains – 12 (110100). Then the new PC value will be:
    0…0101101 + (1…110100) = 0…0100001 (45 + (– 12) = 33)
- The SA field is retained to permit jumps and branches on N or Z based on the contents of Source register A

ISA: Instruction Specifications

- **The specifications provide:**
  - **The name of the instruction**
  - **The instruction's opcode**
  - **A shorthand name for the opcode called a *mnemonic***
  - **A specification for the instruction format**
  - **A register transfer description of the instruction, and**
  - **A listing of the status bits that are meaningful during an instruction's execution (not used in the architectures defined in this chapter)**

ISA: Instruction Specifications (continued)

❑ **TABLE 9-8**
Instruction Specifications for the Simple Computer

Instruction	Opcode	Mne-monic	Format	Description	Status Bits
Move A	0000000	MOVA	RD, RA	$R[DR] \leftarrow R[SA]$*	N, Z
Increment	0000001	INC	RD, RA	$R[DR] \leftarrow R[SA] + 1$*	N, Z
Add	0000010	ADD	RD, RA, RB	$R[DR] \leftarrow R[SA] + R[SB]$*	N, Z
Subtract	0000101	SUB	RD, RA, RB	$R[DR] \leftarrow R[SA] - R[SB]$*	N, Z
Decrement	0000110	DEC	RD, RA	$R[DR] \leftarrow R[SA] - 1$*	N, Z
AND	0001000	AND	RD, RA, RB	$R[DR] \leftarrow R[SA] \wedge R[SB]$*	N, Z
OR	0001001	OR	RD, RA, RB	$R[DR] \leftarrow R[SA] \vee R[SB]$*	N, Z
Exclusive OR	0001010	XOR	RD, RA, RB	$R[DR] \leftarrow R[SA] \oplus R[SB]$*	N, Z
NOT	0001011	NOT	RD, RA	$R[DR] \leftarrow \overline{R[SA]}$*	N, Z
Move B	0001100	MOVB	RD, RB	$R[DR] \leftarrow R[SB]$*	
Shift Right	0001101	SHR	RD, RB	$R[DR] \leftarrow$ sr $R[SB]$*	
Shift Left	0001110	SHL	RD, RB	$R[DR] \leftarrow$ sl $R[SB]$*	
Load Immediate	1001100	LDI	RD, OP	$R[DR] \leftarrow$ zf OP*	
Add Immediate	1000010	ADI	RD, RA, OP	$R[DR] \leftarrow R[SA] +$ zf OP*	N, Z
Load	0010000	LD	RD, RA	$R[DR] \leftarrow M[SA]$*	
Store	0100000	ST	RA, RB	$M[SA] \leftarrow R[SB]$*	
Branch on Zero	1100000	BRZ	RA, AD	if $(R[SA] = 0)$ $PC \leftarrow PC +$ se AD, if $(R[SA] \neq 0)$ $PC \leftarrow PC + 1$	N, Z
Branch on Negative	1100001	BRN	RA, AD	if $(R[SA] < 0)$ $PC \leftarrow PC +$ se AD, if $(R[SA] \geq 0)$ $PC \leftarrow PC + 1$	N, Z
Jump	1110000	JMP	RA	$PC \leftarrow R[SA]$	

* For all of these instructions, $PC \leftarrow PC + 1$ is also executed to prepare for the next cycle.

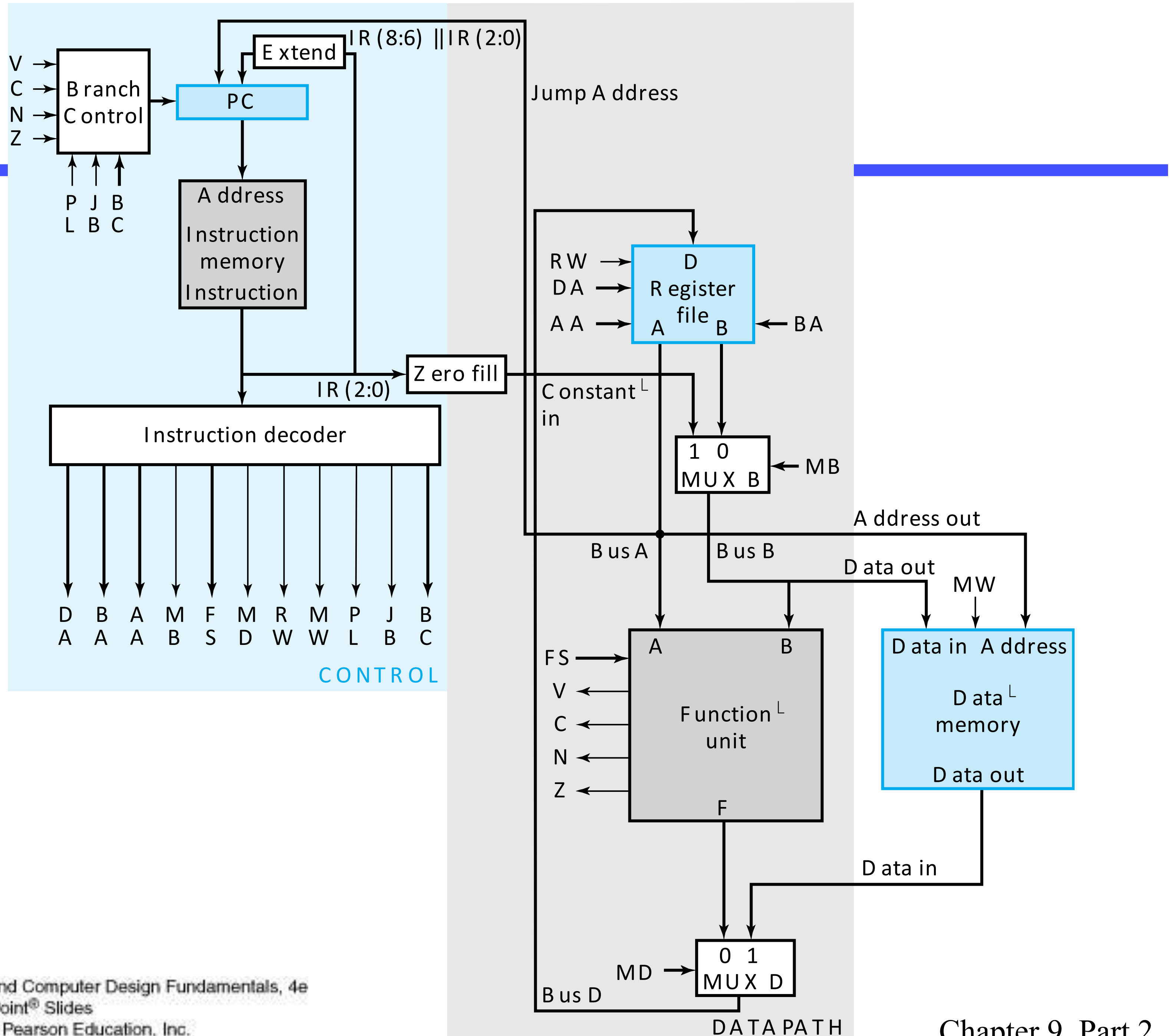
ISA:Example Instructions and Data in Memory

Memory Representation of Instructions and Data

Deciimal Address	Memory Contents	Decimal Opcode	Other Fields	Operation
25	0000101 001010011	5 (Subtract)	DR:1, SA:2, SB:3	R1 ← R2 − R3
35	0100000 000100101	32 (Store)	SA:4, SB:5	M[R4] ← R5
45	1000010 010111011	66 (Add Immediate)	DR:2, SA:7, OP:3	R2 ← R7 + 3
55	1100000 101110100	96 (Branch on Zero)	AD: 44, SA:6	If R6 = 0, PC ← PC − 20
70	0000000001100000	Data = 192. After execution of instruction in 35, Data = 80.		

