

Description of Single Cycle Computer (SCC)

References:

Chapter 9 of M. Morris Mano and Charles Kime, Logic and Computer Design Fundamentals, Pearson Prentice Hall, 4th Edition, 2008.

Overview

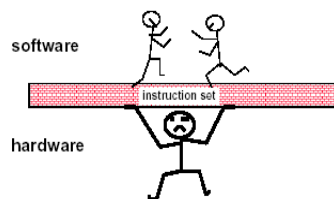
- Part 1 – Datapaths
 - Introduction
 - Datapath Example
 - Arithmetic Logic Unit (ALU)
 - Shifter
 - Datapath Representation and Control Word

- Part 2 – A Simple Computer
 - Instruction Set Architecture (ISA)
 - Single-Cycle Hardwired Control

- Part 3 – Multiple Cycle Hardwired Control
 - Single Cycle Computer Issues
 - Sequential Control Design

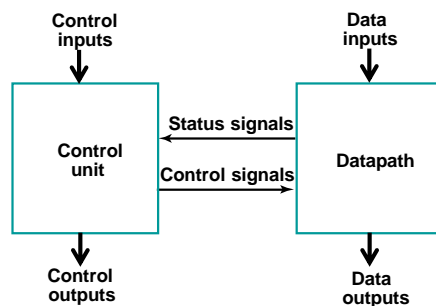
Introduction

- Computer Specification
 - **Instruction Set Architecture (ISA)** - the specification of a computer's appearance to a programmer at its lowest level
 - **Computer Architecture** - a high-level description of the hardware implementing the computer derived from the ISA
 - The architecture usually includes additional specifications such as speed/performance, cost, and reliability



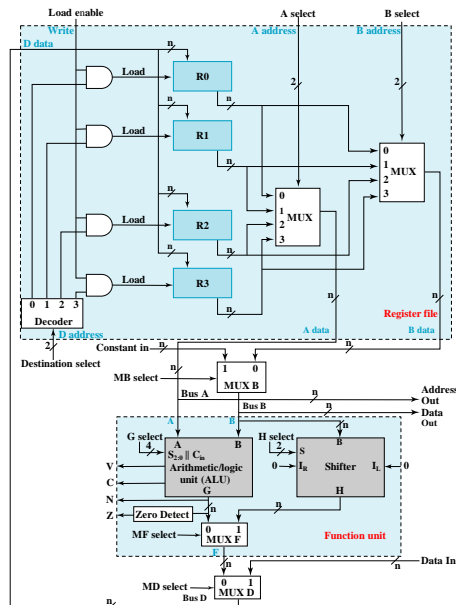
Introduction

- Simple computer architecture decomposed into:
 - **Datapath**: performing operations (i.e., data manipulation)
 - A set of registers
 - Microoperations performed on the data stored in the registers
 - A control interface
 - **Control unit**: controlling datapath operations
 - Programmable & Non-programmable



Datapath Example

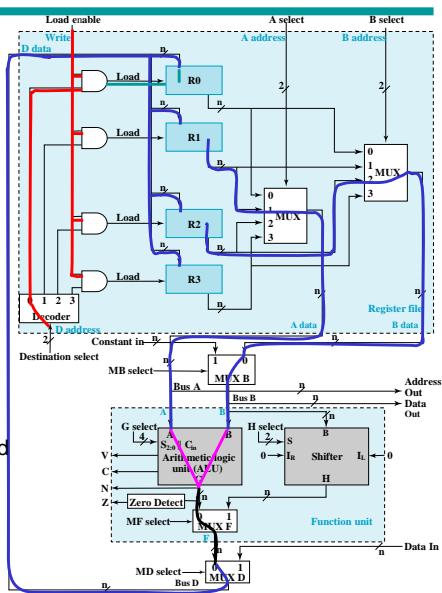
- Register file:
 - Four parallel-load regs
 - Two mux-based register selectors
 - Register destination decoder
- Microoperation implementation
 - Mux B for external constant input
 - Bus A and B with external address and data outputs
 - Function Unit:
 - ALU and Shifter with Mux F for output select
 - Mux D for external data input
 - Logic for generating status bits: V, C, N, Z



Datapath Example: Performing a Microoperation

Microoperation: $R0 \leftarrow R1 + R2$

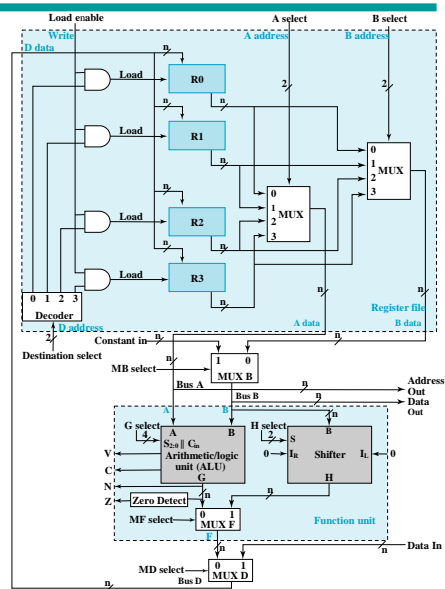
- Apply 01 to A select to place contents of R1 onto Bus A
- Apply 10 to B select to place contents of R2 onto B data and apply 0 to MB select to place B data on Bus B
- Apply 0010 to G select to perform addition $G = \text{Bus A} + \text{Bus B}$
- Apply 0 to MF select and 0 to MD select to place the value of G onto BUS D
- Apply 00 to Destination select to enable the Load input to R0
- Apply 1 to Load Enable to force the Load input to R0 to 1 so that R0 is loaded on the clock pulse (not shown)
- The overall microoperation requires 1 clock cycle



Datapath Example: Key Control Actions for Microoperation Alternatives

Various microoperations:

- Perform a shift microoperation:
apply 1 to MF select
- Use a constant in a micro-operation using Bus B: apply 1 to MB select
- Provide an address and data for a memory or output write microoperation – apply 0 to Load enable to prevent register loading
- Provide an address and obtain data for a memory or output read microoperation – apply 1 to MD select
- For some of the above, other control signals become don't cares

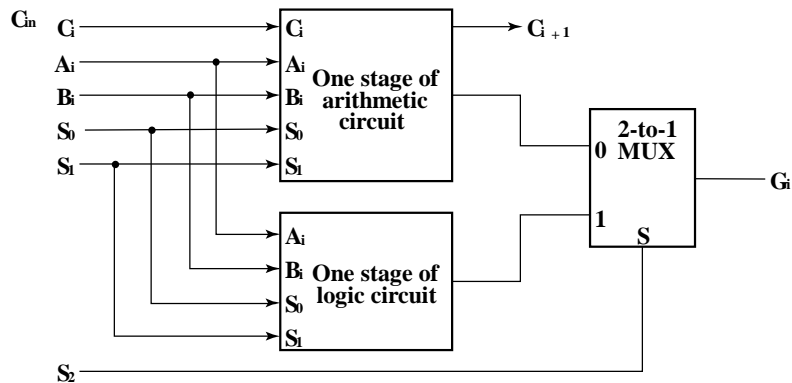


Overview

- Part 1 – Datapaths
 - Introduction
 - Datapath Example
 - Arithmetic Logic Unit (ALU)
 - Shifter
 - Datapath Representation and Control Word
- Part 2 – A Simple Computer
 - Instruction Set Architecture (ISA)
 - Single-Cycle Hardwired Control
- Part 3 – Multiple Cycle Hardwired Control
 - Single Cycle Computer Issues
 - Sequential Control Design

Arithmetic Logic Unit (ALU)

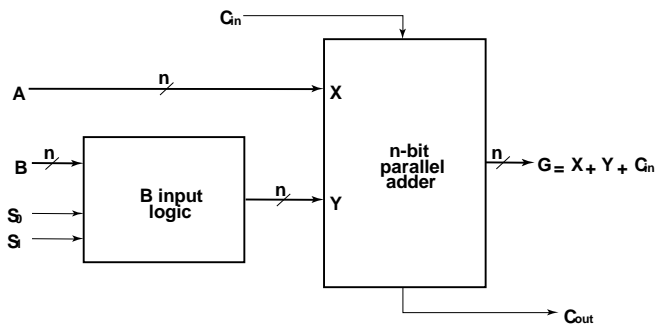
- Decompose the ALU into:
 - An arithmetic circuit & A logic circuit
 - A selector to pick between the two circuits



Arithmetic Circuit

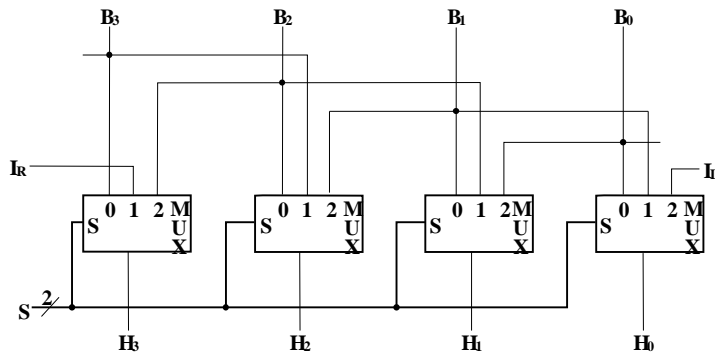
- Arithmetic circuit design
 - Decompose the arithmetic circuit into:
 - An n-bit parallel adder
 - A logic block that selects four choices for input B to the adder
- There are only four functions of B to select as Y in $G = A + Y + C_{in}$:

