EE-379 Embedded Systems and Applications

Computer Basics

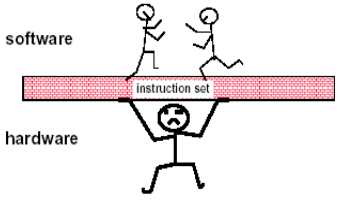
Cristinel Ababei
Department of Electrical Engineering, University at Buffalo
Spring 2013
Note: This course is offered as EE 459/500 in Spring 2013

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 – Pipelining
  - Single-Cycle Computer Issues
  - Pipelining concept
  - Pipelined design of Simple Computer

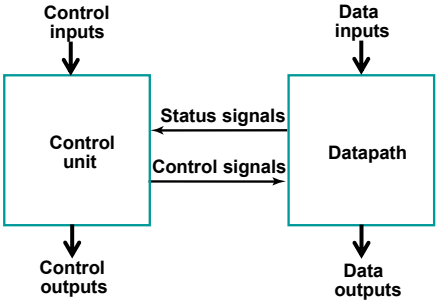
Introduction

- 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



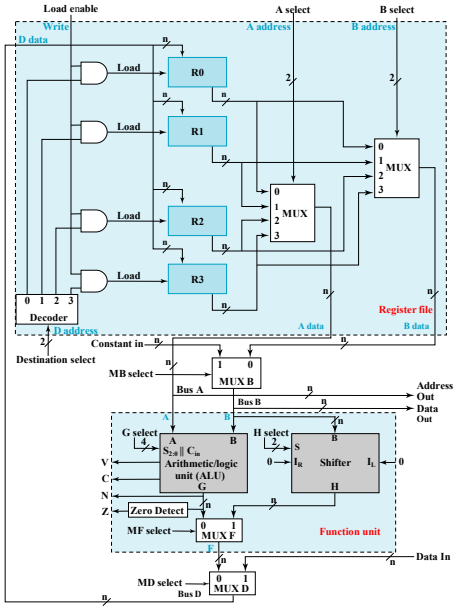
Introduction

- 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

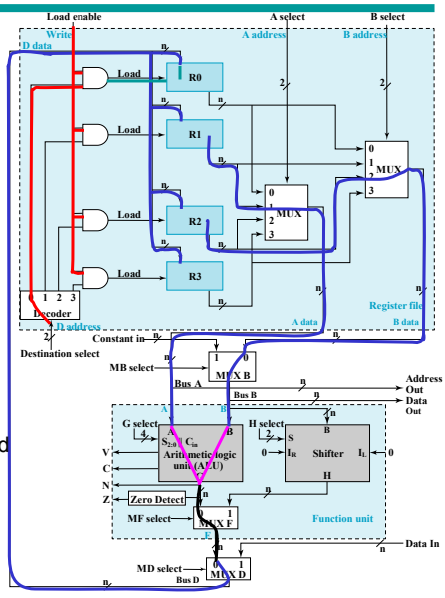
- Register file:
  - Four parallel-load regs
  - Two mux-based register selectors
  - Register destination decoder
- Microoperation implementation
  - Mux B for external constant input
  - Buses A and B with external address and data outputs
  - Function Unit:
    - ALU and Shifter with Mux F for output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



Datapath Example: Performing a Microoperation

Microoperation: R0 ← R1 + R2

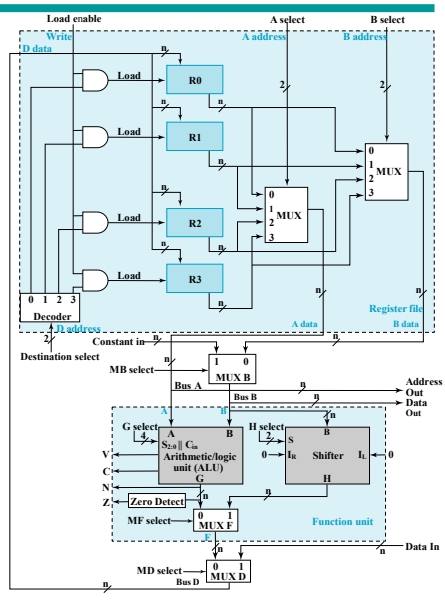
- Apply 01 to A select to place contents of R1 onto Bus A
- Apply 10 to B select to place contents of R2 onto B data and apply 0 to MB select to place B data on Bus B
- Apply 0010 to G select to perform addition G = Bus A + Bus B
- Apply 0 to MF select and 0 to MD select to place the value of G onto 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

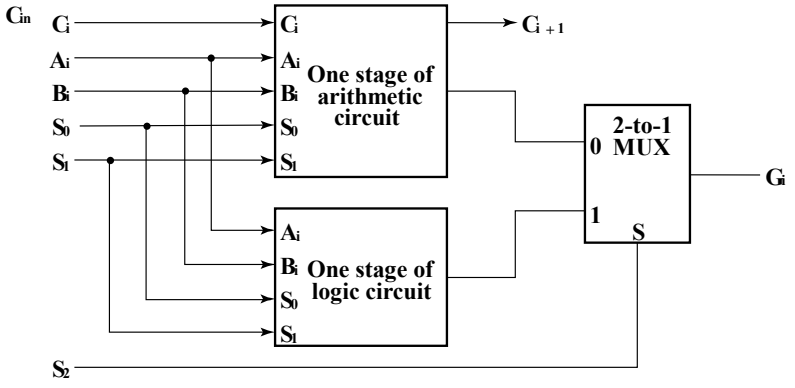


Overview

- 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 – Pipelining
  - Single-Cycle Computer Issues
  - Pipelining concept
  - Pipelined design of Simple Computer

Arithmetic Logic Unit (ALU)

