Register Transfer Level Design Introduction-Summary

Zoran Salcic

University of Auckland, 2024

Outline

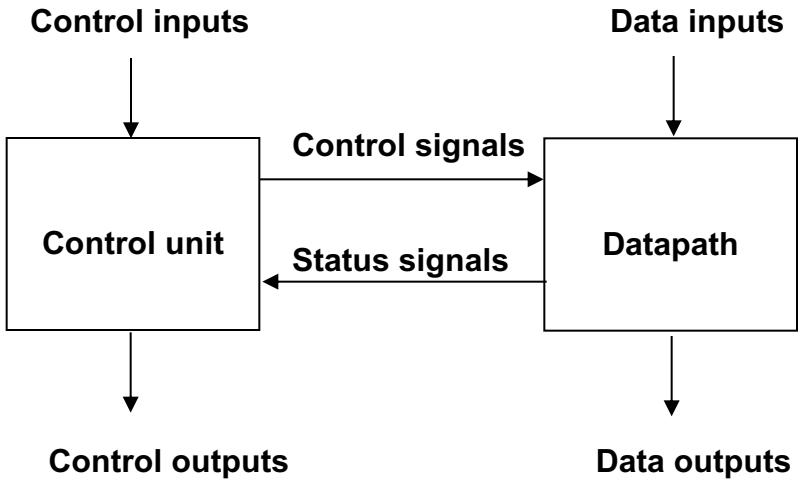
- **Simple and general datapaths**
- **Control units architectures**
- **Register transfer design model**
- Finite state machines with datapath (FSMD)
- Algorithmic state machine (ASM) charts as graphical presentation of FSMDs
- Synthesis from ASMs
- Variable sharing
- Operation sharing
- Chaining and multicycling
- Pipelining (operation, datapath, control unit)

Register transfer level (RTL) design model

The general design model for RTL design consists of control unit and datapath

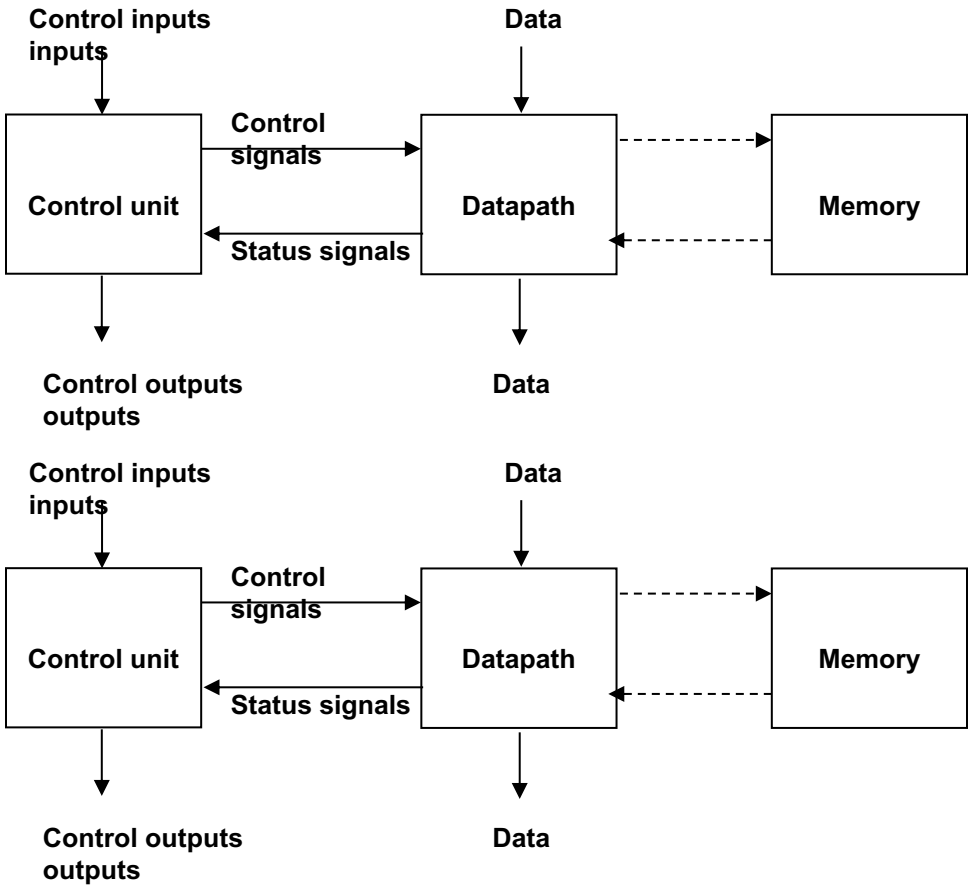
Two types of I/O ports:

- data ports (inputs and outputs) to exchange data with the outside environment
- control ports to control the operations performed by the datapath and receive information about the status of selected registers in the datapath



Register transfer level (RTL) design model

The datapath often receives operands from memory and writes results of operations to memory
Often cascaded – almost never nested



Register transfer level (RTL) design model

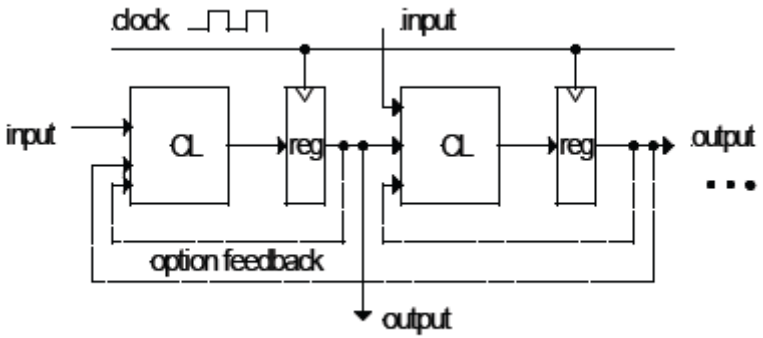
- The datapath takes the operands from storage units, performs computation in the combinatorial units, and returns the results to storage units during each state.
- Control units controls selection of data sources, operations and data destinations by setting proper values of datapath control signals.
- The datapath indicates that data value has been properly stored in a particular storage unit or when a particular relation between data values in the datapath is satisfied.
- Control unit operates on a set of input control signals: external control inputs and internal status signals
- Control unit produces two types of output signals:
  - external signals used to indicate to the environment that the circuit has reached a certain state or finished a particular operation and
  - datapath control signals that select the operation for each component in the datapath

Datapaths

- Used in all standard processor and ASIC implementations to perform complex numerical computation and manipulations
- Consist of temporary storage and processing elements (functional units)
- The variable values and constants are stored in storage components (registers and memories)
- They are fetched from storage components after the active (e.g. rising) edge of the clock signal
- They are transformed in combinational components (processing elements) between two active edges of the clock
- The results are stored back into the storage components at the next active edge of the clock signal

Datapaths

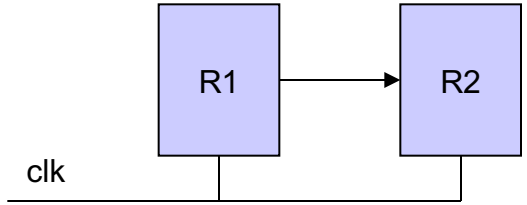
- Data stored in registers and processed by processing elements
- Movement of data – register transfer operations (RTOs)
- RTOs are described by 3 basic components:
  - Set of registers in the system
  - Operations performed on data
  - Control that supervises the sequence of operations



Register Transfers

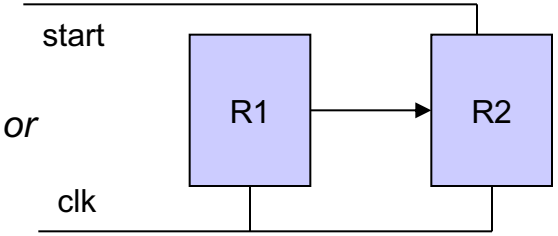
- Registers have capabilities to perform basic operations (load, count, add, subtract, shift)
- Registers perform *microoperations* (usually in parallel on multiple bits)
- Result of operation can be stored in register or transferred to another register
- RTL is Register Transfer Language – represents registers and operations on their content (implies circuitry to perform transfer)
- All transfers occur in response to clock transitions