Y	$C_{in} = 0$	$C_{in} = 1$
0	$G = A$	$G = A + 1$
\overline{B}	$G = A + \overline{B}$	$G = A + \overline{B} + 1$
B	$G = A + B$	$G = A + B + 1$
1	$G = A - 1$	$G = A$



Arithmetic operations

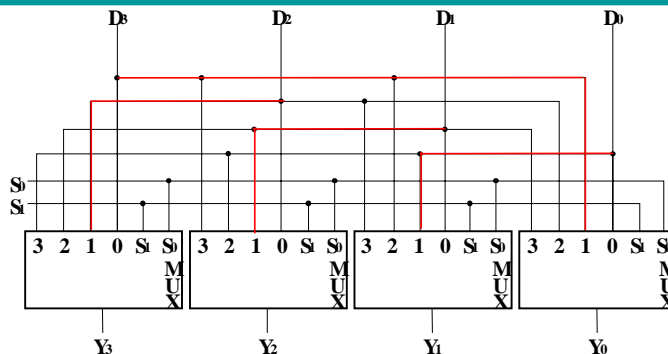
4-Bit Basic Left/Right Shifter



- Serial Inputs:
 - I_R for right shift
 - I_L for left shift
- Shift Functions:

$(S_1, S_0) = 00$	Pass B unchanged
01	Right shift
10	Left shift
11	Unused

Barrel Shifter

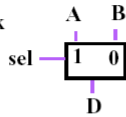


- A rotate is a shift in which the bits shifted out are inserted into the positions vacated
- The circuit rotates its contents left from 0 to 3 positions depending on S:

S = 00	position unchanged	S = 10	rotate left by 2 positions
S = 01	rotate left by 1 positions	S = 11	rotate left by 3 positions

Combinational Shifter from MUXes

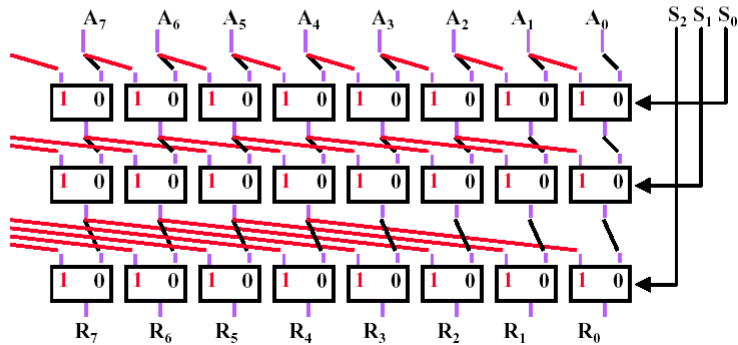
Basic Building Block



• Example 8-bit:

- Layer 1 shifts by 0, 4
- Layer 2 shifts by 0, 2
- Layer 3 shifts by 0, 1

8-bit right shifter



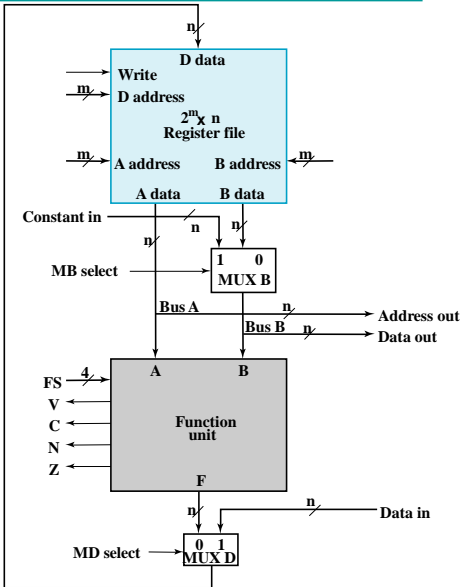
- Large barrel shifters can be constructed using:
 - Layers of multiplexers
 - 2-dimensional array circuits designed at the electronic level

Overview

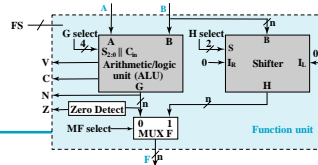
- Part 1 – Datapaths
 - Introduction
 - Datapath Example
 - Arithmetic Logic Unit (ALU)
 - Shifter
 - Datapath Representation and Control Word
- Part 2 – A Simple Computer
 - Instruction Set Architecture (ISA)
 - Single-Cycle Hardwired Control
- Part 3 – Multiple Cycle Hardwired Control
 - Single Cycle Computer Issues
 - Sequential Control Design

Datapath Representation

- In the register file:
 - Select inputs for multiplexers => A address & B address
 - Decoder input => D address
 - Load enable => write
 - Input data to the registers => D data
 - Multiplexer outputs => A data & B data
- The register file now appears like a memory based on clocked flip-flops
- FS?



Definition of Function Select (FS) Codes



FS(3:0)	MF Select	G Select(3:0)	H Select(1:0)	Microoperation
0000	0	0000	XX	$F \leftarrow A$
0001	0	0001	XX	$F \leftarrow A + 1$
0010	0	0010	XX	$F \leftarrow A + B$
0011	0	0011	XX	$F \leftarrow A + B + 1$
0100	0	0100	XX	$F \leftarrow A + \bar{B}$
0101	0	0101	XX	$F \leftarrow A + \bar{B} + 1$
0110	0	0110	XX	$F \leftarrow A - 1$
0111	0	0111	XX	$F \leftarrow A$
1000	0	1X00	XX	$F \leftarrow A \wedge B$
1001	0	1X01	XX	$F \leftarrow A \vee B$
1010	0	1X10	XX	$F \leftarrow A \oplus B$
1011	0	1X11	XX	$F \leftarrow \bar{A}$
1100	1	XXXX	00	$F \leftarrow B$
1101	1	XXXX	01	$F \leftarrow sr B$
1110	1	XXXX	10	$F \leftarrow sl B$

Boolean Equations:

$$MF_i = F_3 F_2$$

$$G_i = F_i$$

$$H_i = F_i$$

The Control Word

- The datapath has many control input signals, can be organized into a **control word**
- To execute a microinstruction, we apply control word values for a clock cycle

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DA		AA		BA		M B		FS			M D		R W		

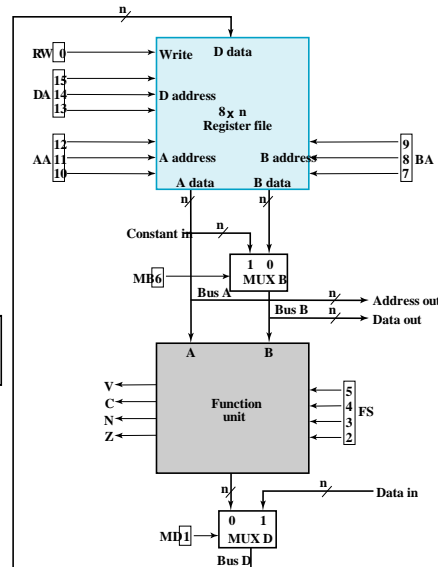
Control word

DA – D Address, AA – A Address

BA – B Address, MB – Mux B

FS – Function Select, MD – Mux D

RW – Register Write



Control Word Encoding

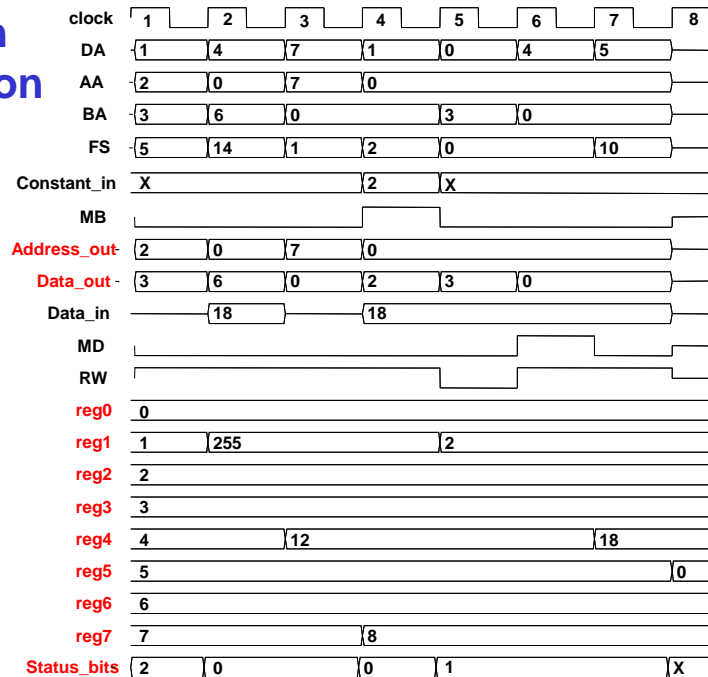
DA, AA, BA		MB		FS		MD		RW	
Function	Code	Function	Code	Function	Code	Function	Code	Function	Code
R0	000	Register	0	$F \leftarrow A$	0000	Function	0	No write	0
R1	001	Constant	1	$F \leftarrow A + 1$	0001	Data In	1	Write	1
R2	010			$F \leftarrow A + B$	0010				
R3	011			$F \leftarrow A + B + 1$	0011				
R4	100			$F \leftarrow A + \bar{B}$	0100				
R5	101			$F \leftarrow A + \bar{B} + 1$	0101				
R6	110			$F \leftarrow A - 1$	0110				
R7	111			$F \leftarrow A$	0111				
				$F \leftarrow A \wedge B$	1000				
				$F \leftarrow A \vee B$	1001				
				$F \leftarrow A \oplus B$	1010				
				$F \leftarrow \bar{A}$	1011				
				$F \leftarrow B$	1100				
				$F \leftarrow sr B$	1101				
				$F \leftarrow sl B$	1110				