Single-Cycle Hardwired Control

- Based on the ISA defined, design a computer architecture to support the ISA
- The architecture is to fetch and execute each instruction in a single clock cycle
- The datapath from Figure 10-11 will be used
- The control unit will be defined as a part of the design
- The block diagram is shown on the next slide


IR (8:6) || IR (2:0)
Extend
V
C
N
Z
Branch Control
PC
PL
JB
BC
Jump Address
Address
Instruction memory
Instruction
RW
DA
AA
D
Register file
A
B
BA
Zero fill
IR (2:0)
Constant in
1 0
MUX B
MB
Instruction decoder
Address out
Bus A
Bus B
Data out
MW
DA
BA
AA
MB
FS
MD
RW
MW
PL
JB
BC
CONTROL
FS
V
C
N
Z
A
B
Function unit
F
Data in
Address
Data memory
Data out
Data in
0 1
MUX D
MD
Bus D
DATA PATH

The Control Unit

- The Data Memory has been attached to the Address Out and Data Out and Data In lines of the Datapath.
- The MW input to the Data Memory is the Memory Write signal from the Control Unit.
- For convenience, the Instruction Memory, which is not usually a part of the Control Unit is shown within it.
- The Instruction Memory address input is provided by the PC and its instruction output feeds the Instruction Decoder.
- Zero-filled IR(2:0) becomes Constant In
- Extended IR(8:6) || IR(2:0) and Bus A are address inputs to the PC.
- The PC is controlled by Branch Control logic

PC Function

- PC function is based on instruction specifications involving jumps and branches taken from Slide 13:

Branch on Zero	BRZ	if $(R[SA] = 0)$ $PC \leftarrow PC + se\,AD$
Branch on Negative	BRN	if $(R[SA] < 0)$ $PC \leftarrow PC + se\,AD$
Jump	JMP	$PC \leftarrow R[SA]$

- In addition to the above register transfers, the PC must also implement: $PC \leftarrow PC + 1$
- The first two transfers above require addition to the PC of: Address Offset = Extended IR(8:6) || IR(2:0)
- The third transfer requires that the PC be loaded with: Jump Address = Bus A = R[SA]
- The counting function of the PC requires addition to the PC of 1

PC Function (continued)

- **Branch Control determines the PC transfers based on five of its inputs defined as follows:**
  - **N,Z – negative and zero status bits**
  - **PL – load enable for the PC**
  - **JB – Jump/Branch select: If JB = 1, Jump, else Branch**
  - **BC – Branch Condition select: If BC = 1, branch for N = 1, else branch for Z = 1.**
- **The above is summarize by the following table:**

PC Operation	PL	JB	BC
Count Up	0	X	X
Jump	1	1	X
Branch on Negative (else Count Up)	1	0	1
Branch on Zero (else Count Up)	1	0	0

- **Sufficient information is provided here to design the PC**

Instruction Decoder

- The combinational instruction decoder converts the instruction into the signals necessary to control all parts of the computer during the single cycle execution
- The input is the 16-bit Instruction
- The outputs are control signals:
  - Register file addresses DA, AA, and BA,
  - Function Unit Select FS
  - Multiplexer Select Controls MB and MD,
  - Register file and Data Memory Write Controls RW and MW, and
  - PC Controls PL, JB, and BC
- The register file outputs are simply pass-through signals:

  DA = DR, AA = SA, and BA = SB

  Determination of the remaining signals is more complex.

Instruction Decoder (continued)

- **The remaining control signals do not depend on the addresses, so must be a function of IR(13:9)**
- **Formulation requires examining relationships between the outputs and the opcodes given in Slides 12 and 13.**
- **Observe that for other than branches and jumps, FS = IR(12:9)**
- **This implies that the other control signals should depend as much as possible on IR(15:13) (which actually were assigned with decoding in mind!)**
- **To make some sense of this, we divide instructions into types as shown in the table on the next page**

Instruction Decoder (continued)

Truth Table for Instruction Decoder Logic

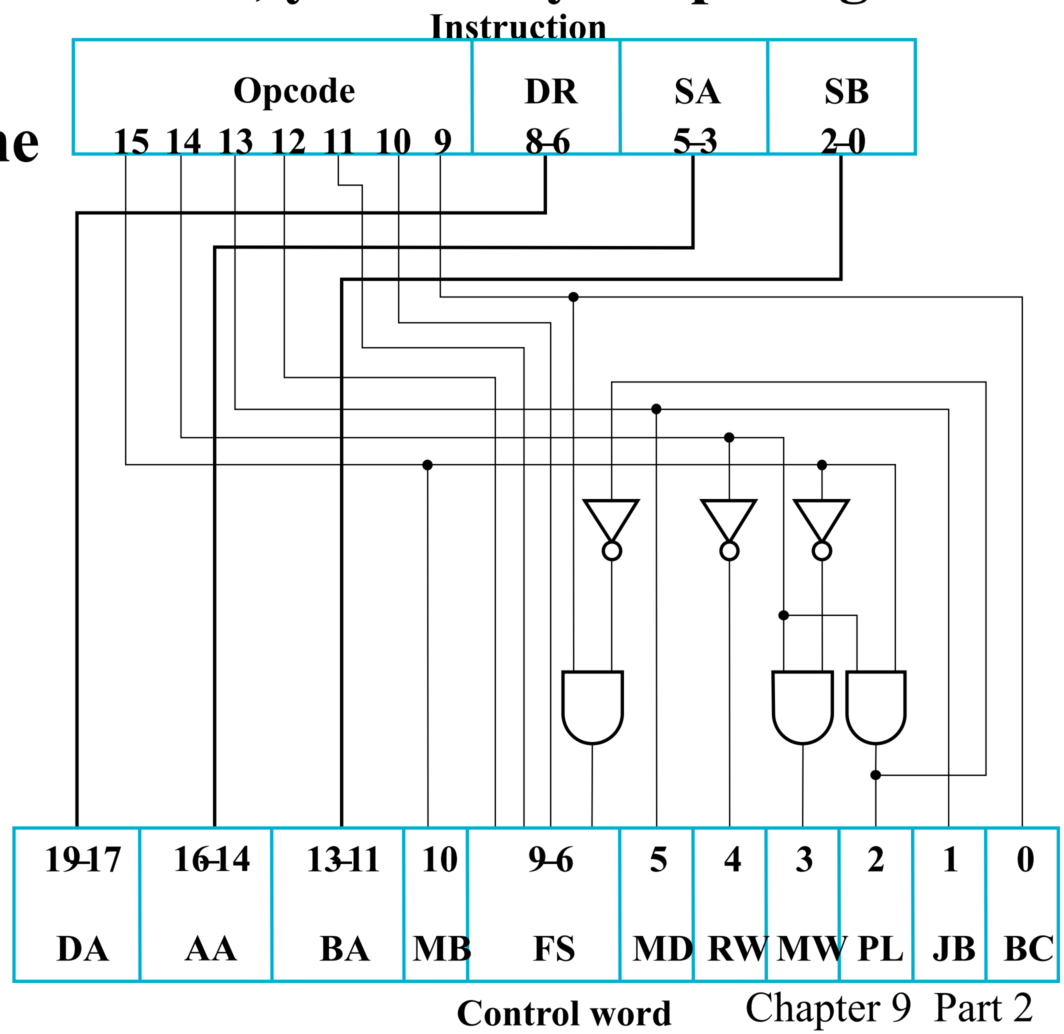
	Instruction Bits				Control Word Bits						
Instruction Function Type	**15**	**14**	**13**	**9**	**MB**	**MD**	**RW**	**MW**	**PL**	**JB**	**BC**
Function unit operations using registers	0	0	0	X	0	0	1	0	0	X	X
Memory read	0	0	1	X	0	1	1	0	0	X	X
Memory write	0	1	0	X	0	X	0	1	0	X	X
Function unit operations using register and constant	1	0	0	X	1	0	1	0	0	X	X
Conditional branch on zero (Z)	1	1	0	0	X	X	0	0	1	0	0
Conditional branch on negative (N)	1	1	0	1	X	X	0	0	1	0	1
Unconditional Jump	1	1	1	X	X	X	0	0	1	1	X

Instruction Decoder (continued)

- The types are based on the blocks controlled and the seven signals to be generated; types can be divided into two groups:
  - Datapath and Memory Control (First 4 types)
  - PC Control (Last 3 types)
- In Datapath and Memory Control blocks controlled are considered:
  - Mux B (1st and 4th types)
  - Memory and Mux D (2nd and 3rd types)
  - By assigning codes with no or only one 1 for these, implementation of MB, MD, RW and MW are simplified.
- In Control Unit more of a bit setting approach was used:
  - Bit 15 = Bit 14 = 1 were assigned to generate PL
  - Bit 13 values were assigned to generate JB.
  - Bit 9 was use as BC which contradicts FS = 0000 needed for branches. To force FS(6) to 0 for branches, Bit 9 into FS(6) is disabled by PL.
- Also, useful bit correlations between values in the two groups were exploited in assigning the codes.