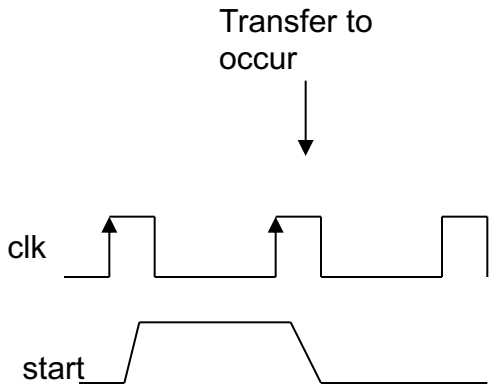
Simple transfer R2 ← R1



Conditional transfer
if (start=1) then (R2 ← R1) or

start: R2 ← R1





Register Transfers Language

- Allows multiple simultaneous transfers: *R1 ← R2, R2 ←R1*
- Memory address specified by square brackets: *DR ←M[AR]*
- RTL vs VHDL vs Verilog

Operation	**RTL**	**VHDL**	**Verilog**
Combinational assignment	=	<=	assign =
Register transfer	←	<=	<=
Addition	+	+	+
Subtraction	-	-	-
Bitwise AND	∧	and	&
Bitwise OR	∨	or	|
Bitwise XOR	⊕	xor	^
Bitwise NOT	¯	not	~
Shift left (logical)	sl	sll	<<
Shift right (logical)	sr	srl	>>
Vectors/Registers	A(3:0)	A(3 downto 0)	A[3:0]
Concatenation	||	&	{ , }

Register Transfers - Microoperations

- Transfers

Symbol	Description	Examples
Letters (and numerals)	Denotes a register	AR, $R2$, DR, IR
Parentheses	Denotes a part of a register	$R2(1)$, $R2(7:0)$, $AR(L)$
Arrow	Denotes transfer of data	$R1 \leftarrow R2$
Comma	Separates simultaneous transfers	$R1 \leftarrow R2$, $R2 \leftarrow R1$
Square brackets	Specifies an address for memory	$DR \leftarrow M[AR]$

Register Transfers - Microoperations

- Arithmetic

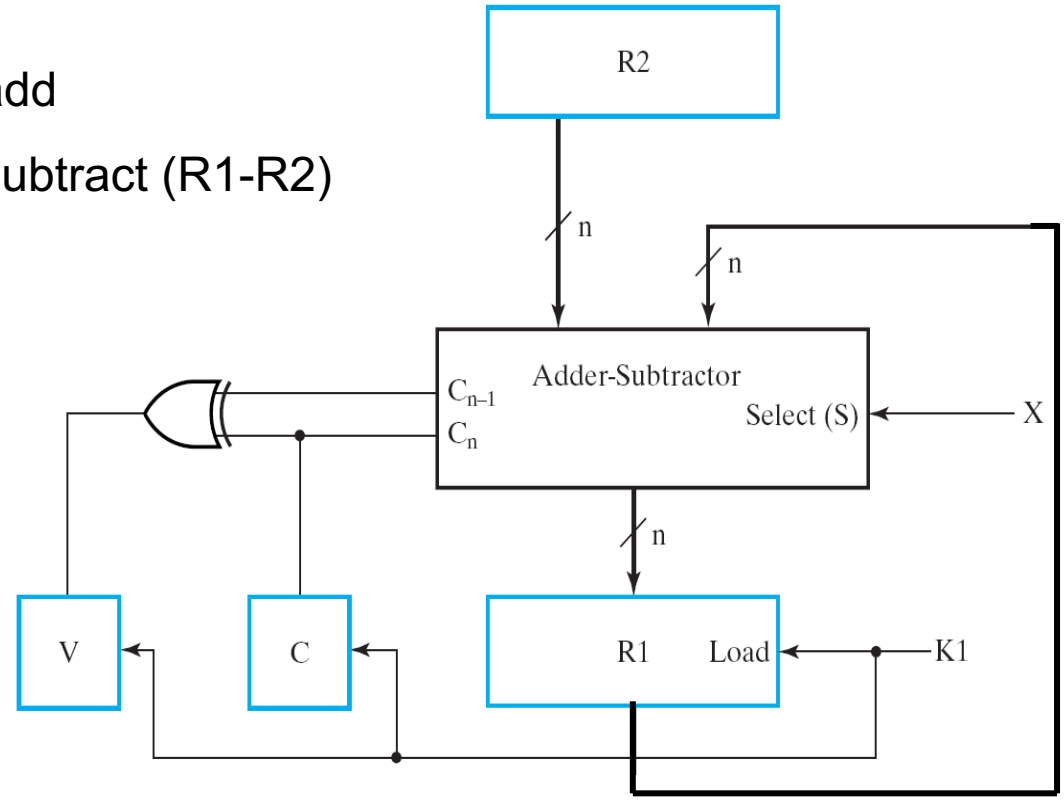
Symbolic designation	Description
$R0 \leftarrow R1 + R2$	Contents of $R1$ plus $R2$ transferred to $R0$
$R2 \leftarrow \overline{R2}$	Complement of the contents of $R2$ (1's complement)
$R2 \leftarrow \overline{R2} + 1$	2's complement of the contents of $R2$
$R0 \leftarrow R1 + \overline{R2} + 1$	$R1$ plus 2's complement of $R2$ transferred to $R0$ (subtraction)
$R1 \leftarrow R1 + 1$	Increment the contents of $R1$ (count up)
$R1 \leftarrow R1 - 1$	Decrement the contents of $R1$ (count down)