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

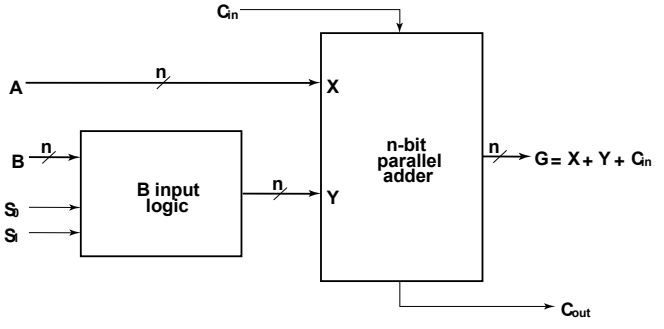


Arithmetic Circuit

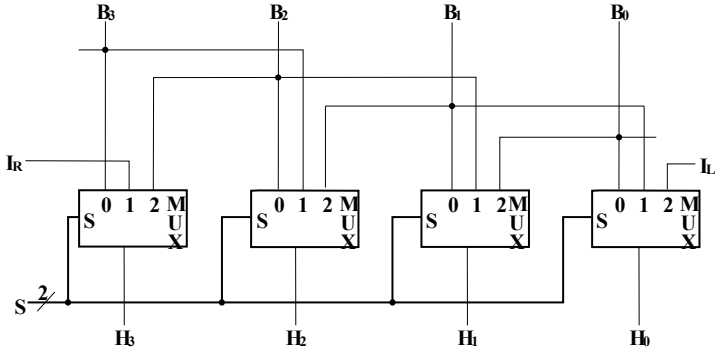
- Arithmetic circuit design
  - Decompose the arithmetic circuit into:
    - An n-bit parallel adder
    - A logic block that selects four choices for input B to the adder
- There are only four functions of B to select as Y in $G = A + Y + C_{in}$:

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

Arithmetic operations

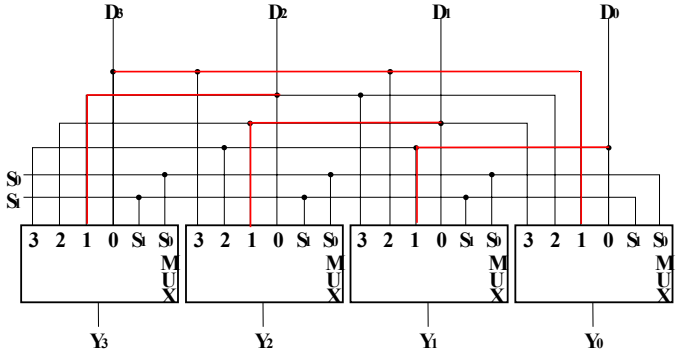


4-Bit Basic Left/Right Shifter



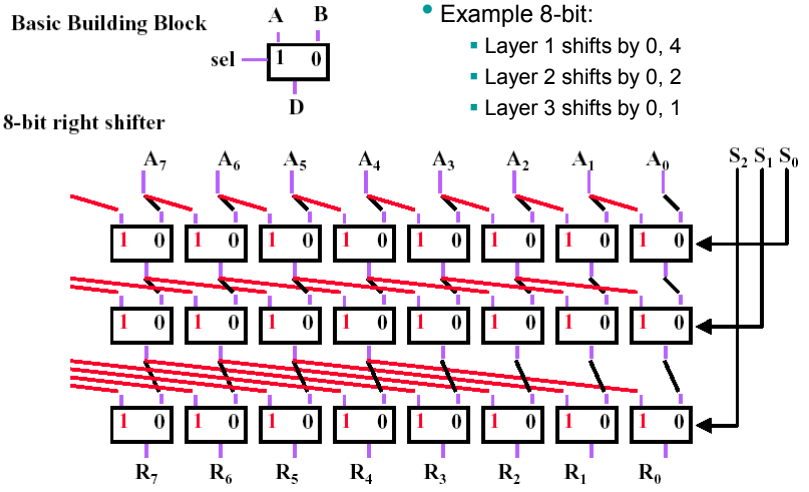
- Serial Inputs:
  - I_R for right shift
  - I_L for left shift
- Shift Functions:
  (S_1, S_0) = 00 Pass B unchanged
  01 Right shift
  10 Left shift
  11 Unused

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

Combinational Shifter from MUXes



- Example 8-bit:
  - Layer 1 shifts by 0, 4
  - Layer 2 shifts by 0, 2
  - Layer 3 shifts by 0, 1

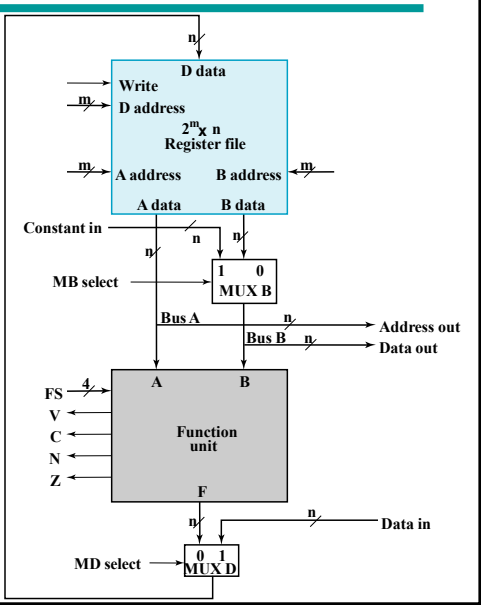
- Large barrel shifters can be constructed using:
  - Layers of multiplexers
  - 2-dimensional array circuits designed at the electronic level

Overview

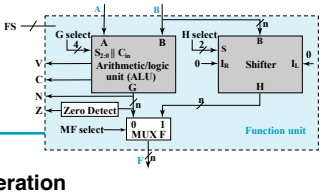
- Part 1 – Datapaths
  - Introduction
  - Datapath Example
    - Arithmetic Logic Unit (ALU)
    - Shifter
  - Datapath Representation and Control Word
- Part 2 – A Simple Computer
  - Instruction Set Architecture (ISA)
  - Single-Cycle Hardwired Control
- Part 3 – Multiple Cycle Hardwired Control
  - 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	$F \leftarrow A + \overline{B} + 1$
0100	0	0100	XX	$F \leftarrow A + \overline{B}$
0101	0	0101	XX	$F \leftarrow A + \overline{B} + 1$
0110	0	0110	XX	$F \leftarrow A - 1$
0111	0	0111	XX	$F \leftarrow$
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	$F \leftarrow$
1100	1	XXXX	00	$F \leftarrow$
1101	1	XXXX	01	$F \leftarrow B$
1110	1	XXXX	10	$F \leftarrow B$

Boolean Equations:

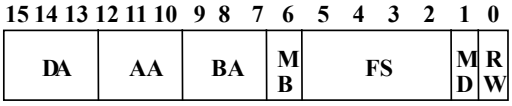
$MF_i = F_3 F_2$

$G_i = F_i$

$H_i = F_i$

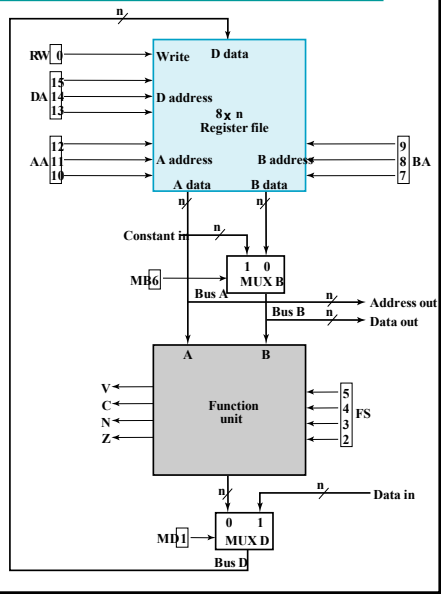
The Control Word

- 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



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

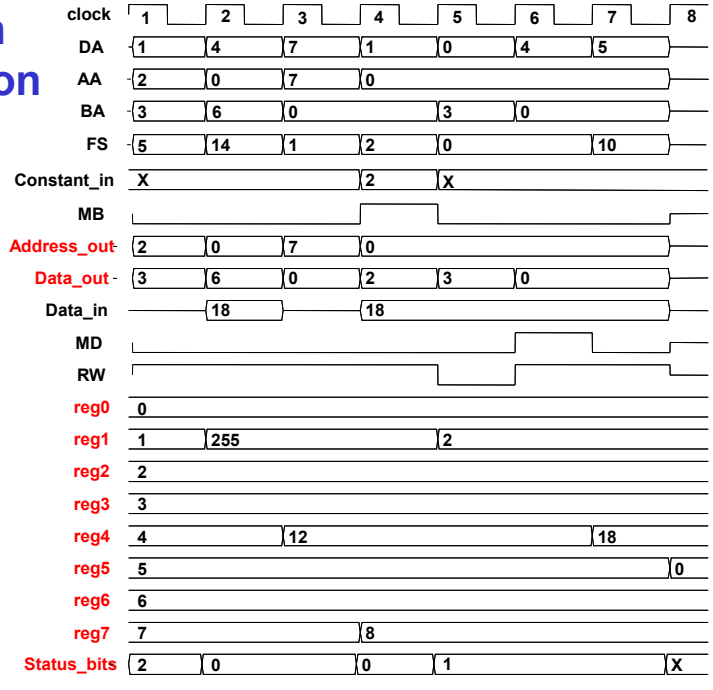
DA, AA, BA		MB		FS		MD		RW	
Function	**Code**	**Function**	**Code**	**Function**	**Code**	**Function**	**Code**	**Function**	**Code**
*R*0	000	Register	0	$F \leftarrow$	0000	Function	0	No write	0
*R*1	001	Constant	1	$F \leftarrow + 1$	0001	Data In	1	Write	1
*R*2	010			$F \leftarrow + B$	0010				
*R*3	011			$F \leftarrow + \overline{B} + 1$	0011				
*R*4	100			$F \leftarrow + \overline{B}$	0100				
*R*5	101			$F \leftarrow + \overline{B} + 1$	0101				
*R*6	110			$F \leftarrow - 1$	0110				
*R*7	111			$F \leftarrow$	0111				
				$F \leftarrow \wedge$	1000				
				$F \leftarrow \vee$	1001				
				$F \leftarrow \oplus$	1010				
				$F \leftarrow$	1011				
				$F \leftarrow$	1100				
				$F \leftarrow B$	1101				
				$F \leftarrow B$	1110				

Microoperations for the Datapath – Symbolic & Binary Representation

Micro-operation	DA	AA	BA	MB	FS	MD	RW
$R1 \leftarrow$ 2 $-R3$	$R1$	$R2$	$R3$	Register	$F = A + \overline{B} + 1$	Function	Write
$R4 \leftarrow$ R6	$R4$	—	$R6$	Register	$F =$ sl B	Function	Write
$R7 \leftarrow$ 7 + 1	$R7$	$R7$	—	Register	$F = A + 1$	Function	Write
$R1 \leftarrow$ 0 + 2	$R1$	$R0$	—	Constant	$F = A + B$	Function	Write
Data out $\leftarrow$ 3	——		$R3$	Register	—	—	No Write
$R4 \leftarrow$ ata in	$R4$	——		—	—	Data in	Write
$R5 \leftarrow$	$R5$	$R0$	$R0$	Register	$F = A \oplus$	Function	Write

Micro-operation	DA	AA	BA	MB	FS	MD	RW
$R1 \leftarrow$ 2 $-R3$	001	010	011	0	0101	0	1
$R4 \leftarrow$ R6	100	XXX	110	0	1110	0	1
$R7 \leftarrow$ 7 + 1	111	111	XXX	0	0001	0	1
$R1 \leftarrow$ 0 + 2	001	000	XXX	1	0010	0	1
Data out $\leftarrow$ 3	XXX	XXX	011	0	XXXX	X	0
$R4 \leftarrow$ ata in	100	XXX	XXX	X	XXXX	1	1
$R5 \leftarrow$	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 – Pipelining
  - Single-Cycle Computer Issues
  - Pipelining concept
  - Pipelined design of Simple Computer

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

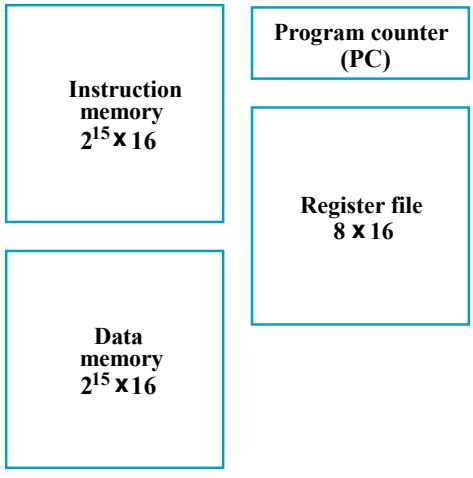
- Instructions are stored in RAM or ROM as a *program,* the addresses for instructions are provided by a *program counter (PC)*
  - Count up or load a new address
  - The PC and associated control logic are part of the Control Unit
- A typical instruction specifies:
  - Operands to use
  - Operation 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

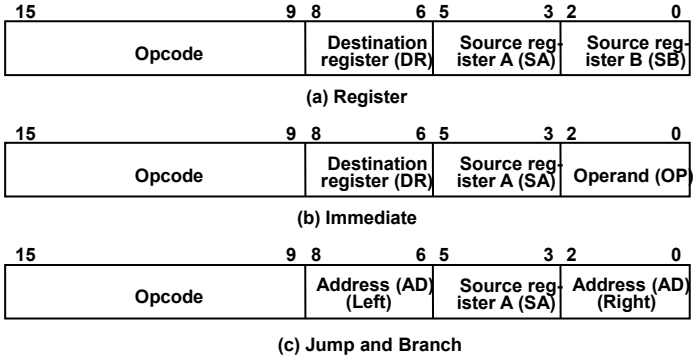
ISA: Storage Resources

- "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



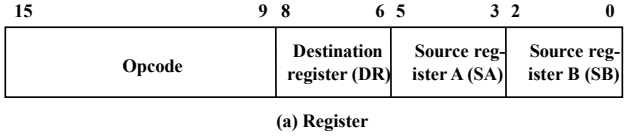
ISA: Instruction Formats

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



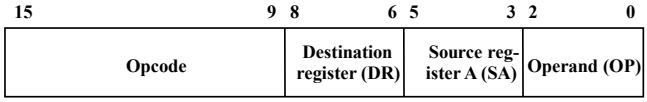
- All formats contain an Opcode field in bits 9 through 15.
  - The Opcode specifies the operation to be performed

ISA: Instruction Format - Register



- This format supports:
  - R1 ← R2 + R3
  - R1 ← 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?