Microoperations for the Datapath – Symbolic & Binary Representation

Micro-operation	DA	AA	BA	MB	FS	MD	RW
$R1 \leftarrow R2 - R3$	R1	R2	R3	Register	$F = A + \bar{B} + 1$	Function	Write
$R4 \leftarrow s! R6$	R4	—	R6	Register	$F = s! B$	Function	Write
$R7 \leftarrow R7 + 1$	R7	R7	—	Register	$F = A + 1$	Function	Write
$R1 \leftarrow R0 + 2$	R1	R0	—	Constant	$F = A + B$	Function	Write
Data out $\leftarrow R3$	—	—	R3	Register	—	—	No Write
$R4 \leftarrow$ Data in	R4	—	—	—	—	Data in	Write
$R5 \leftarrow 0$	R5	R0	R0	Register	$F = A \oplus B$	Function	Write

Micro-operation	DA	AA	BA	MB	FS	MD	RW
$R1 \leftarrow R2 - R3$	001	010	011	0	0101	0	1
$R4 \leftarrow s! R6$	100	XXX	110	0	1110	0	1
$R7 \leftarrow R7 + 1$	111	111	XXX	0	0001	0	1
$R1 \leftarrow R0 + 2$	001	000	XXX	1	0010	0	1
Data out $\leftarrow R3$	XXX	XXX	011	0	XXXX	X	0
$R4 \leftarrow$ Data in	100	XXX	XXX	X	XXXX	1	1
$R5 \leftarrow 0$	101	000	000	0	1010	0	1

Datapath Simulation



Overview

- Part 1 – Datapaths
 - Introduction
 - Datapath Example
 - Arithmetic Logic Unit (ALU)
 - Shifter
 - Datapath Representation and Control Word
- Part 2 – A Simple Computer
 - **Instruction Set Architecture (ISA)**
 - Single-Cycle Hardwired Control
- Part 3 – Multiple Cycle Hardwired Control
 - Single Cycle Computer Issues
 - Sequential Control Design

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

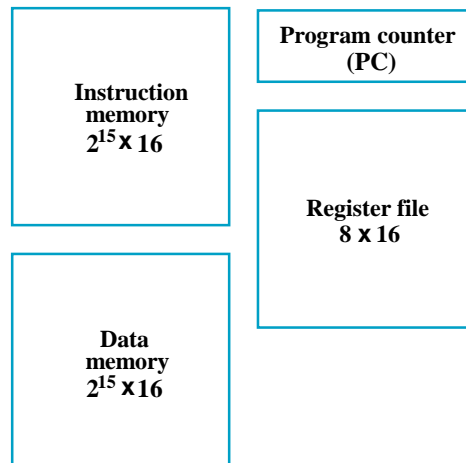
- **Instructions** are stored in RAM or ROM as a *program*, the addresses for instructions are provided by a *program counter (PC)*
 - Count up or load a new address
 - The PC and associated control logic are part of the Control Unit
- A typical instruction specifies:
 - Operands to use
 - Operation to be performed
 - Where to place the result, or which instruction to execute next
- Executing an instruction
 - Activate the necessary sequence of operations specified by the instruction
 - Be controlled by the control unit and performed in:
 - Datapath
 - Control unit
 - External hardware such as memory or input/output

ISA Examples

- RISC (Reduced Instruction Set Computer)
 - Digital Alpha
 - Sun Sparc
 - MIPS RX000
 - IBM PowerPC
 - HP PA/RISC
- CISC (Complex Instruction Set Computer)
 - Intel x86
 - Motorola 68000
 - DEC VAX
- VLIW (Very Large Instruction Word)
 - Intel Itanium

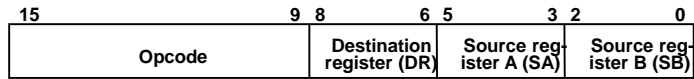
ISA: Storage Resources

- "Harvard architecture": separate instruction and data memories
- Permit use of single clock cycle per instruction implementation
- Due to use of "cache" in modern computer architectures, it is a fairly realistic model

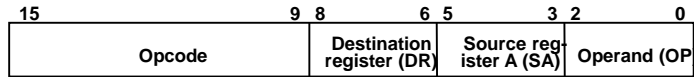


ISA: Instruction Formats

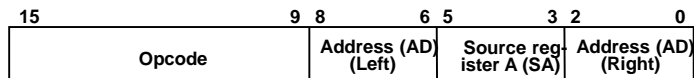
- The three formats are: Register, Immediate, and Jump/Branch



(a) Register



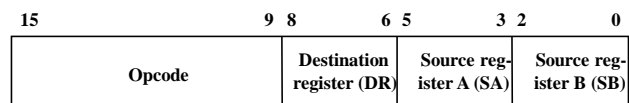
(b) Immediate



(c) Jump and Branch

- All formats contain an Opcode field in bits 9 through 15.
 - The Opcode specifies the operation to be performed

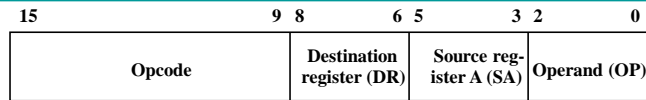
ISA: Instruction Format - Register



(a) Register

- This format supports:
 - $R1 \leftarrow R2 + R3$
 - $R1 \leftarrow \text{sl } R2$
- Three 3-bit register fields:
 - DR - destination register (R1 in the examples)
 - SA - the A source register (R2 in the first example)
 - SB - 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?

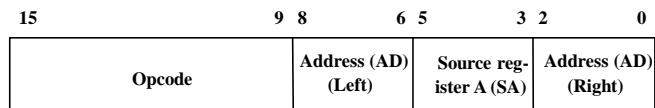
ISA: Instruction Format - Immediate



(b) Immediate

- This format supports:
 - $R1 \leftarrow R2 + 3$
- The B Source Register field is replaced by an Operand field OP specifying a constant. (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 - Jump & Branch



(c) Jump and Branch

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

ISA: Instruction Specifications

Instruction	Opcode	Mnemonic	Format	Description	Status Bits
Move A	000000	MOVA	RD,RA	$R[DR] \leftarrow R[SA]$	N, Z
Increment	000001	INC	RD,RA	$R[DR] \leftarrow R[SA] + 1$	N, Z
Add	000010	ADD	RD,RA,SB	$R[DR] \leftarrow R[SA] + R[SB]$	N, Z
Subtract	000011	SUB	RD,RA,SB	$R[DR] \leftarrow R[SA] - R[SB]$	N, Z
Decrement	000110	DEC	RD,RA	$R[DR] \leftarrow R[SA] - 1$	N, Z
AND	000100	AND	RD,RA,SB	$R[DR] \leftarrow R[SA] \wedge R[SB]$	N, Z
OR	000101	OR	RD,RA,SB	$R[DR] \leftarrow R[SA] \vee R[SB]$	N, Z
Exclusive OR	000110	XOR	RD,RA,SB	$R[DR] \leftarrow R[SA] \oplus R[SB]$	N, Z
NOT	000111	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	$R[DR] \leftarrow R[SA] + zf OP$	
Load	0010000	LD	RD,RA	$R[DR] \leftarrow M[R[SA]]$	
Store	0100000	ST	RA,RB	$M[R[SA]] \leftarrow R[SB]$	
Branch on Zero	1100000	BRZ	RA,AD	if $(R[SA] = 0)$ $PC \leftarrow PC + se AD$	
Branch on Negative	1100001	BRN	RA,AD	if $(R[SA] < 0)$ $PC \leftarrow PC + se AD$	
Jump	1110000	JMP	RA	$PC \leftarrow R[SA]$	

ISA: Example Instructions and Data in Memory

Memory Representation of Instruction and Data

Decimal Address	Memory Contents	Decimal Opcode	Other Field	Operation
25	0000101 001 010 011	5 (Subtract)	DR:1, SA:2, SB:3	$R1 \leftarrow R2 - R3$
35	0100000 000 100 101	32 (Store)	SA:4, SB:5	$M[R4] \leftarrow R5$
45	1000010 010 111 011	66 (Add Immediate)	DR: 2, SA :7, OP :3	$R2 \leftarrow R7 + 3$
55	1100000 101 110 100	96 (Branch on Zero)	AD: 44, SA:6	If $R6 = 0$, $PC \leftarrow PC - 20$
70	0000000 001100 000	Data = 192. After execution of instruction in 35, Data = 80.		