Instruction Decoder (continued)

- The end result by use of the types, careful assignment of codes, and use of don't cares, yields very simple logic:
- This completes the design of most of the essential parts of the single-cycle simple computer



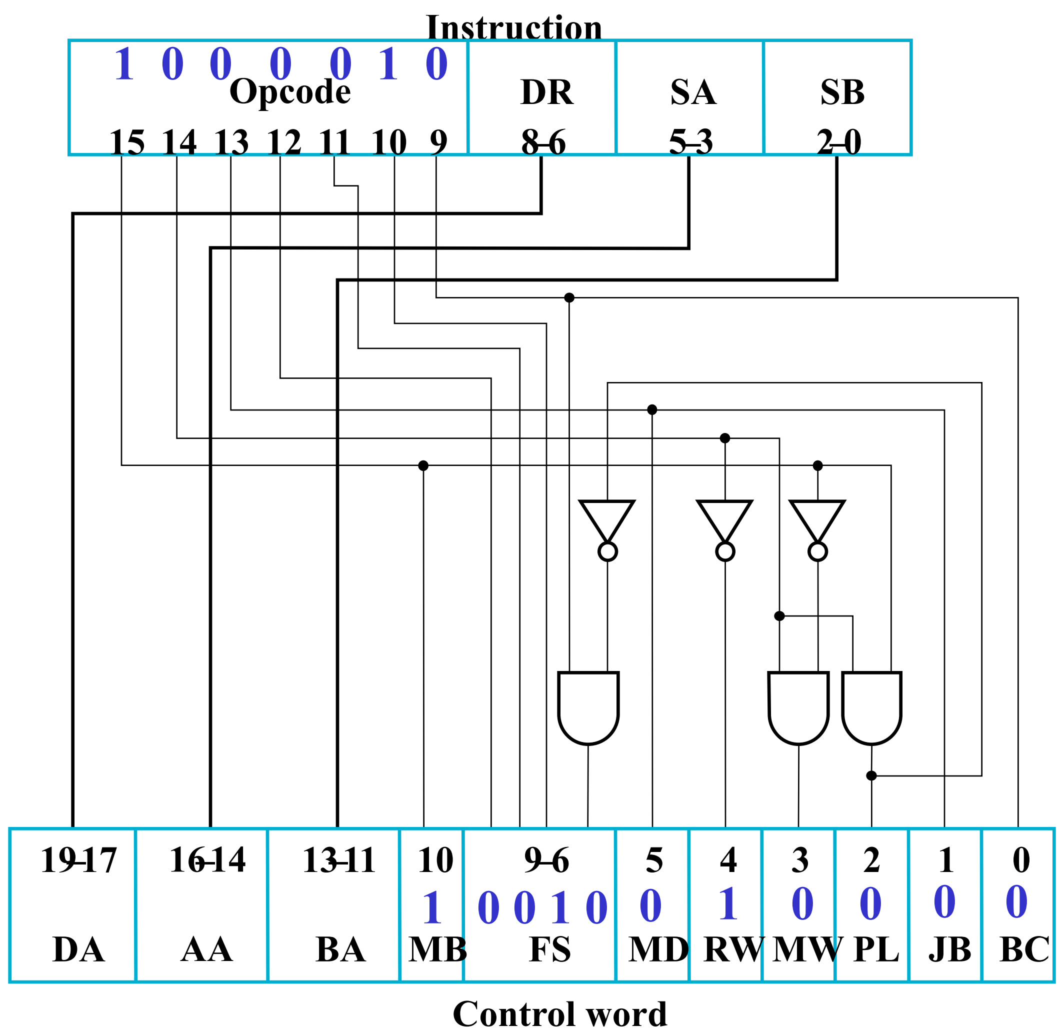
Example Instruction Execution

Six Instructions for the Single-Cycle Computer

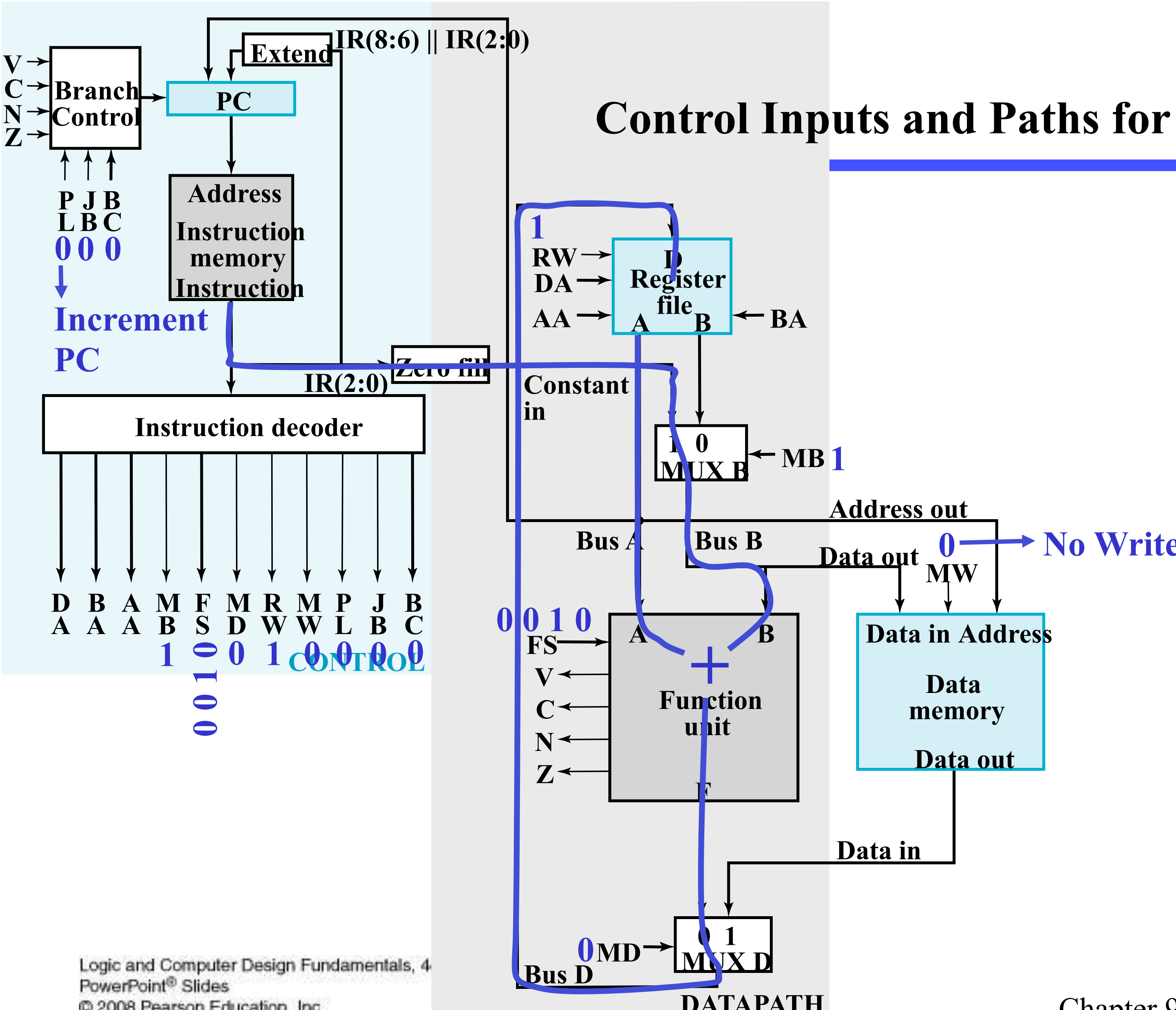
Operation Code	Symbolic Name	Format	Description	Function	MB	MD	RW	MW	PL	JB	BC
1000010	ADI	Immediate	Add immediate operand	$R[\mathrm{DR}] \leftarrow R[\mathrm{SA}] + \mathrm{zf}\ I(2{:}0)$	1	0	1	0	0	0	0
0010000	LD	Register	Load memory content into register	$R[\mathrm{DR}] \leftarrow M[R[\mathrm{SA}]]$	0	1	1	0	0	1	0
0100000	ST	Register	Store register content in memory	$M[R[\mathrm{SA}]] \leftarrow R[\mathrm{SB}]$	0	1	0	1	0	0	0
0001110	SL	Register	Shift left	$R[\mathrm{DR}] \leftarrow \mathrm{sl}\ R[\mathrm{SB}]$	0	0	1	0	0	1	0
0001011	NOT	Register	Complement register	$R[\mathrm{DR}] \leftarrow \overline{R[\mathrm{SA}]}$	0	0	1	0	0	0	1
1100000	BRZ	Jump/Branch	If $R[\mathrm{SA}] = 0$, branch to $PC + \mathrm{se}\ AD$	If $R[\mathrm{SA}] = 0$, $PC \leftarrow PC + \mathrm{se\ AD}$ If $R[\mathrm{SA}] \neq 0$, $PC \leftarrow PC + 1$	1	0	0	0	1	0	0

- **Decoding, control inputs and paths shown for ADI, LD and BRZ on next 6 slides**