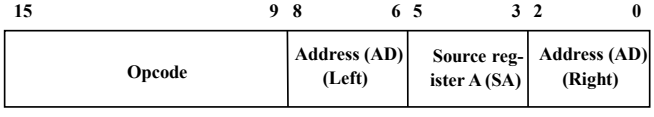
ISA: Instruction Format - Immediate



(b) Immediate

- This format supports:
  - R1 ← 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	0000000	MOVA	RD,RA	R[DR] ← R[SA]	N, Z
Increment	0000001	INC	RD,RA	R[DR] ← R[SA] + 1	N, Z
Add	0000010	ADD	RD,RA,RB	R[DR] ← R[SA] + R[SB]	N, Z
Subtract	0000101	SUB	RD,RA,RB	R[DR] ← R[SA] − R[SB]	N, Z
Decrement	0000110	DEC	RD,RA	R[DR] ← R[SA] − 1	N, Z
AND	0001000	AND	RD,RA,RB	R[DR] ← R[SA] ∧ R[SB]	N, Z
OR	0001001	OR	RD,RA,RB	R[DR] ← R[SA] ∨ R[SB]	N, Z
Exclusive OR	0001010	XOR	RD,RA,RB	R[DR] ← $\overline{R[SA]}$ ⊕ R[SB]	N, Z
NOT	0001011	NOT	RD,RA	R[DR] ← $\overline{R[SA]}$	N, Z
Move B	0001100	MOVB	RD,RB	R[DR] ← R[SB]	
Shift Right	0001101	SHR	RD,RB	R[DR] ← sr R[SB]	
Shift Left	0001110	SHL	RD,RB	R[DR] ← sl R[SB]	
Load Immediate	1001100	LDI	RD, OP	R[DR] ← zf OP	
Add Immediate	1000010	ADI	RD,RA,OP	R[DR] ← R[SA] + zf OP	
Load	0010000	LD	RD,RA	R[DR] ← M[R[SA]]	
Store	0100000	ST	RA,RB	M[R[SA]]← R[SB]	
Branch on Zero	1100000	BRZ	RA,AD	if (R[SA] = 0) PC ← PC + se AD	
Branch on Negative	1100001	BRN	RA,AD	if (R[SA] < 0) PC ← PC + se AD	
Jump	1110000	JMP	RA	PC ← R[SA]	

ISA: Example Instructions and Data in Memory

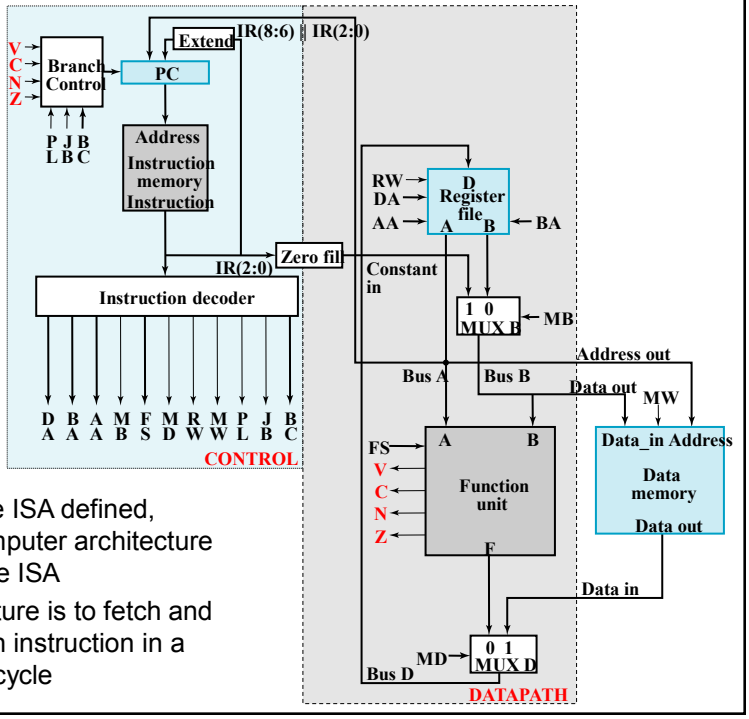
Memory Representation of Instruction and Data

Decimal Address	Memory Contents	Decimal Opcode	Other Field	Operation
25	0000101 001 010 011	5 (Subtract)	DR:1, SA:2, SB:3	R1 ← R2 − R3
35	0100000 000 100 101	32 (Store)	SA:4, SB:5	M[R4] ← R5
45	1000010 010 111 011	66 (Add Immediate)	DR: 2, SA :7, OP :3	R2 ← R7 + 3
55	1100000 101 110 100	96 (Branch on Zero)	AD: 44, SA:6	If R6 = 0, PC ← 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 – Pipelining
  - Single-Cycle Computer Issues
  - Pipelining concept
  - Pipelined design of Simple Computer

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

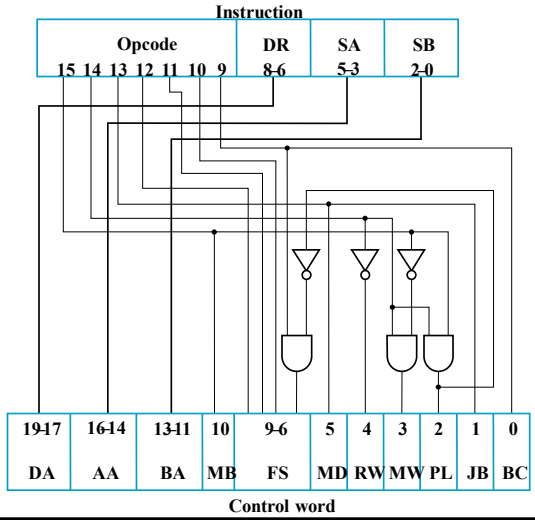
	Instruction Bits				Control Word Bits						
Instruction Function Type	**15**	**14**	**13**	**9**	**MB**	**MD**	**RW**	**MW**	**PL**	**JB**	**BC**
1. Function unit operations using registers	0	0	0	X	0	0	1	0	0	X	X
2. Memory read	0	0	1	X	0	1	1	0	0	X	X
3. Memory write	0	1	0	X	0	X	0	1	0	X	X
4. Function unit 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 (N)	1	1	0	1	X	X	0	0	1	0	1
7. Unconditional Jump	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