Register Transfers - Microoperations

- Arithmetic – example

$\overline{X}k1: R1 \leftarrow R1 + R2$ add

$Xk1: R1 \leftarrow R1 + \overline{R2} + 1$ subtract (R1-R2)



Register Transfers - Microoperations

- Logic

Symbolic designation	Description
$R0 \leftarrow \overline{R1}$	Logical bitwise NOT (1's complement)
$R0 \leftarrow R1 \wedge R2$	Logical bitwise AND (clears bits)
$R0 \leftarrow R1 \vee R2$	Logical bitwise OR (sets bits)
$R0 \leftarrow R1 \oplus R2$	Logical bitwise XOR (complements bits)

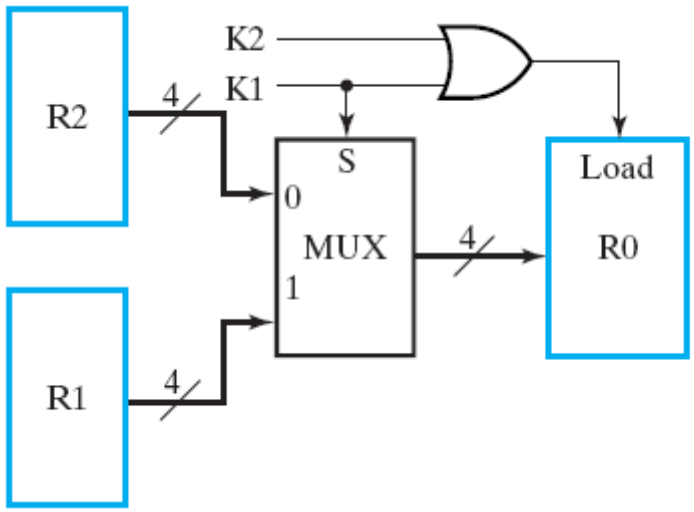
- Shift

		Eight-bit examples	
Type	Symbolic designation	Source *R2*	After shift: Destination *R1*
shift left	R1← sl R2	10011110	00111100
shift right	R1← sr R2	11100101	01110010

Register Transfers – Using Multiplexers

if (k1=1) then (R0←R1) else if (k2=1) then (R0←R2)