Overview

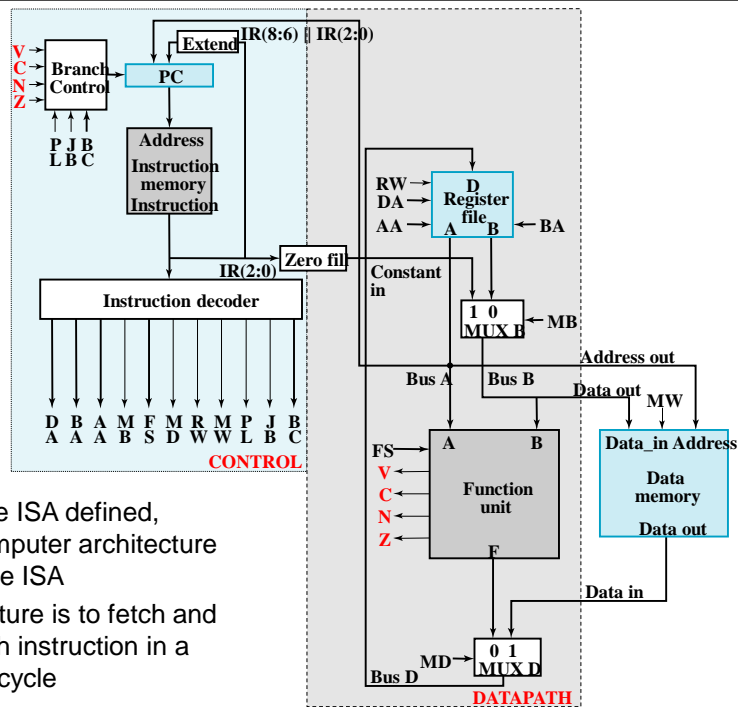
- Part 1 – Datapaths
 - Introduction
 - Datapath Example
 - Arithmetic Logic Unit (ALU)
 - Shifter
 - Datapath Representation and Control Word

- Part 2 – A Simple Computer
 - Instruction Set Architecture (ISA)
 - Single-Cycle Hardwired Control

- Part 3 – Multiple Cycle Hardwired Control
 - Single Cycle Computer Issues
 - Sequential Control Design

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 Control Unit

- Datapath: the Data Memory has been attached to the *Address Out*, *Data Out*, and *Data In* lines of the Datapath.
- Control Unit:
 - The *MW* input to the Data Memory is the Memory Write signal from the Control Unit.
 - 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

Program Counter (PC) Function

- PC function is based on instruction specifications involving jumps and branches:

Branch on Zero	BRZ	if (R[SA] = 0) PC ← PC + seAD
Branch on Negative	BRN	if (R[SA] < 0) PC ← PC + seAD
Jump	JMP	PC ← R[SA]

- The first two transfers 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]
- In addition to the above register transfers, the PC must implement the counting function:
 - PC ← PC + 1

PC Function (Contd.)

- Branch Control determines the PC transfers based on five inputs:
 - 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.

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

Instruction Decoder

- Converts the instruction into the signals necessary to control the computer during the single cycle execution, combinational
 - Inputs: the 16-bit Instruction
 - Outputs: control signals
 - DA, AA, and BA: Register file addresses (IR (8:0))
 - simply pass-through signals: DA = DR, AA = SA, and BA = SB
 - FS: Function Unit Select
 - MB and MD: Multiplexer Select Controls
 - RW and MW: Register file and Data Memory Write Controls
 - PL, JB, and BC: PC Controls
- Observe that for other than branches and jumps, FS = IR(12:9)
 - The other control signals should depend as much as possible on IR(15:13)

Instruction Decoder (Contd.)

Truth Table for Instruction Decoder Logic

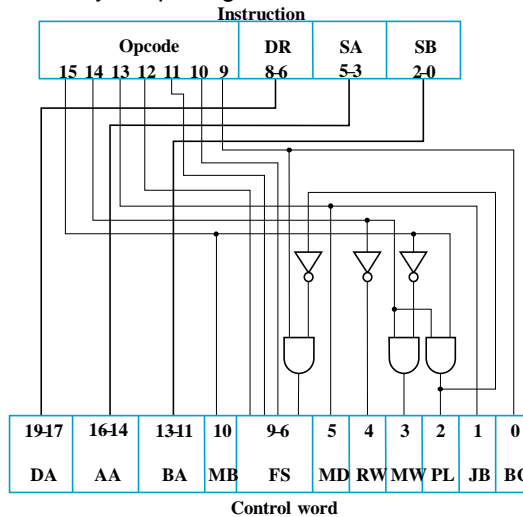
Instruction Function Type	Instruction Bits				Control Word Bits							
	15	14	13	9	MB	MD	RW	MW	PL	JB	BC	
1. Function unit operations using registers	0	0	0	X	0	0	1	0	0	X	X	
2. Memory read	0	0	1	X	0	1	1	0	0	X	X	
3. Memory write	0	1	0	X	0	X	0	1	0	X	X	
4. Function unit operations using register and constant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

Instruction Decoder (Contd.)

- Instruction types are based on the control blocks and the seven control signals to be generated (MB, MD, RW, MW, PL, JB, BC):
 - Datapath and Memory Control (types 1-4)
 - Mux B
 - Memory and Mux D
 - PC Control (types 5-7)
 - Bit 15 = Bit 14 = 1 => PL
 - Bit 13 => JB.
 - Bit 9 was use as BC which contradicts FS = 0000 needed for branches. To force FS(0) to 0 for branches, Bit 9 into FS(0) is disabled by PL.

Instruction Decoder (Contd.)

- 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



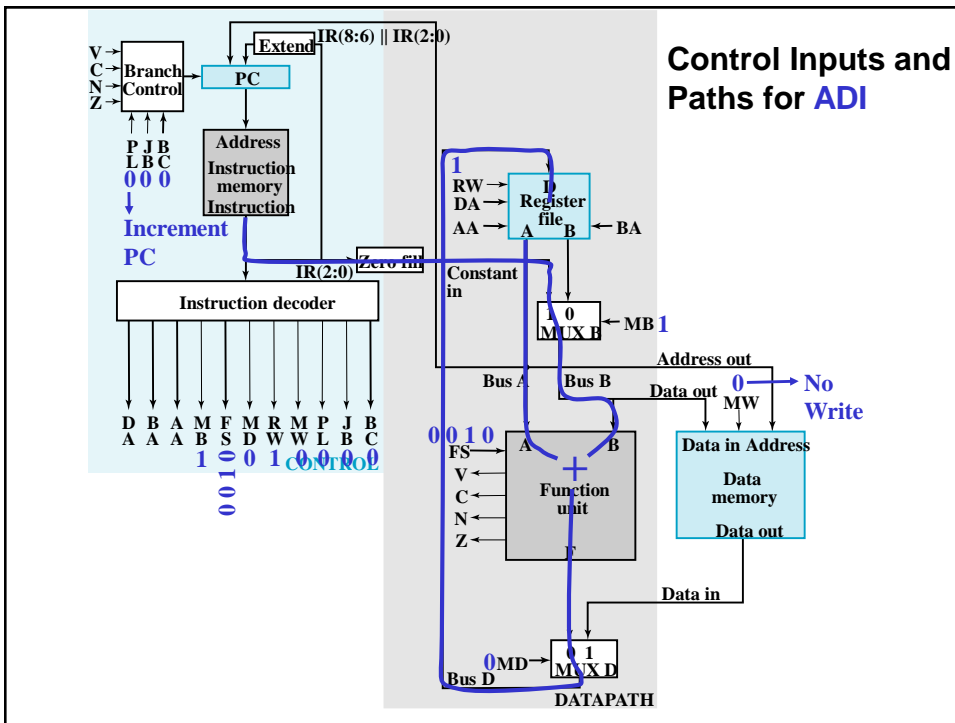
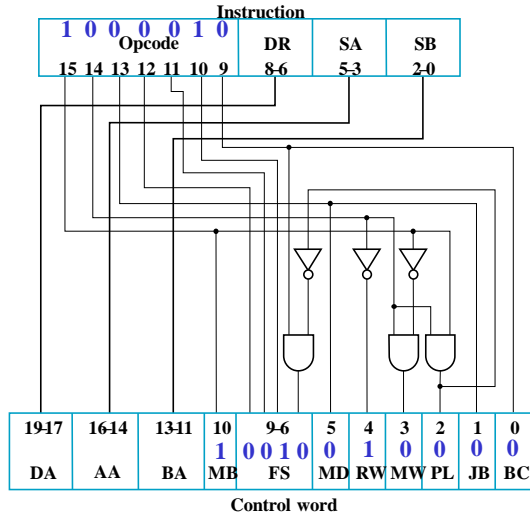
Example Instruction Execution