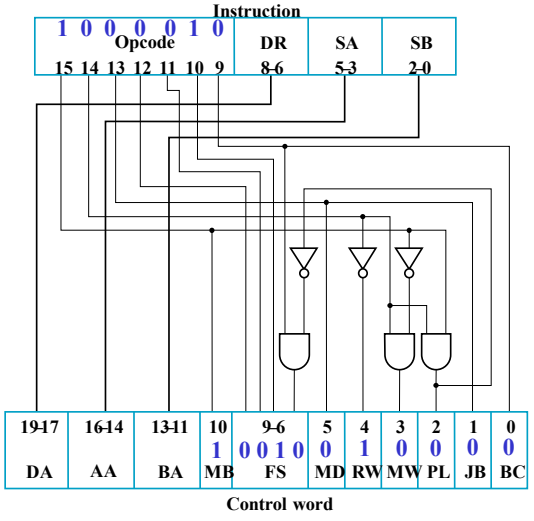
Example Instruction Execution

Six Instructions for the Single-Cycle Computer

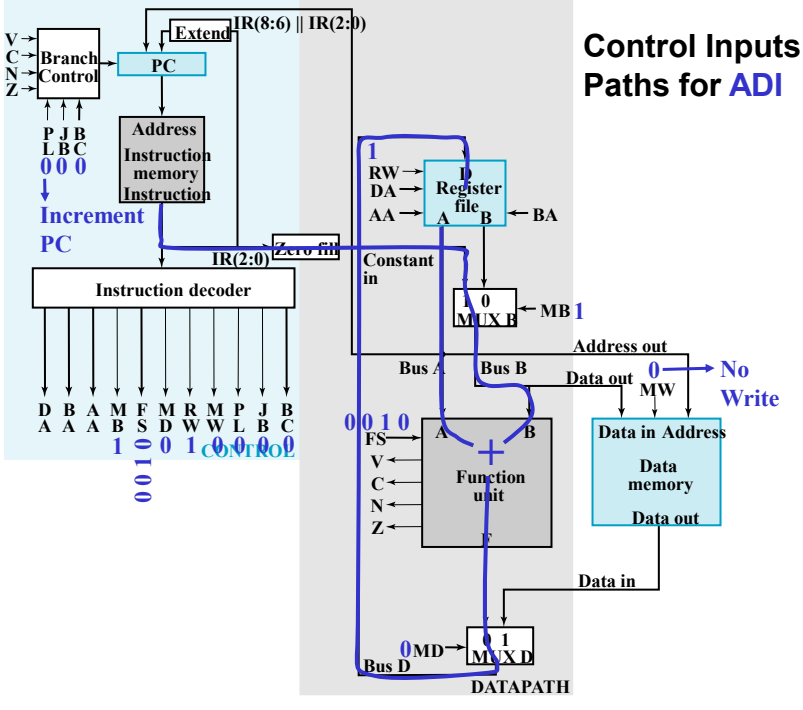
Operation code	Symbolic Name	Format	Description	Function	MB	MD	RW	MW	PL	JB	BC
1000 010	ADI	Immediate	Add immediate operand	$R[DR] \leftarrow R[SA] + zf\,I(2:0)$	1	0	1	0	0	0	0
0010 000	LD	Register	Load memory content into register	$R[DR] \leftarrow M[R[SA]]$	0	1	1	0	0	1	0
0100 000	ST	Register	Store register 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