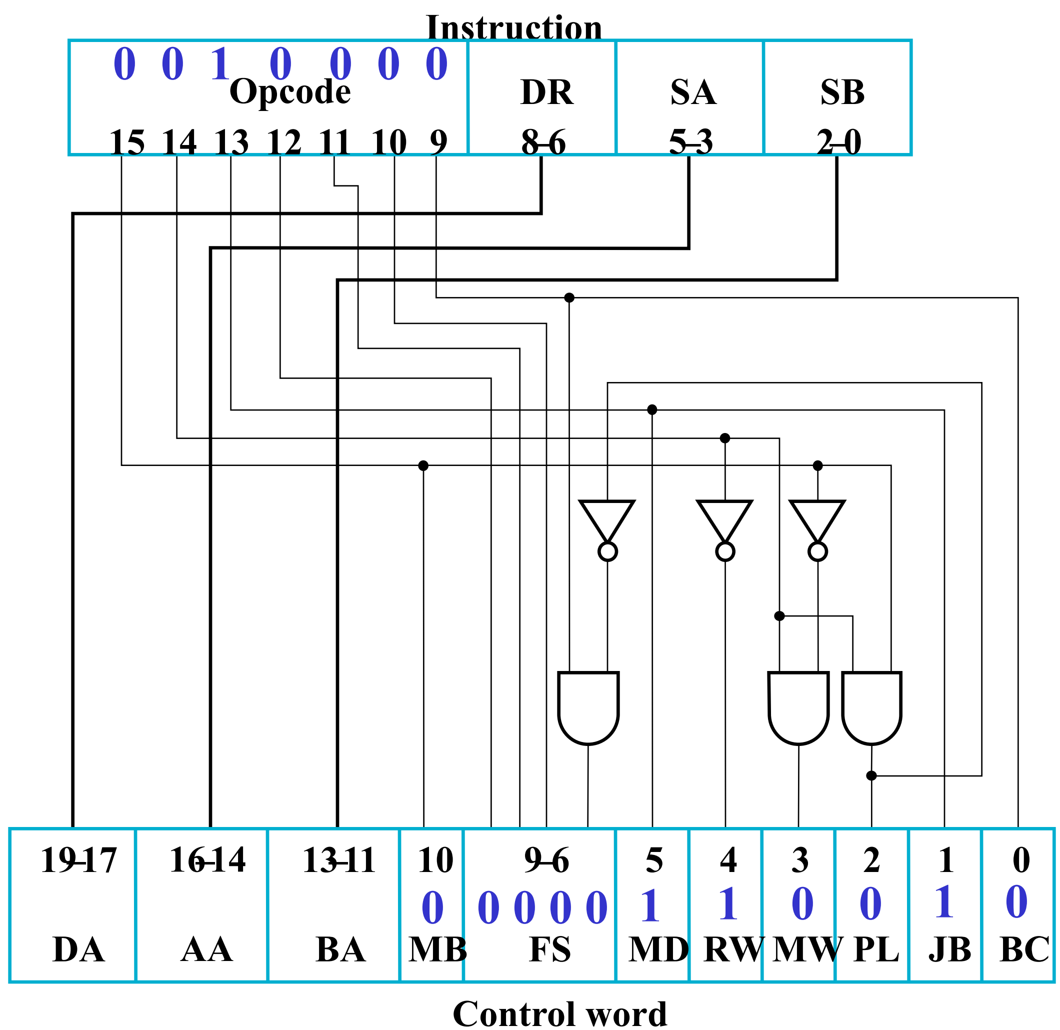
Decoding for ADI



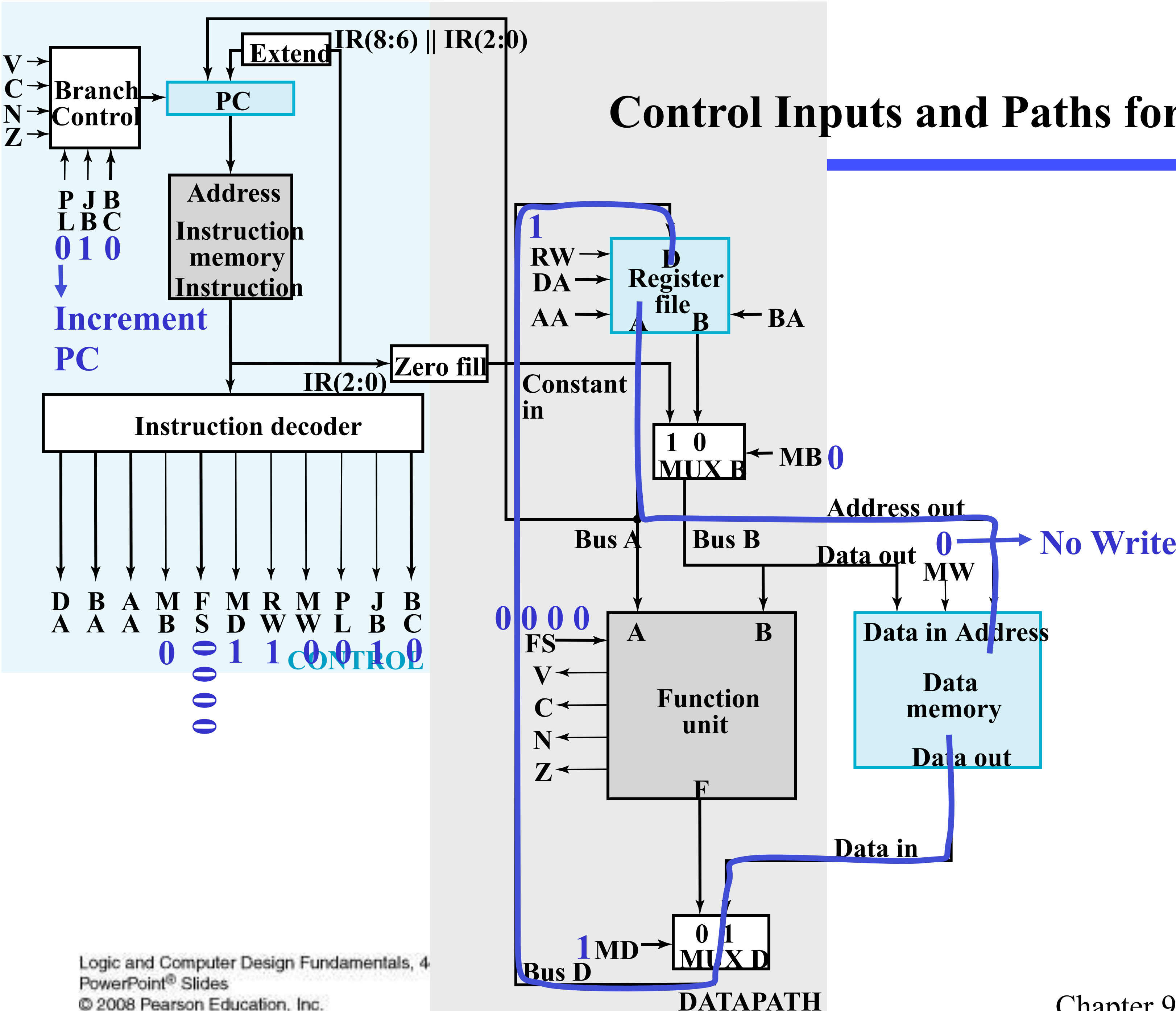
Control Inputs and Paths for ADI



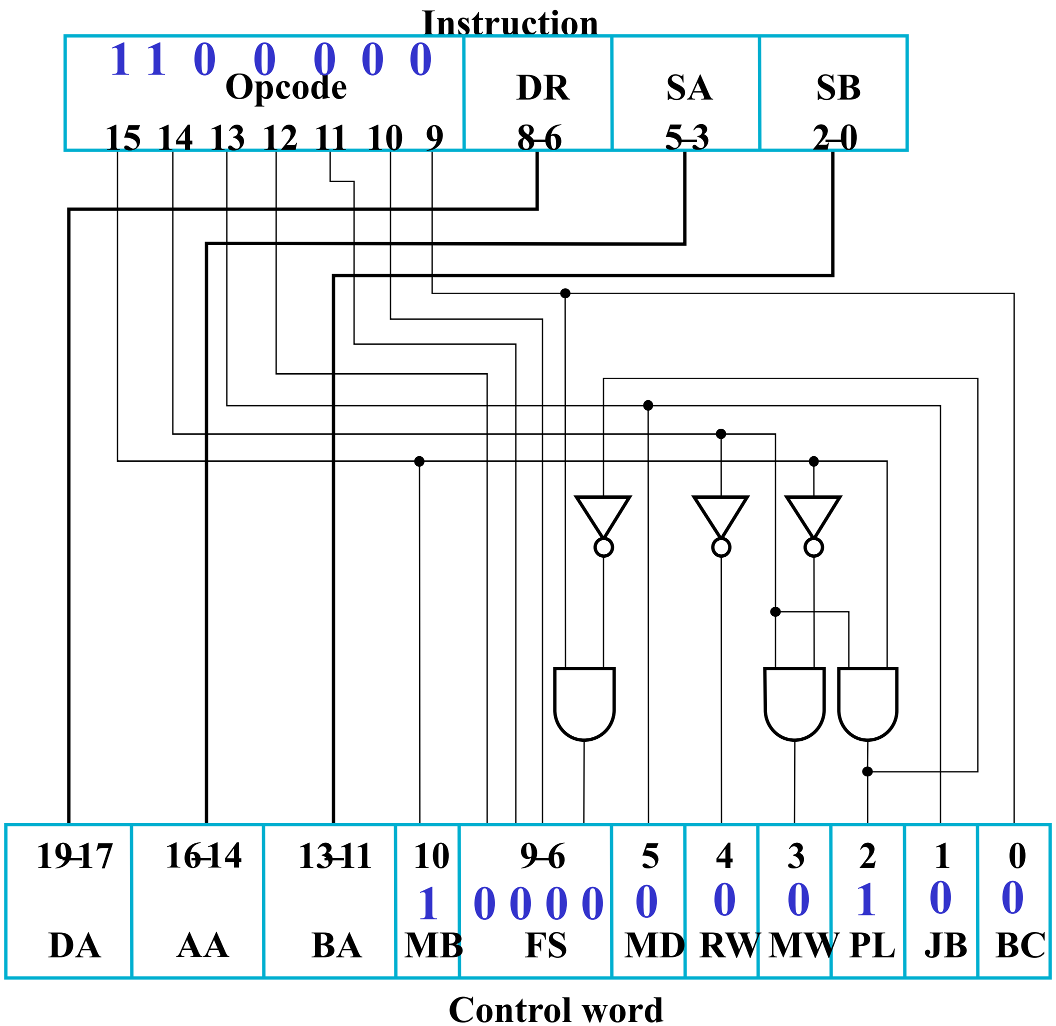
Decoding for LD



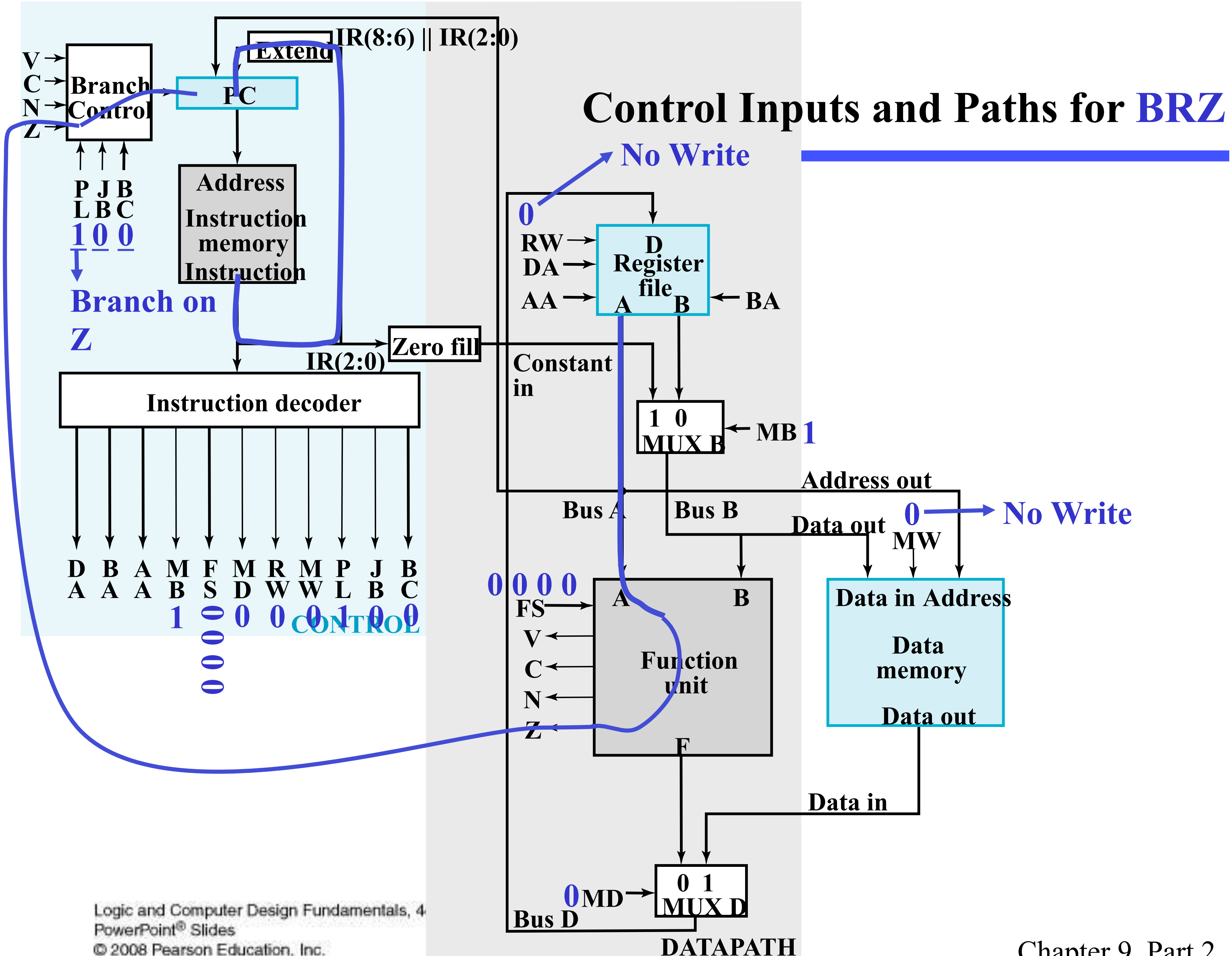
Control Inputs and Paths for LD



Decoding for BRZ

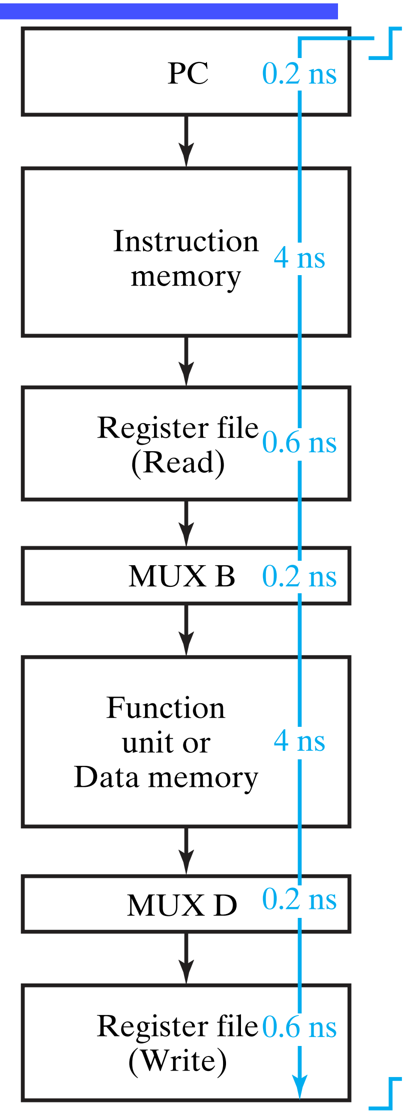


Control Inputs and Paths for BRZ



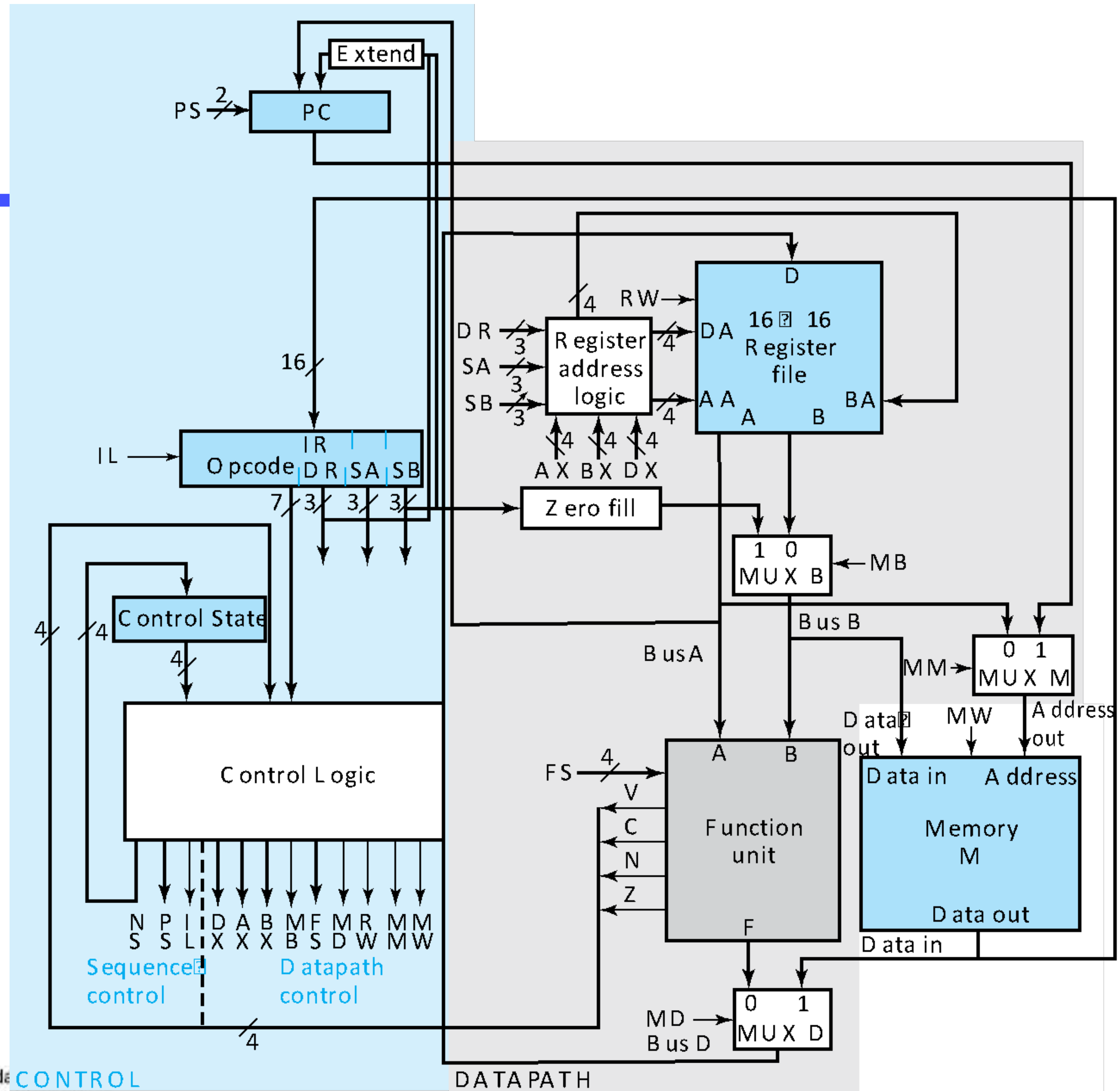
Single-Cycle Computer Issues

- **Shortcoming of Single Cycle Design**
  - **Complexity of instructions executable in a single cycle is limited**
  - **Accessing both an instruction and data from a simple single memory impossible**