Six Instructions for the Single-Cycle Computer

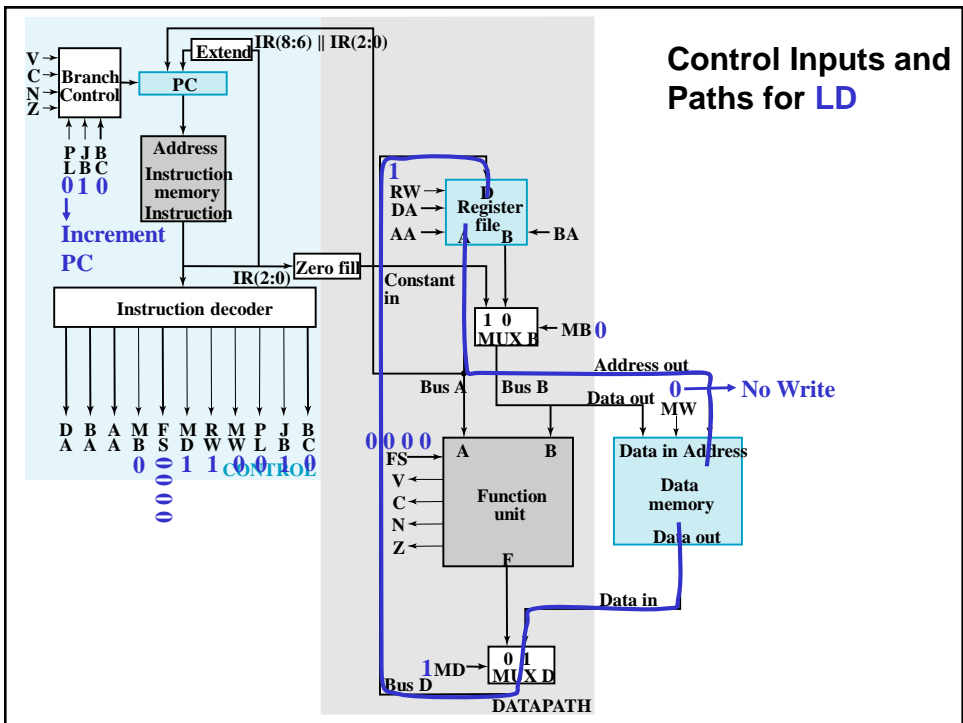
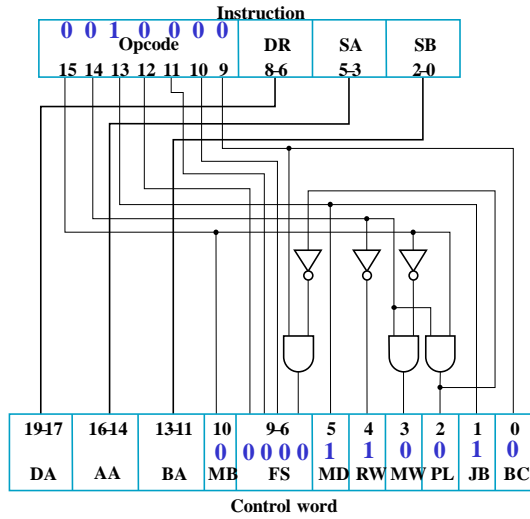
Operation code	Symbolic Name	Format	Description	Function	MB	MD	RW	MW	PL	JB	BC
1000 010	ADI	Immediate	Add immediate operand	$R[DR] \leftarrow R[SA] + zf / (2:0)$	1	0	1	0	0	0	0
0010 000	LD	Register	Load memory content in to register	$R[DR] \leftarrow M[R[SA]]$	0	1	1	0	0	1	0
0100 000	ST	Register	Store register content in memory	$M[R[SA]] \leftarrow R[SB]$	0	1	0	1	0	0	0
0001 110	SL	Register	Shift left	$R[DR] \leftarrow sl R[SB]$	0	0	1	0	0	1	0
0001 011	NOT	Register	Complement register	$R[DR] \leftarrow \overline{R[SA]}$	0	0	1	0	0	0	1
1100 000	BRZ	Jump/Branch	If $R[SA] = 0$, branch to PC + se AD	If $R[SA] = 0$, $PC \leftarrow PC + se AD$ If $R[SA] \neq 0$, $PC \leftarrow PC + 1$	1	0	0	0	1	0	0

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

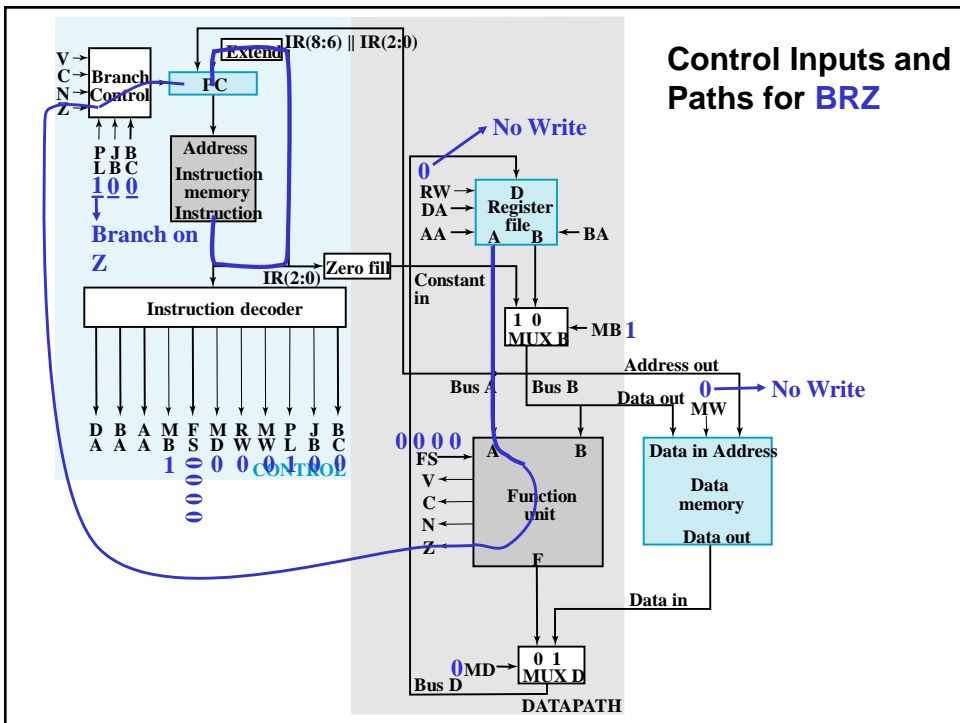
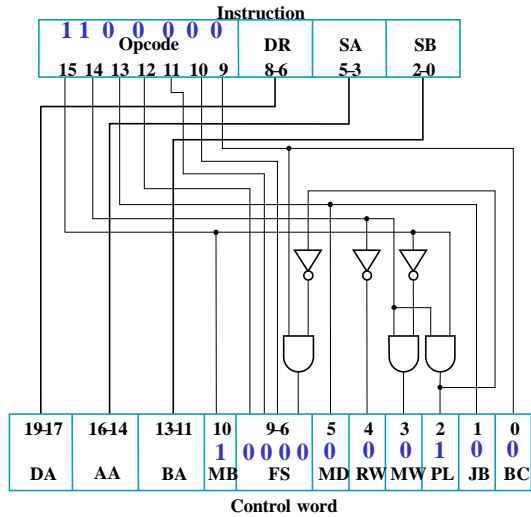
Decoding for ADI



Decoding for LD



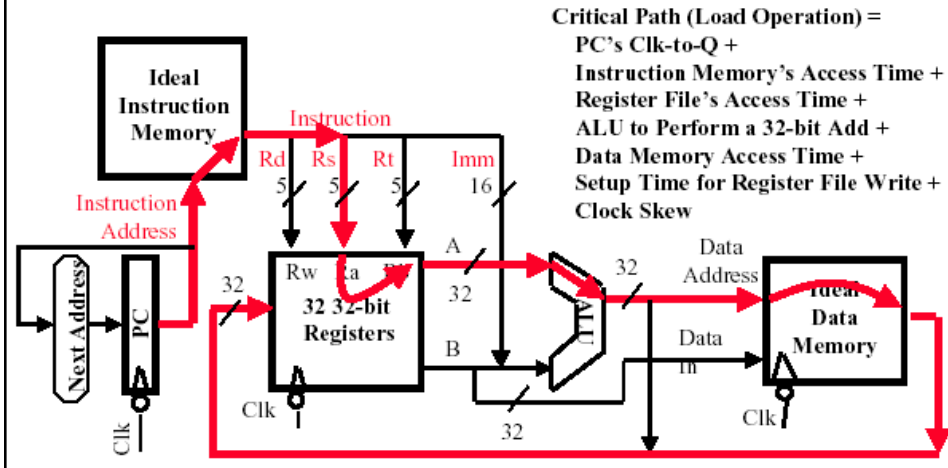
Decoding for BRZ



Abstract View of Critical Path

◦ Register file and ideal memory:

- The CLK input is a factor ONLY during write operation
- During read operation, behave as combinational logic:
 - Address valid => Output valid after “access time.”