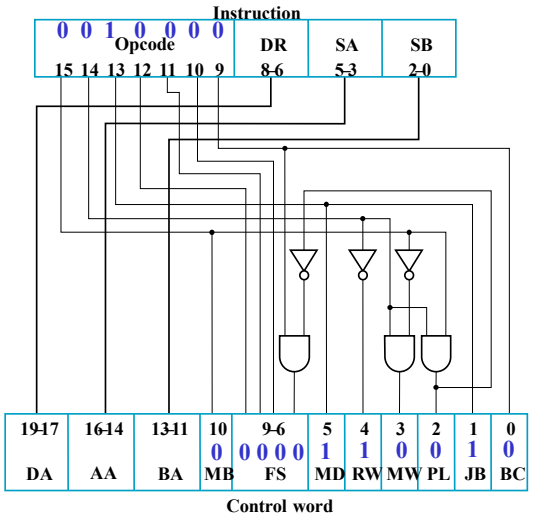
Decoding for ADI

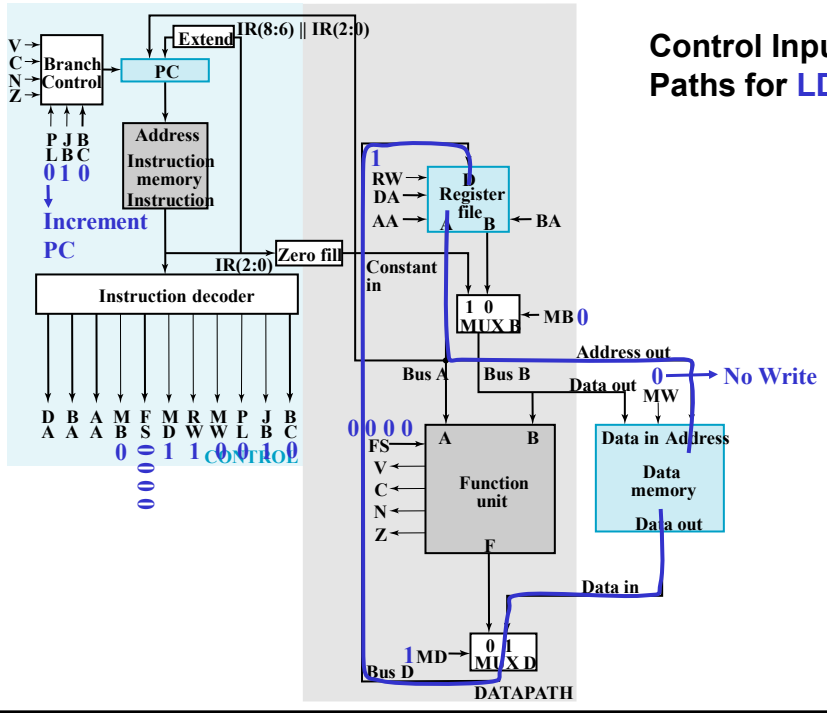


Control Inputs and Paths for ADI

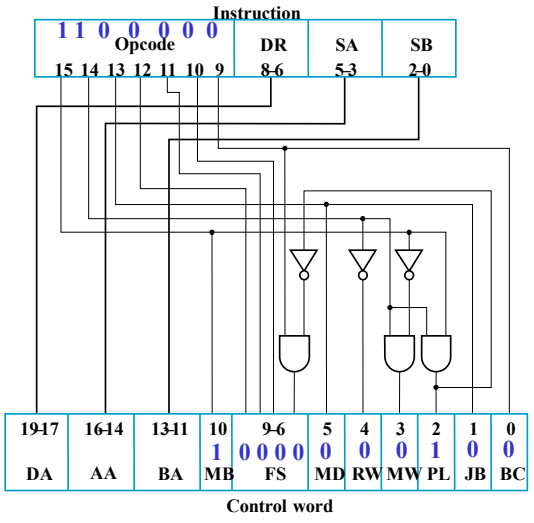


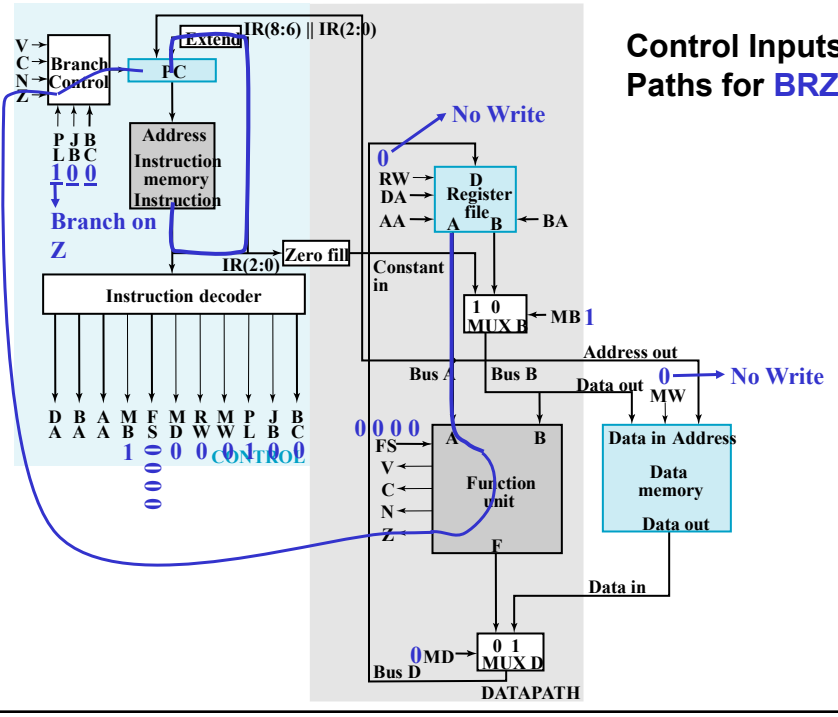
Decoding for LD





Decoding for BRZ





Control Inputs and Paths for BRZ

Overview

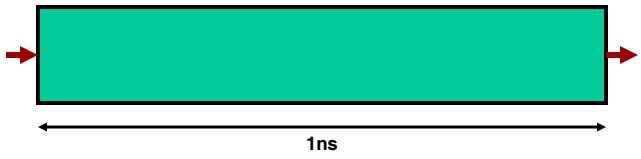
- Part 1 – Datapaths
  - Introduction
  - Datapath Example
    - Arithmetic Logic Unit (ALU)
    - Shifter
  - Datapath Representation and Control Word
- Part 2 – A Simple Computer
  - Instruction Set Architecture (ISA)
  - Single-Cycle Hardwired Control
- Part 3 – Pipelining
  - Single-Cycle Computer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

Single-Cycle Computer Issues

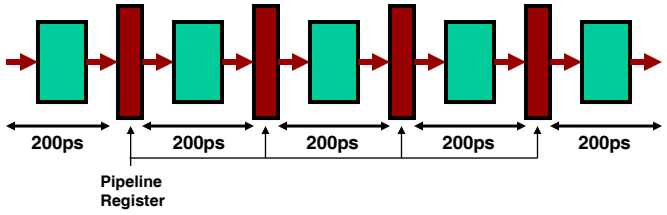
- Shortcoming of Single Cycle Design
  - Complexity of instructions executable in a single cycle is limited
  - Accessing both an instruction and data from a simple single memory impossible
  - A long worst case delay path limits clock frequency and the rate of performing instructions
- Handling of Shortcomings
  - The first two shortcomings can be handled by the **multiple-cycle** computer (not covered in these slides)
  - **The third shortcoming is dealt with by using a technique called pipelining**

Pipelining a digital system

- Key idea: break big computation into pieces

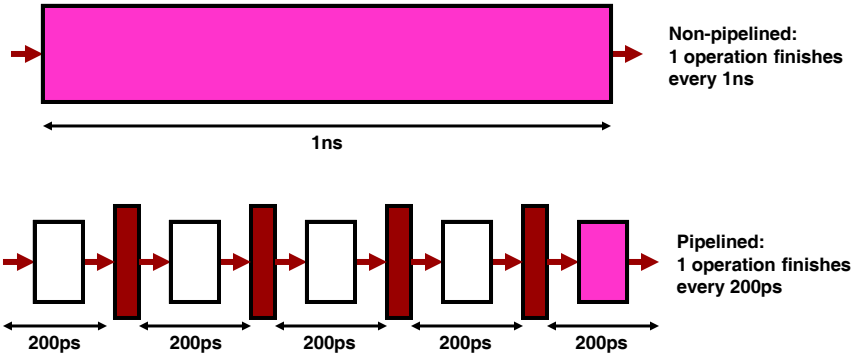


- Separate each piece with a pipeline register



Pipelining a digital system

- Why do this? Because it's faster for repeated computations



Pipelining

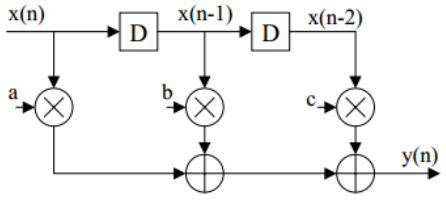
- Pipelining increases **throughput**, but not total computation time of a task
  - Answer available every 200ps, BUT
  - A single computation still takes 1ns
- Limitations:
  - Computations must be divisible into stage size
  - Pipeline registers add overhead

Pipelining

- **Pipelining transformation** leads to a reduction in the critical path, which can be exploited to increase the clock speed or to reduce power consumption at same speed.
- In parallel processing, multiple outputs are computed in parallel in a clock period. Therefore, the **effective clock speed** is increased by the level of parallelism.