  - **A long worst case delay path limits clock frequency and the rate of performing instructions**
- **Handling of Shortcomings**
  - **The first two shortcomings can be handled by the multiple-cycle computer discussed here**
  - **The third shortcoming is dealt with by using a technique called pipelining described in Chapter 12**



Multiple-Cycle Computer

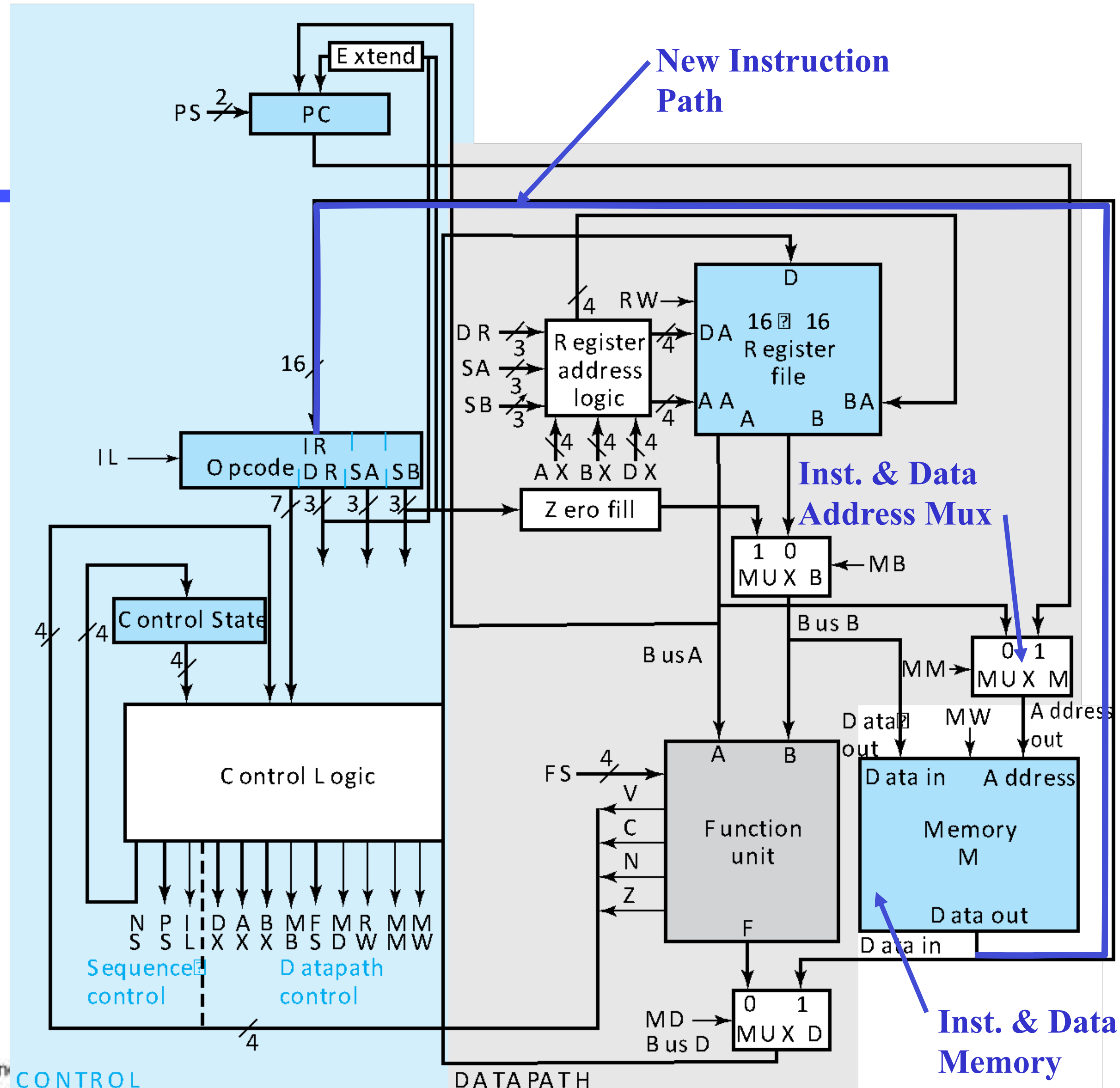
- Converting the single-cycle computer into a multiple-cycle computer involves:
  - Modifications to the datapath/memory
  - Modification to the control unit
  - Design of a multiple-cycle hardwired control
- The block diagram of the single-cycle SC architecture is given on the next slide for use in developing the multiple-cycle SC architecture



Extend
PS
2
PC
16
IL
IR
Opcode
DR
SA
SB
7
3
3
3
Control State
4
4
4
Control Logic
NS
PS
IL
DX
AX
BX
MB
FS
MD
RW
MM
MW
Sequencing control
Datapath control
4
CONTROL
DR
SA
SB
3
3
3
Register address logic
4
4
4
4
4
AX
BX
DX
RW
DA
AA
D
16 x 16 Register file
A
B
BA
Zero fill
1 0 MUX B
MB
Bus B
Bus A
0 1 MUX M
MM
Data out
MW
Address out
FS
4
V
C
N
Z
A
B
Function unit
F
Data in
Address
Memory M
Data out
Data in
0 1 MUX D
MD
Bus D
DATAPATH