Overview

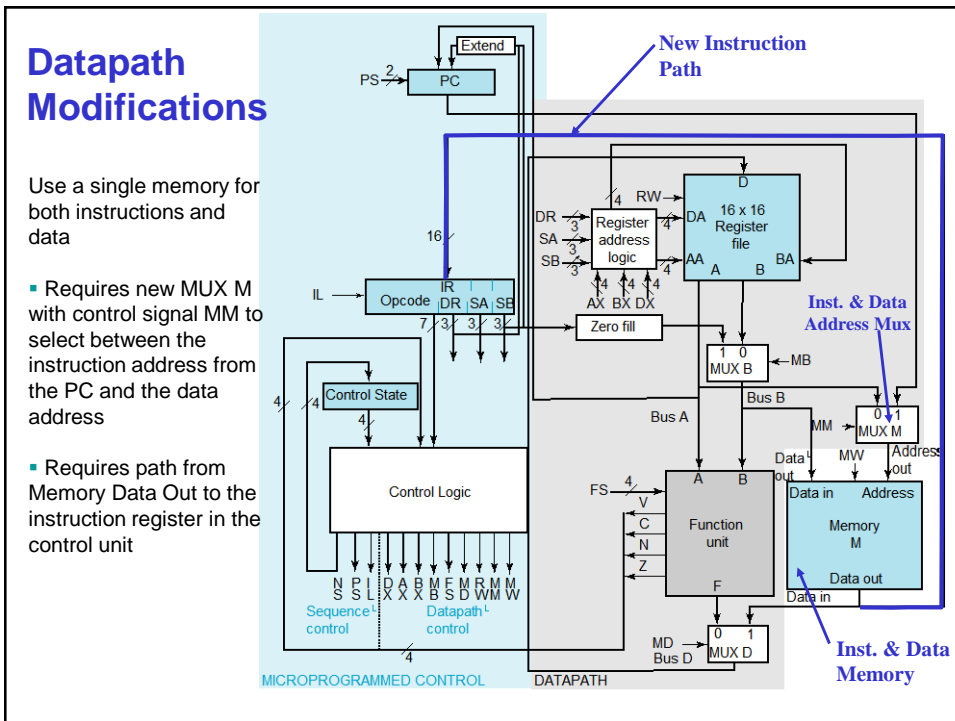
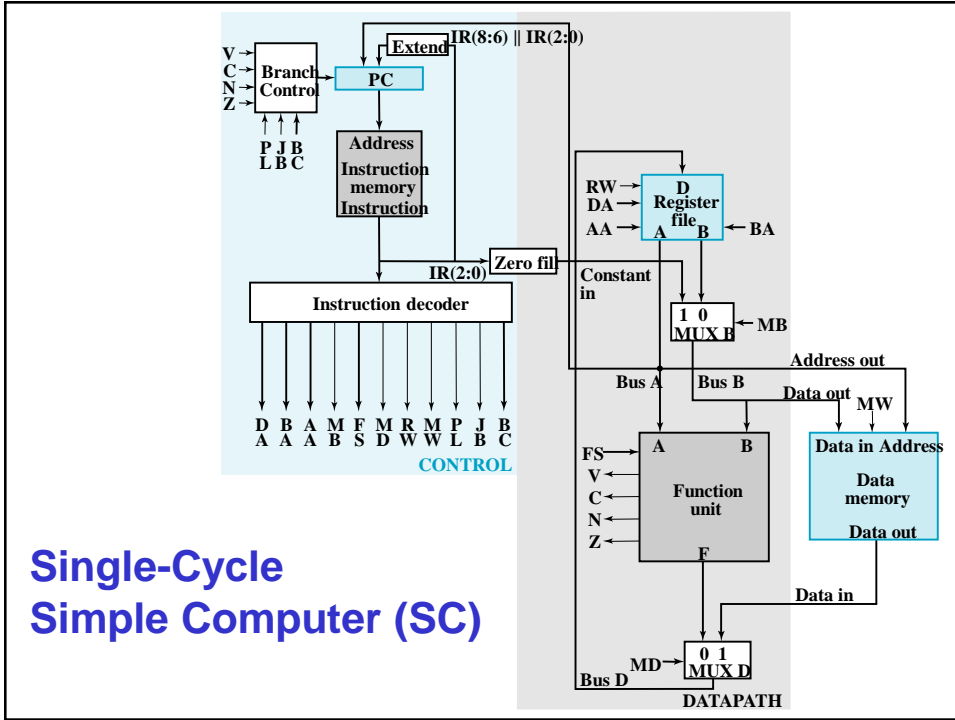
- Part 1 – Datapaths
 - Introduction
 - Datapath Example
 - Arithmetic Logic Unit (ALU)
 - Shifter
 - Datapath Representation and Control Word
- Part 2 – A Simple Computer
 - Instruction Set Architecture (ISA)
 - Single-Cycle Hardwired Control
- Part 3 – Multiple Cycle Hardwired Control
 - Single Cycle Computer Issues
 - Sequential Control Design

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
 - The third shortcoming is dealt with by using a technique called **pipelining** described in later lectures

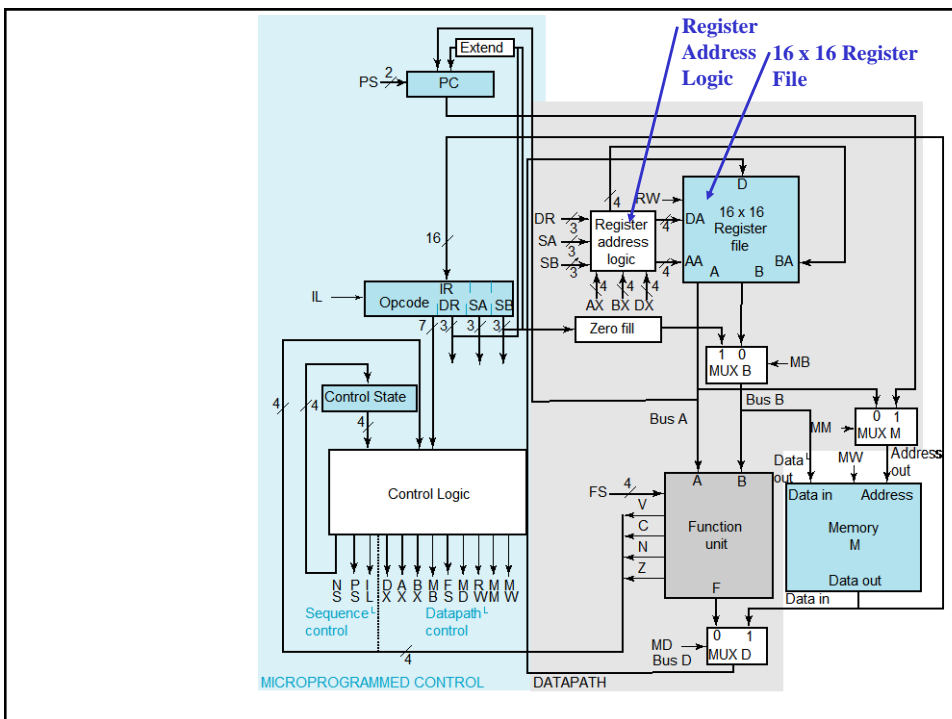
Multiple-Cycle Computer

- 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



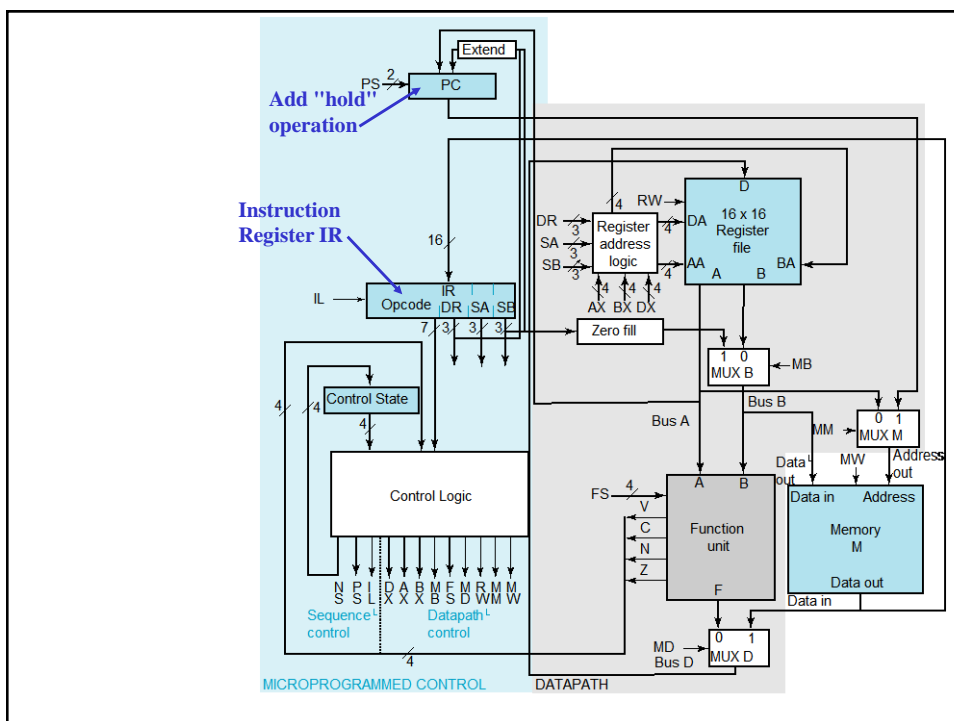
Datapath Modifications (Continued)