Example: 3-tap FIR digital filter

Example 1: Consider a 3-tap FIR filter: y(n)=ax(n)+bx(n-1)+cx(n-2)



T_M : *multiplica tion – time*

T_A : *Addition – time*

– The critical path (or the minimum time required for processing a new sample) is limited by 1 multiply and 2 add times. Thus, the "sample period" (or the "sample frequency") is given by:

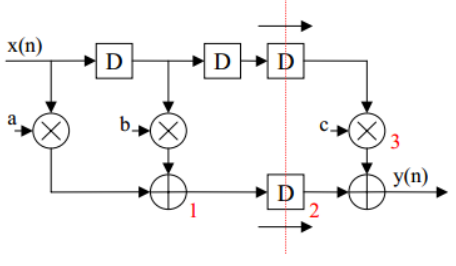
$$T_{sample} \geq T_M + 2T_A$$

$$f_{sample} \leq \frac{1}{T_M + 2T_A}$$

Example: 3-tap FIR digital filter

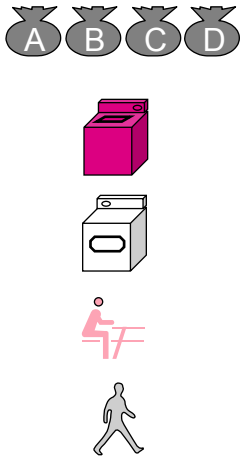
- The pipelined implementation: By introducing 2 additional latches in Example 1, the critical path is reduced from T_M+2T_A to T_M+T_A.The schedule of events for this pipelined system is shown in the following table. You can see that, at any time, 2 consecutive outputs are computed in an interleaved manner.

Clock	Input	Node 1	Node 2	Node 3	Output
0	x(0)	ax(0)+bx(-1)	—	—	—
1	x(1)	ax(1)+bx(0)	ax(0)+bx(-1)	cx(-2)	y(0)
2	x(2)	ax(2)+bx(1)	ax(1)+bx(0)	cx(-1)	y(1)
3	x(3)	ax(3)+bx(2)	ax(2)+bx(1)	cx(0)	y(2)

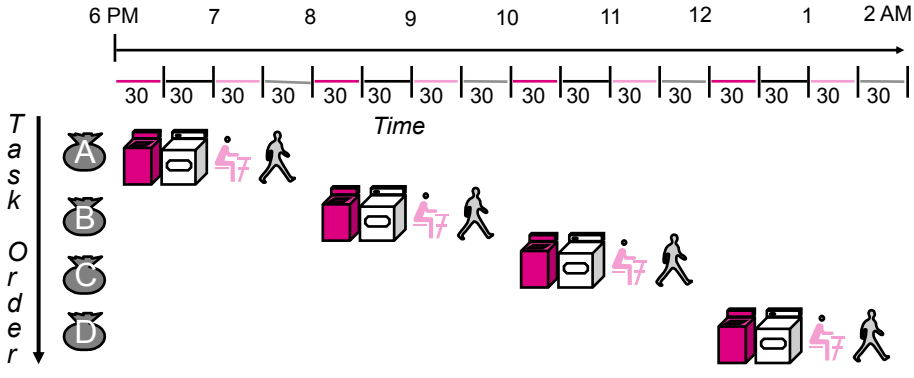


The Laundry Analogy

- Ann, Brian, Cathy, Dave each have one load of clothes to wash, dry, and fold
- Washer takes 30 minutes
- Dryer takes 30 minutes
- “Folder” takes 30 minutes
- “Stasher” takes 30 minutes to put clothes into drawers

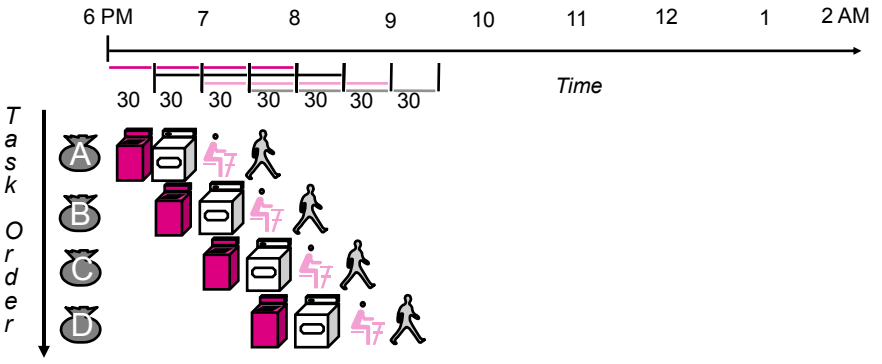


If we do laundry sequentially...