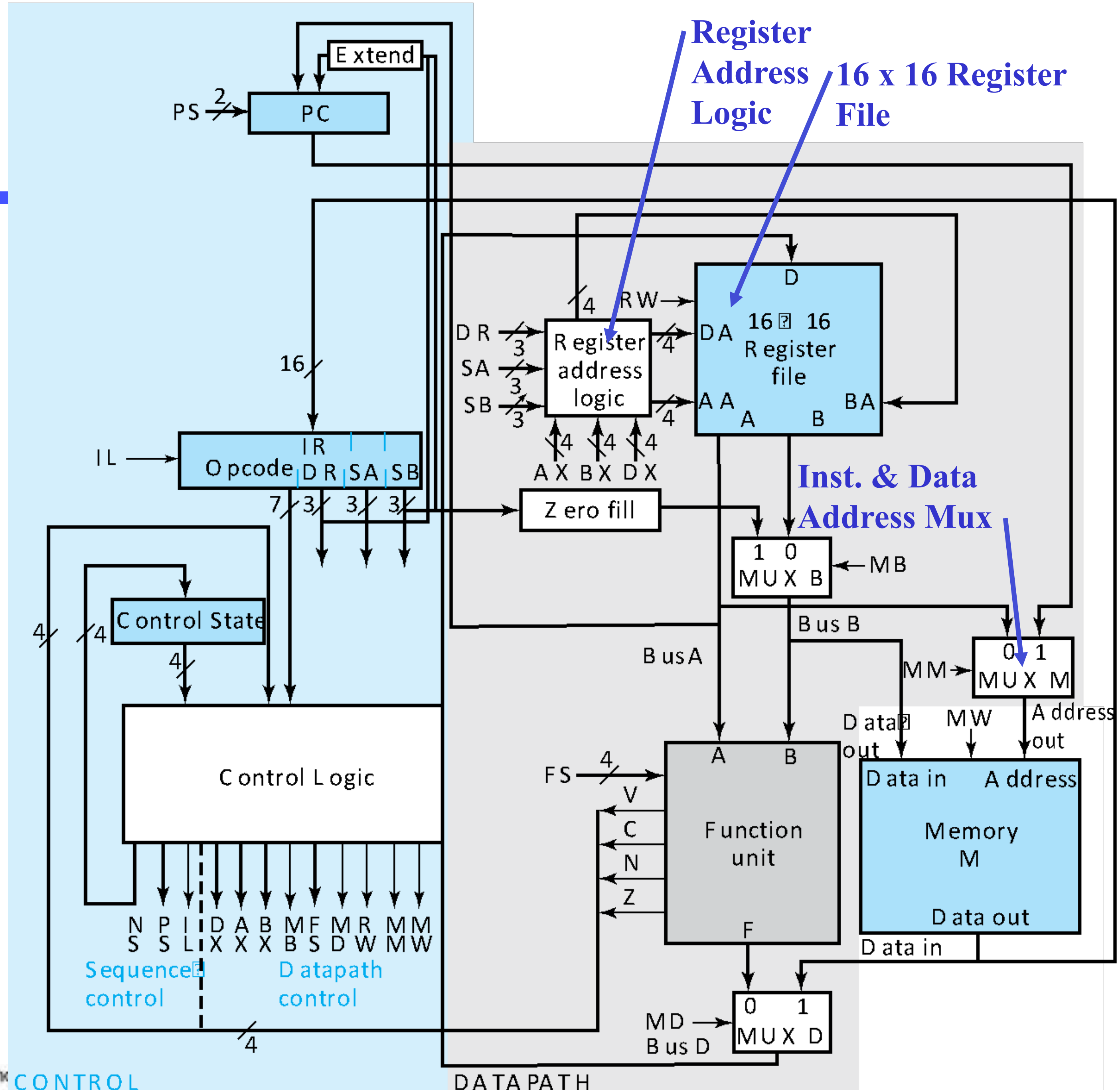
Datapath Modifications

- Modifications appear on the next slide
- Use a single memory for both instructions and data
  - Not essential to the multiple-cycle design, but done to illustrate the concept
  - Requires new MUX M with control signal MM to select the instruction address from the PC or the data address
  - Requires path from Memory Data Out to the instruction path in the control unit


New Instruction Path
Inst. & Data Address Mux
Inst. & Data Memory
Extend
PS
2
PC
16
IL
IR
Opcode
DR
SA
SB
7
3
3
3
Control State
4
4
4
Control Logic
N S
P S
I L
D X
A X
B X
M B
F S
M D
R W
M M
M W
Sequence control
Datapath control
4
CONTROL
DR
SA
SB
RW
Register address logic
AX
BX
DX
DA
AA
BA
D
16 × 16 Register file
A
B
Zero fill
1 0 MUX B
MB
Bus B
Bus A
MM
0 1 MUX M
Data out
MW
Address out
Data in
Address
Memory M
Data out
Data in
FS
V
C
N
Z
Function unit
F
MD
Bus D
0 1 MUX D
DATAPATH

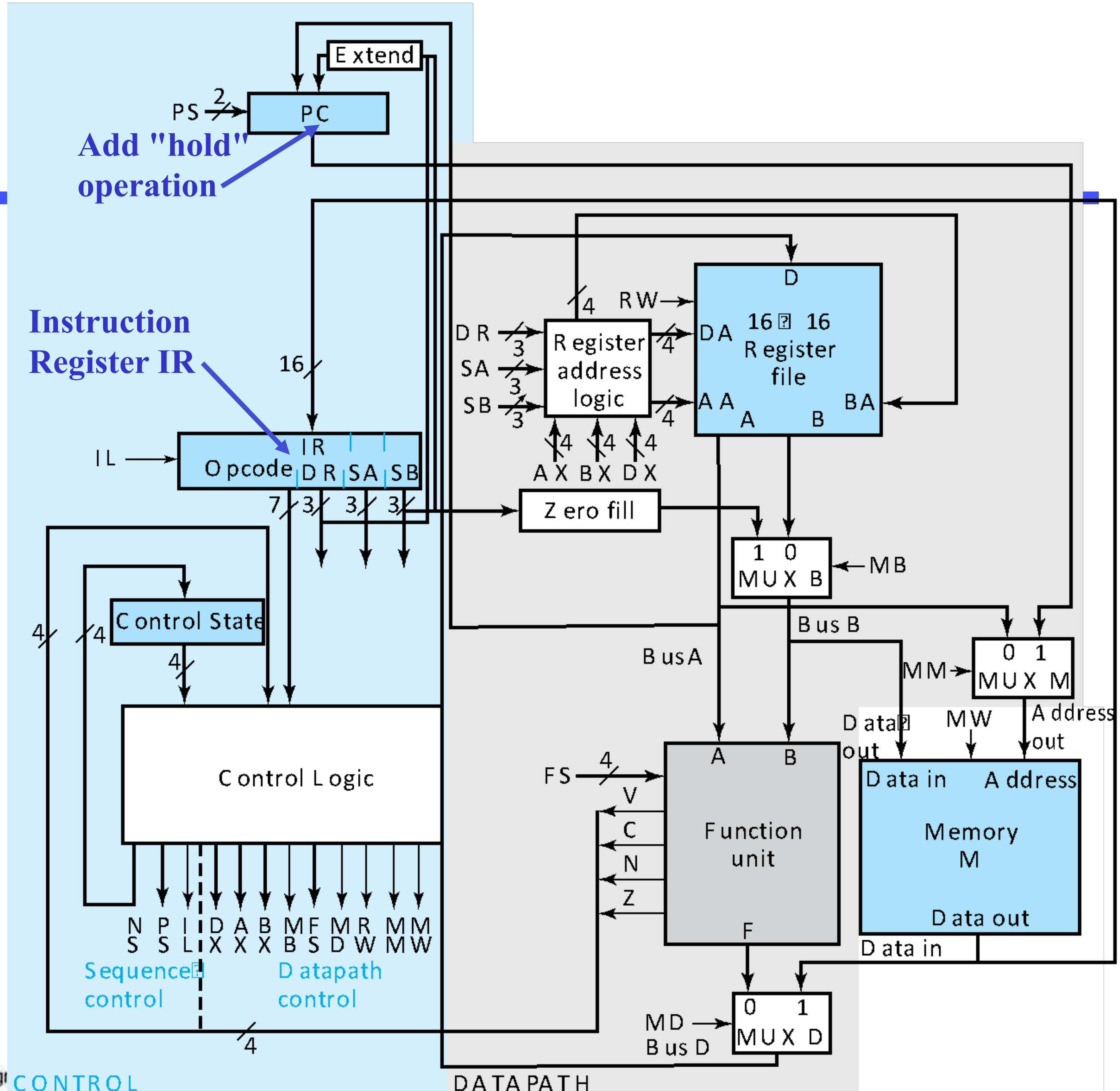
Datapath Modifications (continued)

- **To hold operands between cycles, need additional registers**
  - **Add 8 temporary storage registers to the Register File**
    - **Register File becomes 16 x 16**
    - **Addresses to Register File increase from 3 to 4 bits**
  - **Register File addresses come from:**
    - **The instruction for the Storage Resource registers (0 to 7)**
    - **The control word for the Temporary Storage registers (8 to 15)**
    - **The control word specifies the source for Register File addresses**
  - **Add Register Address Logic to the Register File to select the register address sources**
  - **Three new control fields for register address source selection and temporary storage addressing: DX, AX, BX**