$$k1 : R0 \leftarrow R1, \overline{k1}k2 : R0 \leftarrow R2$$



Register Transfers – Multiplexer and Bus-based Transfers

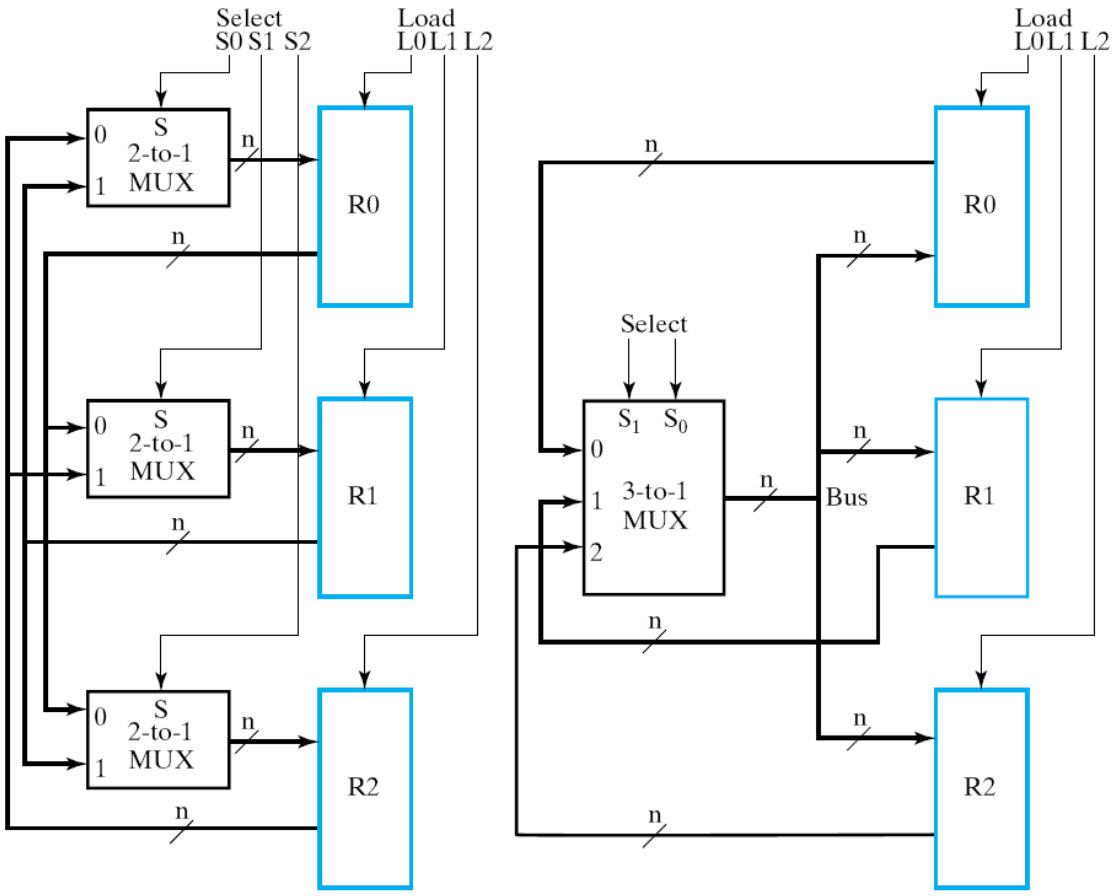
Transfer examples:

R0←R1

R1←R0, R2←R0

Is this possible:

R1←R0, R0←R1

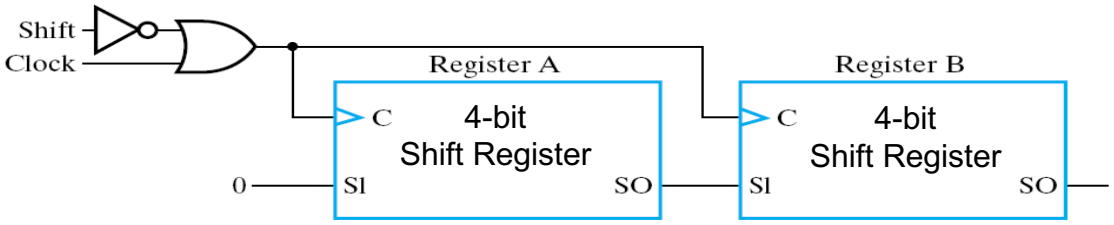


(a) Dedicated multiplexers

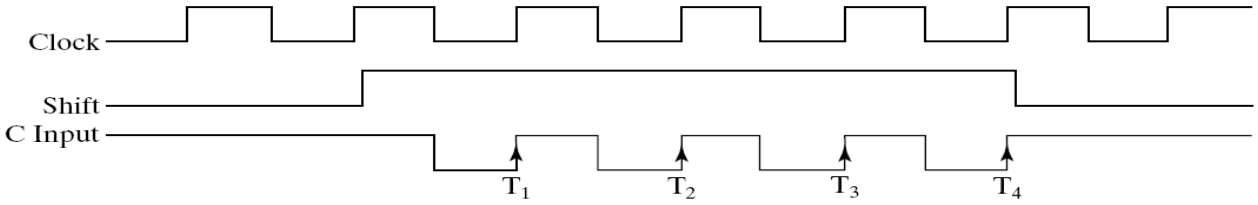
(b) Single Bus

Serial Transfers and Microoperations

Timing pulse	Shift Register A				Shift Register B			
Initial value	1	0	1	1	0	0	1	0
After T_1	0	1	0	1	1	0	0	1
After T_2	0	0	1	0	1	1	0	0
After T_3	0	0	0	1	0	1	1	0
After T_4	0	0	0	0	1	0	1	1