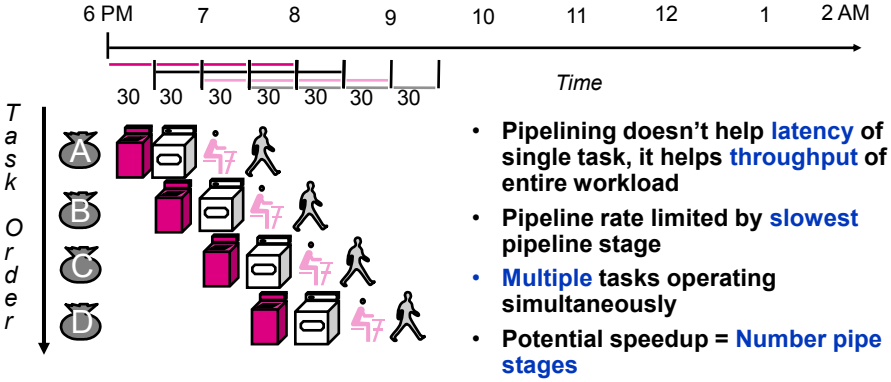
- Time Required: 8 hours for 4 loads

To Pipeline, We Overlap Tasks



- Time Required: 3.5 Hours for 4 Loads

To Pipeline, We Overlap Tasks



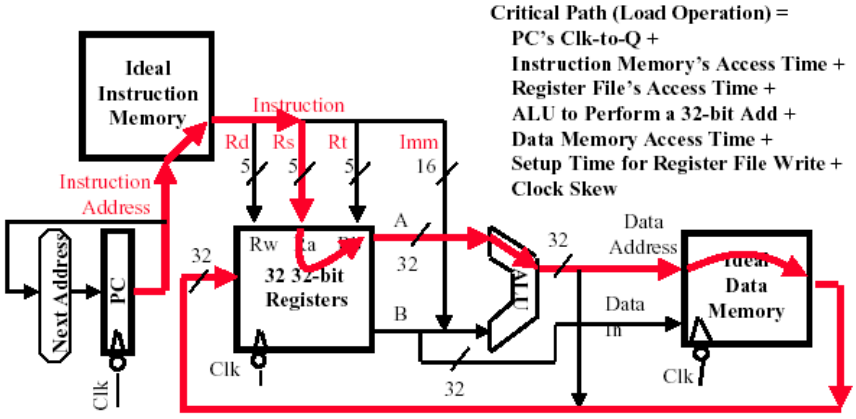
- **Pipelining doesn't help latency of single task, it helps throughput of entire workload**
- **Pipeline rate limited by slowest pipeline stage**
- **Multiple tasks operating simultaneously**
- **Potential speedup = Number pipe stages**
- **Unbalanced lengths of pipe stages reduces speedup**
- **Time to "fill" pipeline and time to "drain" it reduces speedup**

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 – Pipelining
  - Single-Cycle Computer Issues
  - Pipelining concept
  - Pipelined design of Simple Computer

Abstract View of Critical Path

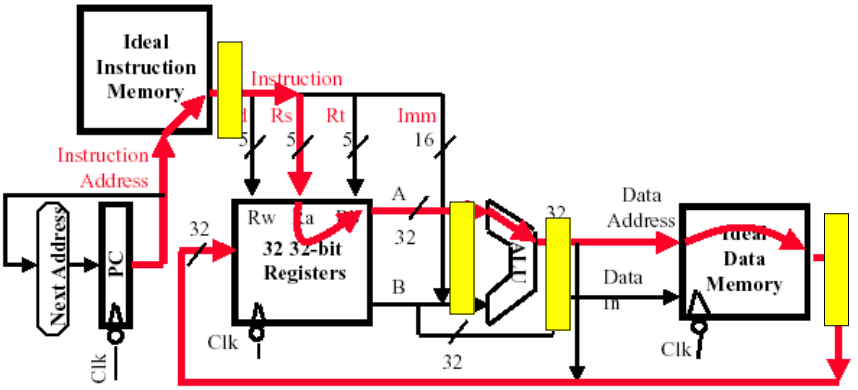
- Register file and ideal memory:
  - The CLK input is a factor ONLY during write operation
  - During read operation, behave as combinational logic:
    - Address valid => Output valid after "access time."



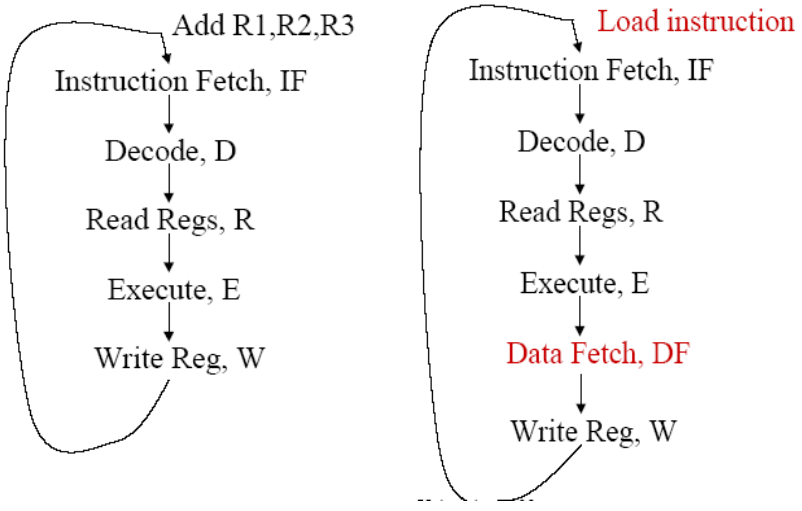
Critical Path (Load Operation) =
PC's Clk-to-Q +
Instruction Memory's Access Time +
Register File's Access Time +
ALU to Perform a 32-bit Add +
Data Memory Access Time +
Setup Time for Register File Write +
Clock Skew

Pipelined critical path

- Critical path is longest path between stage registers



Steps in Instruction Processing



Un-pipelined (Non-overlapped) Implementation

- Consider loads with DF stage

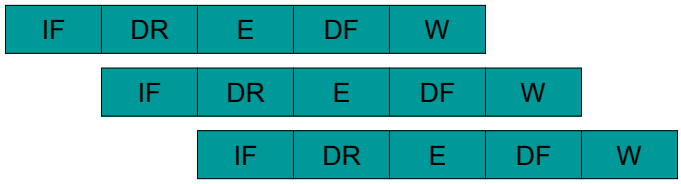
	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
I1	IF	DR	E	DF	W					
I2						IF	DR	E	DF	W
I3										
...										

Pipelined Implementation

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
I1	IF	DR	E	DF	W					
I2		IF	DR	E	DF	W				
I3			IF	DR	E	DF	W			
I4				IF	DR	E	DF	W		
I5					IF	DR	E	DF	W	
I6						IF	DR	E	DF	W

5-stage Pipeline

- CPU stages
  - IF: Instruction fetch
  - DR: Instruction decode & Register read
  - E: Execute
  - DF: Data fetch (Memory load/store)
  - W: Write Back Registers
- Another set of mnemonic names
  - IF, ID, E, MEM, WB



Computer Pipelines

- Execute billions of instruction, so **throughput** is what matters
- Throughput versus latency
  - \+ Throughput increases
  - \- Latency for a single instruction increases
    - May have to wait longer for single instruction to complete
- Allows much faster clock cycle
- RISC pipeline architecture features:
  - All instructions same length
  - Registers located in same place in instruction format
  - Memory operands only in loads and stores

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)
- Pipelining

For Adventurers

- I have a VHDL implementation of the Single Cycle Computer (SCC) described in this presentation. It's verified via simulation and actual hardware implementation on an FPGA (Spartan-6)
- If you are interested in studying it via simulation:
  - Download and install the free Aldec-HDL simulator as described here:
    - http://dejazzer.com/ee478/labs/lab1_aldec_tutorial.pdf
  - Then, simulate the SCC using the VHDL source code available for download here:
    - http://dejazzer.com/ee478/labs/lab11_files.zip