- Additional registers needed to hold operands between cycles
 - 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)
 - 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



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

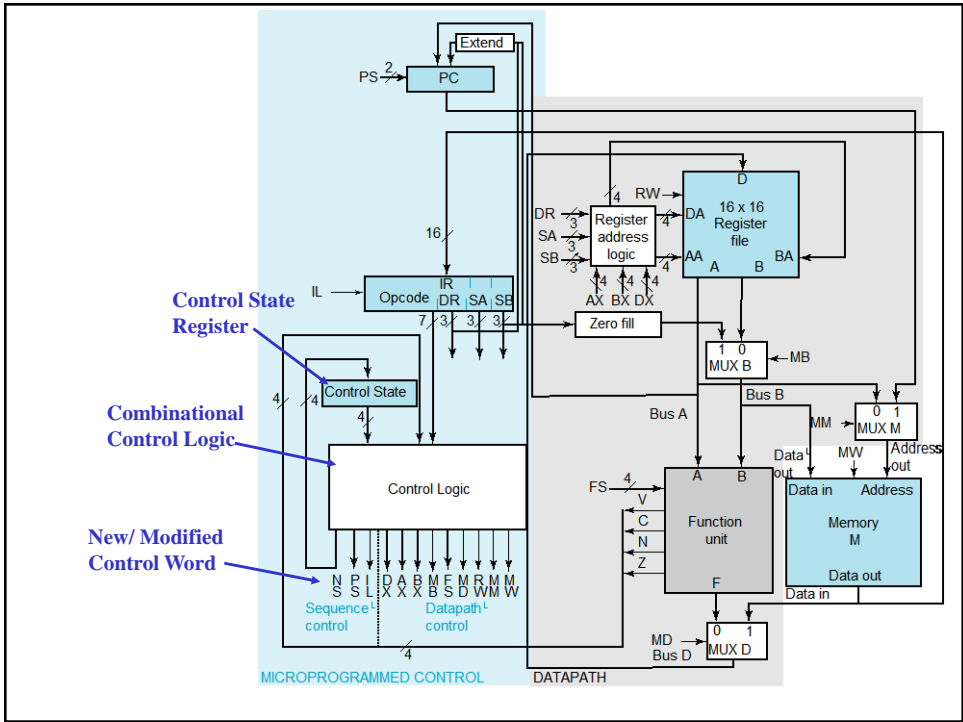


Overview

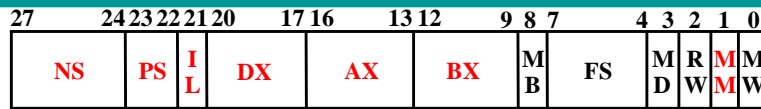
- Part 1 – Datapaths
 - Introduction
 - Datapath Example
 - Arithmetic Logic Unit (ALU)
 - Shifter
 - Datapath Representation and Control Word
- Part 2 – A Simple Computer
 - Instruction Set Architecture (ISA)
 - Single-Cycle Hardwired Control
- Part 3 – Multiple Cycle Hardwired Control
 - Single Cycle Computer Issues
 - **Sequential Control Design**

Sequential Control Design

- To control microoperations over **multiple** cycles, a Sequential Control replaces the Instruction Decoder
 - Input: Opcode, Status Bits, Control State
 - Output:
 - Control Word (Modified Datapath Control part)
 - Next State: Control Word (New Sequencing Control part)
 - Consists of:
 - 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 Word



Sequencing

Datapath

- Datapath part: field MM added, and fields DX, AX, and BX replace DA, AA, and BA, respectively
 - If the MSB of a field is 0, e.g., AX = 0XXX, then AA is 0 concatenated with SA (3bits) 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
 - E.g., 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	Address	No	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			$F \leftarrow A + \bar{B} + 1$	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							
						$F \leftarrow \bar{A}$	1011							
						$F \leftarrow B$	1100							
						$F \leftarrow sr B$	1101							
						$F \leftarrow sl B$	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 instr.	1	
	Branch	10			
	Jump	11			

ASM Charts for Sequential Control

- An instruction requires two steps:
 - *Instruction fetch* – obtaining an instruction from memory
 - *Instruction execution* – the execution of a sequence of microoperations to perform instruction processing
 - Due to the use of the IR, these two steps require a minimum of **two clock** cycles

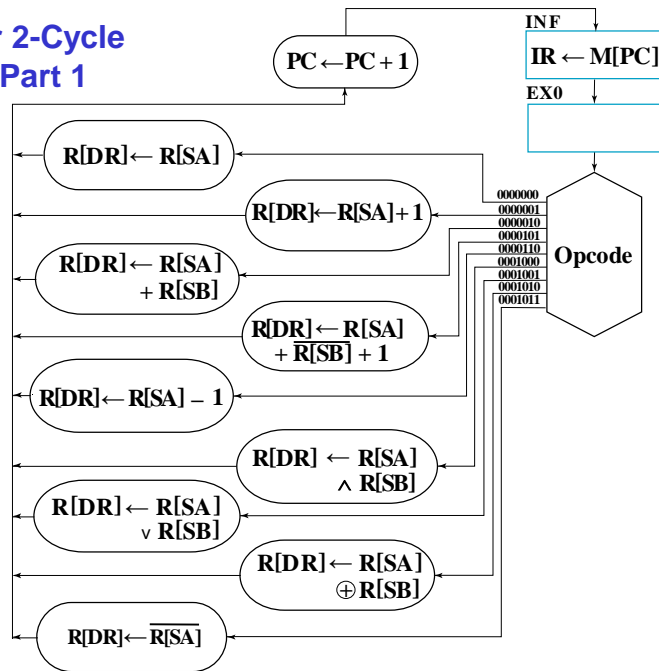
- ISA: Instruction Specifications and ASM charts for the instructions (that all require two clock cycles)
 - A vector decision box is used for the opcode
 - Scalar decision boxes are used for the status bits

ISA: Instruction Specifications (for reference)

Instruction Specifications for the Simple Computer - Part 1

Instruction	Opcode	Mnemonic	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

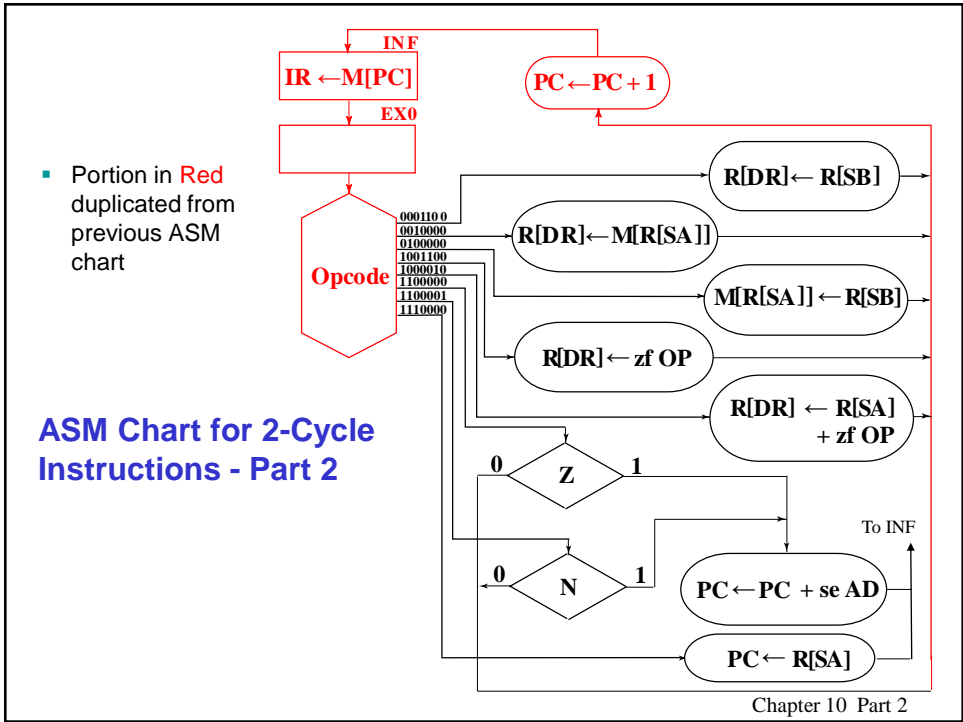
ASM Chart for 2-Cycle Instructions - Part 1