Register Address Logic
16 x 16 Register File
Inst. & Data Address Mux
Extend
PS
2
PC
16
IL
IR
Opcode
DR
SA
SB
7
3
Zero fill
Control State
4
Control Logic
NS PS IL DX AX BX MB FS MD RW MM MW
Sequencing control
Datapath control
CONTROL
DR
SA
SB
Register address logic
AX BX DX
RW
DA
AA
BA
D
16 × 16 Register file
A
B
1 0 MUX B
MB
Bus B
Bus A
0 1 MUX M
MM
MW
Data out
Address out
Data in
Address
Memory M
Data out
Data in
FS
V
C
N
Z
Function unit
F
0 1 MUX D
MD
Bus D
DATAPATH

Control Unit Modifications

- **Must hold instruction over the multiple cycles to draw on instruction information throughout instruction execution**
  - **Requires an Instruction Register (IR) to hold the instruction**
    - **Load control signal IL**
  - **Requires the addition of a "hold" operation to the PC since it only counts up to obtain a new instruction**
    - **New encoding for the PC operations uses 2 bits**


Add "hold" operation
Instruction Register IR
Extend
PS
PC
IL
IR
Opcode
DR
SA
SB
Control State
Control Logic
N S
P S
I L
D X
A X
B X
M B
F S
M D
R W
M M
M W
Sequence control
Datapath control
CONTROL
RW
DR
SA
SB
Register address logic
DA
AA
BA
16 x 16 Register file
D
A
B
AX
BX
DX
Zero fill
MUX B
MB
Bus A
Bus B
MM
MUX M
Data out
MW
Address out
FS
Function unit
V
C
N
Z
F
Data in
Address
Memory M
Data out
Data in
MD
Bus D
MUX D
DATAPATH

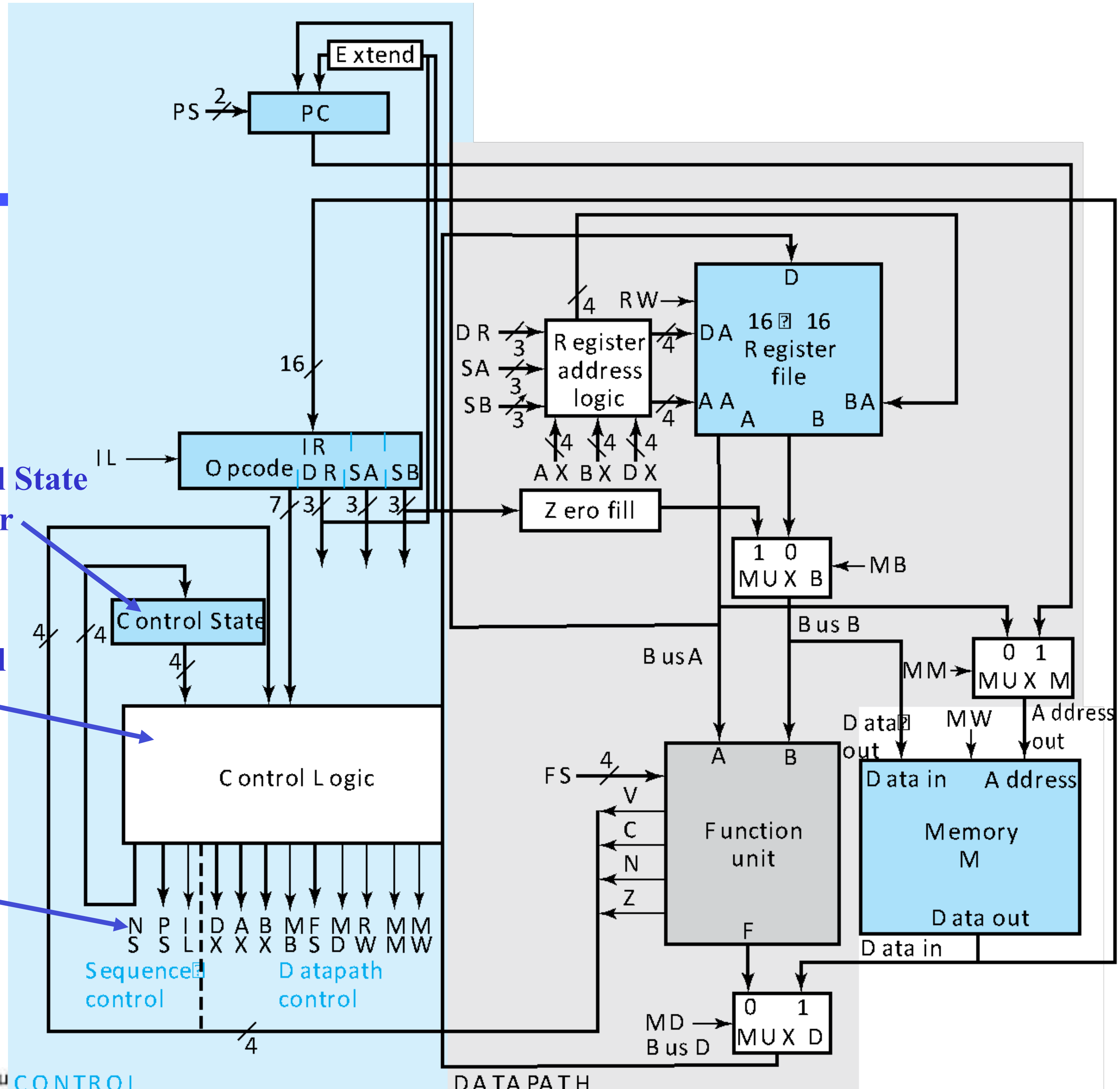
Sequential Control Design

- In order to control microoperations over multiple cycles, a Sequential Control replaces the Instruction Decoder
  - Input: Opcode, Status Bits
  - Output: Control Word (Modified Datapath Control part)
  - Control State
  - Next State: Control Word (New Sequencing Control part)
  - Consists of (see next slide):
    - Register to store the Control State
    - Combinational Logic to generate the Control Word (both sequencing and datapath control parts)
  - The Combinational Logic is quite complex so we assume that it is implemented by using a PLA or synthesized logic and focus on ASM level design

Control State Register

Combinational Control Logic

New/ Modified Control Word



Control Word

27 24	23 22	21	20 17	16 13	12 9	8	7 4	3	2	1	0
NS	PS	IL	DX	AX	BX	MB	FS	MD	RW	MM	MW

Sequencing (NS, PS, IL) — Datapath (DX through MW)

- **Datapath part: fields DA, AA, and BA replaced by DX, AX, and BX, respectively, and field MM added**
  - **If the MSB of a field is 0, e.g., AX = 0XXX, then AA is 0 concatenated with 3 bits obtained from the SA field in the IR**
  - **If the MSB of a field is 1, e. g. AX = 1011, then AA = 1011**
- **Sequencing part:**
  - **IL controls the loading of the IR**
  - **PS controls the operations of the PC**
  - **NS gives the next state of the Control State register**
    - **NS is 4 bits, the length of the Control State register - 16 states are viewed as adequate for this design**

Encoding for Datapath Control

DX	AX	BX	Code	MB	Code	FS	Code	MD	RW	MM	MW	Code
R[DR]	R[SA]	R[SB]	0XXX	Register	0	$F \leftarrow A$	0000	FnUt	No write	Address Out	No write	0
$R8$	$R8$	$R8$	1000	Constant	1	$F \leftarrow A + 1$	0001	Data In	Write	PC	Write	1
$R9$	$R9$	$R9$	1001			$F \leftarrow A + B$	0010					
$R10$	$R10$	$R10$	1010			Unused	0011					
$R11$	$R11$	$R11$	1011			Unused	0100					
$R12$	$R12$	$R12$	1100			$F \leftarrow A + \overline{B} + 1$	0101					
$R13$	$R13$	$R13$	1101			$F \leftarrow A - 1$	0110					
$R14$	$R14$	$R14$	1110			Unused	0111					
$R15$	$R15$	$R15$	1111			$F \leftarrow A \wedge B$	1000					
						$F \leftarrow A \vee B$	1001					
						$F \leftarrow A \oplus B$	1010					
						$F \leftarrow \overline{A}$	1011					
						$F \leftarrow B$	1100					
						$F \leftarrow \text{sr } B$	1101					
						$F \leftarrow \text{sl } B$	1110					
						Unused	1111					

Encoding for Sequencing Control

NS	PS		IL	
Next State	Action	Code	Action	Code
Gives next state of Control State Register	Hold PC	00	No load	0
	Inc PC	01	Load IR	1
	Branch	10		
	Jump	11		

Terms of Use

- All (or portions) of this material © 2008 by Pearson Education, Inc.
- Permission is given to incorporate this material or adaptations thereof into classroom presentations and handouts to instructors in courses adopting the latest edition of Logic and Computer Design Fundamentals as the course textbook.
- These materials or adaptations thereof are not to be sold or otherwise offered for consideration.
- This Terms of Use slide or page is to be included within the original materials or any adaptations thereof.