ISA: Instruction Specifications (for reference)

Instruction Specifications for the Simple Computer - Part 2

Instruction	Opcode	Mnemonic	Format	Description	Status Bits
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	$R[DR] \leftarrow R[SA] + zf OP$	
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 $(R[SA] = 0)$ $PC \leftarrow PC + se AD$	
Branch on Negative	1100001	BRN	RA,AD	if $(R[SA] < 0)$ $PC \leftarrow PC + se AD$	
Jump	1110000	JMP	RA	$PC \leftarrow R[SA]$	



State Table for 2-Cycle Instructions

State	Inputs		Next state	Outputs												Comments
	Opcode	VCNZ		I L	P S	DX	AX	BX	M B	FS	M D	R W	M W	M W		
INF	XXXXXX	XXXX	EX0	1	00	XXXX	XXXX	XXXX	X	XXXX	X	0	1	0	IR ← M[PC]	
EX0	0000000	XXXX	INF	0	01	0XXX	0XXX	XXXX	X	0000	0	1	X	0	MOVA R[DR] ← R[SA]*	
EX0	0000001	XXXX	INF	0	01	0XXX	0XXX	XXXX	X	0001	0	1	X	0	INC R[DR] ← R[SA] + 1*	
EX0	0000010	XXXX	INF	0	01	0XXX	0XXX	0XXX	0	0010	0	1	X	0	ADD R[DR] ← R[SA] + R[SB]*	
EX0	0000101	XXXX	INF	0	01	0XXX	0XXX	0XXX	0	0101	0	1	X	0	SUB R[DR] ← R[SA] + R[SB] + 1*	
EX0	0000110	XXXX	INF	0	01	0XXX	0XXX	XXXX	X	0110	0	1	X	0	DEC R[DR] ← R[SA] + (-1)*	
EX0	0001000	XXXX	INF	0	01	0XXX	0XXX	0XXX	0	1000	0	1	X	0	AND R[DR] ← R[SA] ∧ R[SB]*	
EX0	0001001	XXXX	INF	0	01	0XXX	0XXX	0XXX	0	1001	0	1	X	0	OR R[DR] ← R[SA] ∨ R[SB]*	
EX0	0001010	XXXX	INF	0	01	0XXX	0XXX	0XXX	0	1010	0	1	X	0	XOR R[DR] ← R[SA] ⊕ R[SB]*	
EX0	0001011	XXXX	INF	0	01	0XXX	0XXX	XXXX	X	1011	0	1	X	0	NOT R[DR] ← R[SA]*	
EX0	0001100	XXXX	INF	0	01	0XXX	0XXX	0XXX	0	1100	0	1	X	0	MOVB R[DR] ← R[SB]*	
EX0	0010000	XXXX	INF	0	01	0XXX	0XXX	XXXX	X	XXXX	1	1	0	0	LD R[DR] ← M[R[SA]]*	
EX0	0100000	XXXX	INF	0	01	0XXX	0XXX	0XXX	0	XXXX	X	0	0	1	ST M[R[SA]] ← R[SB]*	
EX0	1001100	XXXX	INF	0	01	0XXX	0XXX	XXXX	1	1100	0	1	0	0	LDI R[DR] ← zf OP*	
EX0	1000010	XXXX	INF	0	01	0XXX	0XXX	XXXX	1	0010	0	1	0	0	ADI R[DR] ← R[SA] + zf OP*	
EX0	1100000	XXX1	INF	0	10	XXXX	XXXX	XXXX	X	0000	X	0	0	0	BRZ PC ← PC + se AD	
EX0	1100000	XXX0	INF	0	01	XXXX	XXXX	XXXX	X	0000	X	0	0	0	BRZ PC ← PC + 1	
EX0	1100001	XX1X	INF	0	10	XXXX	XXXX	XXXX	X	0000	X	0	0	0	BRN PC ← PC + se AD	
EX0	1100001	XX0X	INF	0	01	XXXX	XXXX	XXXX	X	0000	X	0	0	0	BRN PC ← PC + 1	
EX0	1110000	XXXX	INF	0	11	XXXX	XXXX	XXXX	X	0000	X	0	0	0	JMP PC ← R[SA]	

* For this state and input combinations, PC ← PC+1 also occurs

3-Process ASM VHDL Code

```
entity controller is
  port ( opcode : in std_logic_vector(6 downto 0);
        reset, clk : in std_logic;
        zero, negative : in std_logic;
        IL, MB, MD, MM, RW, MW : out std_logic;
        PS : out std_logic_vector(1 downto 0);
        DX, AX, BX, FS : out std_logic_vector(3 downto 0);
  );
end controller;

architecture Behavioral of controller is
  type state_type is (RES, FTH, EX);
  signal cur_state, next_state : state_type;
begin
  state_register:process(clk, reset)
  begin
    if (reset='1') then
      cur_state<=RES;
    elsif (clk'event and clk='1') then
      cur_state<=next_state;
    end if;
  end process;
end Behavioral;
```

3-Process ASM VHDL Code

```
out_func: process (cur_state, opcode, zero, negative)
begin
  (IL,PS, MB, FS, MD, RW, MW, MM) <= std_logic_vector('0x'000');
  FS<="0000";
  case cur_state is
    when RES =>
      next_state <= FTH;
    when FTH =>
      -- set the control vector values
      next_state <= EXE;
    when EXE =>
      case opcode is
        when "0000000" +>

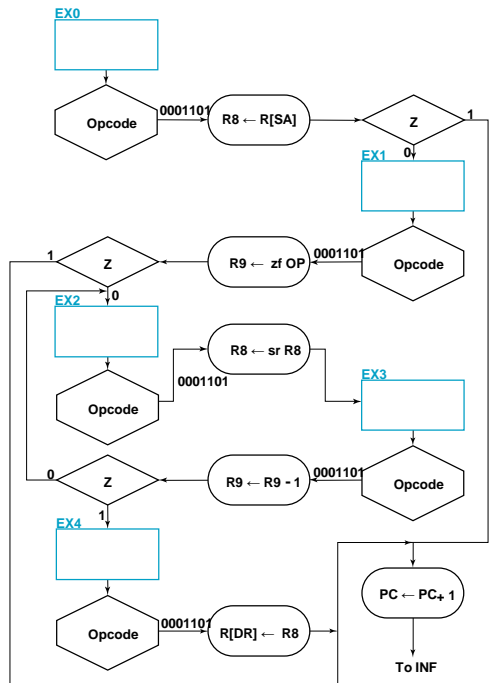
```

end process;

end Behavioral;

ASM Chart for Multiple Bits Right Shift

- R8 – used to perform shifts
- R9 – used to store and decrement shift count
- Zero test in EX1 is to determine if the shift amount is 0; if so, goes to state INF



State Table For Multiple Bits Right Shift

State	Inputs		Next state	Outputs											Comments	
	Opcode	VCNZ		I	L	PS	DX	AX	BX	MB	FS	MD	RW	MM		MW
EX0	0001101	XXX0	EX1	0	00	1000	0XXX	XXXX	X	0000	0	1	X	0	SRM	R8 ← R[SA], Z̄: → EX1
EX0	0001101	XXX1	INF	0	01	1000	0XXX	XXXX	X	0000	0	1	X	0	SRM	R8 ← R[SA], Z: → INF*
EX1	0001101	XXX0	EX2	0	00	1001	XXXX	XXXX	1	1100	0	1	X	0	SRM	R9 ← zf OP, Z̄: → EX2
EX1	0001101	XXX1	INF	0	01	1001	XXXX	XXXX	1	1100	0	1	X	0	SRM	R9 ← zf OP, Z: → INF*
EX2	0001101	XXXX	EX3	0	00	1000	XXXX	1000	0	1101	0	1	X	0	SRM	R8 ← sr R8, → EX3
EX3	0001101	XXX0	EX2	0	00	1001	1001	XXXX	X	0110	0	1	X	0	SRM	R9 ← R9 - 1, Z̄: → EX2
EX3	0001101	XXX1	EX4	0	00	1001	1001	XXXX	X	0110	0	1	X	0	SRM	R9 ← R9 - 1, Z: → EX4
EX4	0001101	XXXX	INF	0	01	0XXX	1000	XXXX	X	0000	0	1	X	0	SRM	R[DR] ← R8, → INF*

* For this state and input combinations, PC ← PC+1 also occurs

Summary

- Concept of Datapath for implementing computer microinstructions
- Control word provides a means of organizing the control of the microoperations
- Concept of ISA and instruction formats and operations of Simple Computer (SC)
- Single clock cycle vs. multiple cycle (instruction fetch + instruction execution)