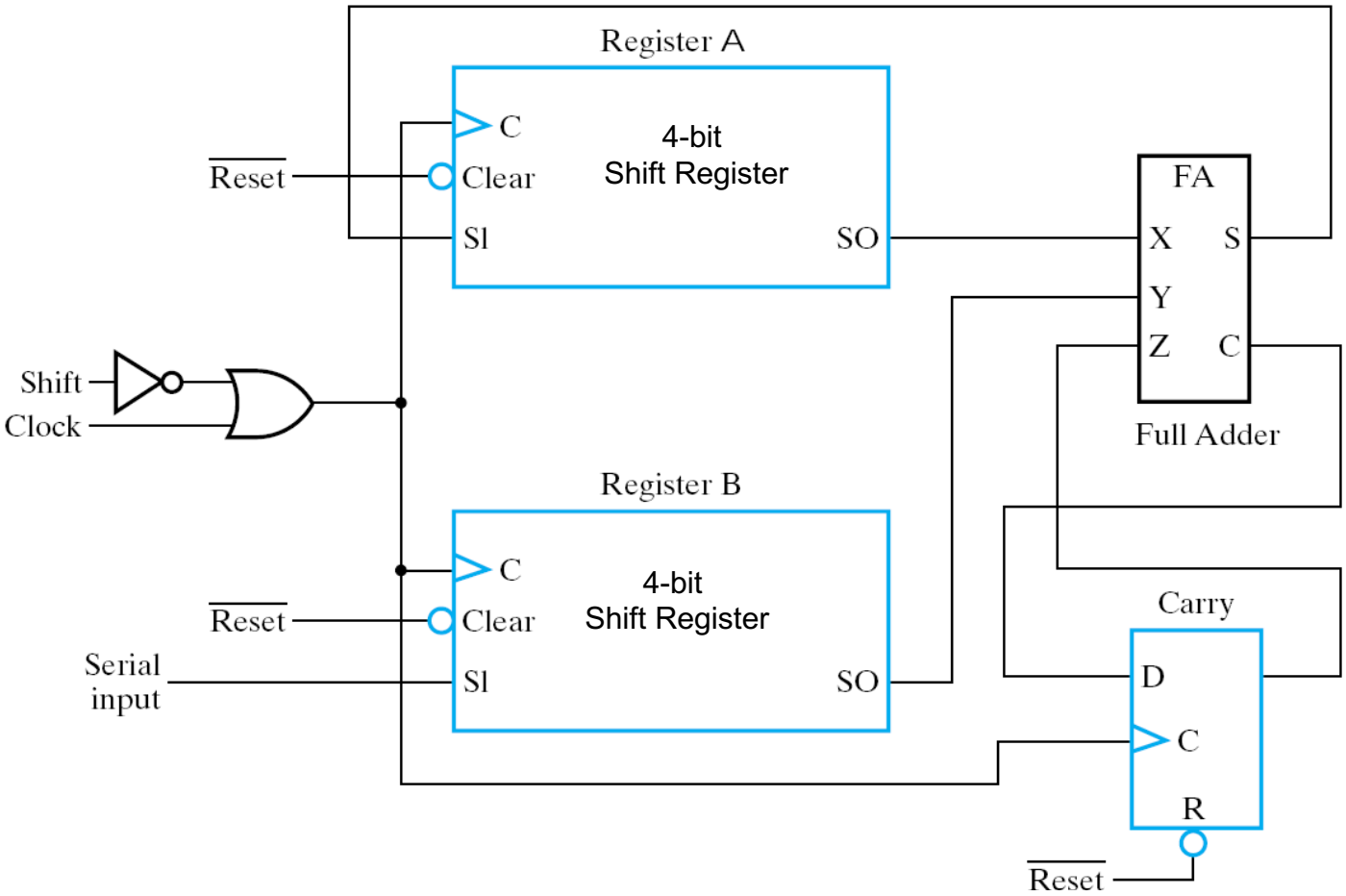


(a) Block diagram



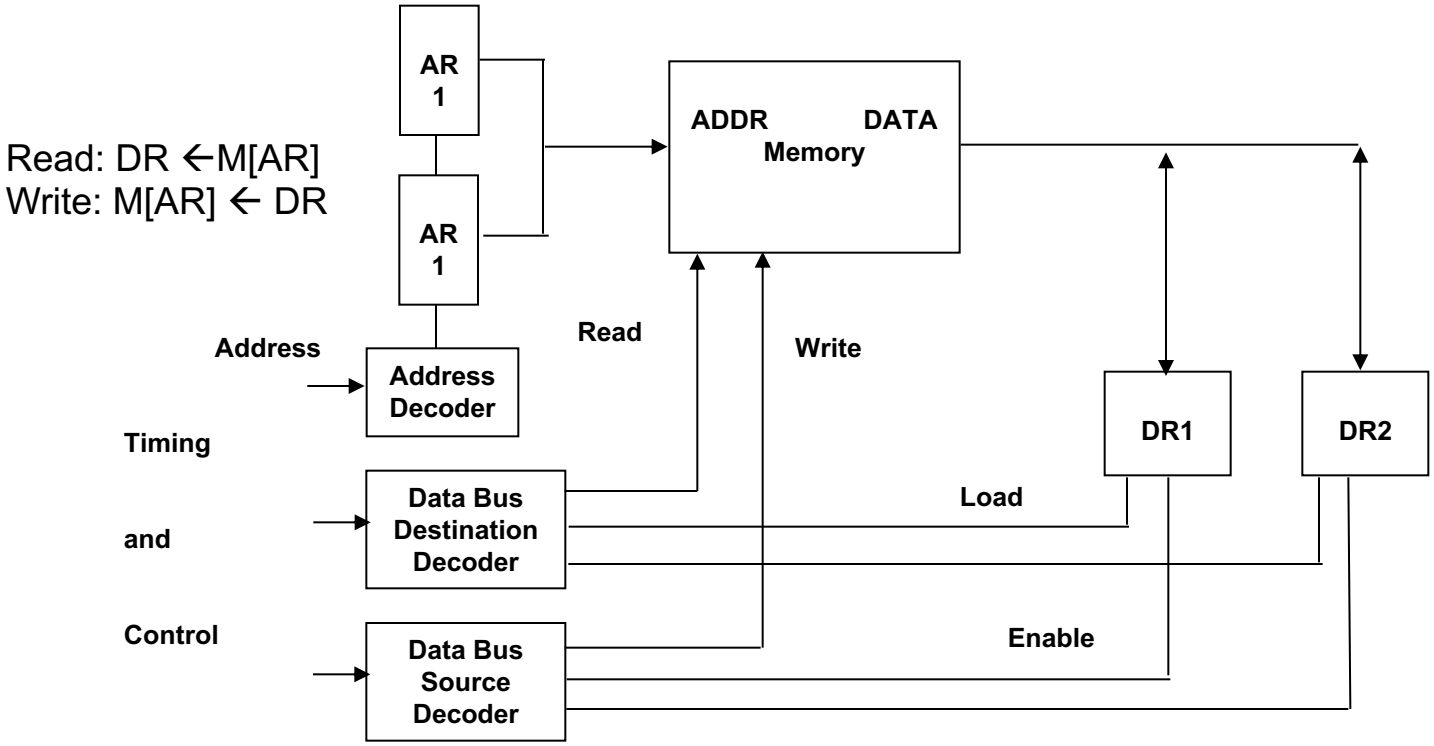
(b) Timing diagram

Serial Addition



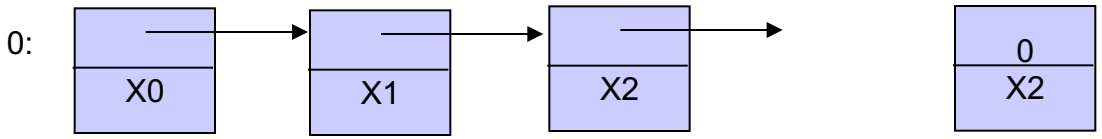
Memory transfers

- Often multiple units access the same memory (e.g. 2 source registers for the address bus – AR1, AR2 - and 2 data registers – DR1, DR2)
- If memory is slow compared to other logic
  - The address and write data may need to be held on buses for multiple cycles
  - Data is captured in data register after multiple cycles after read is started



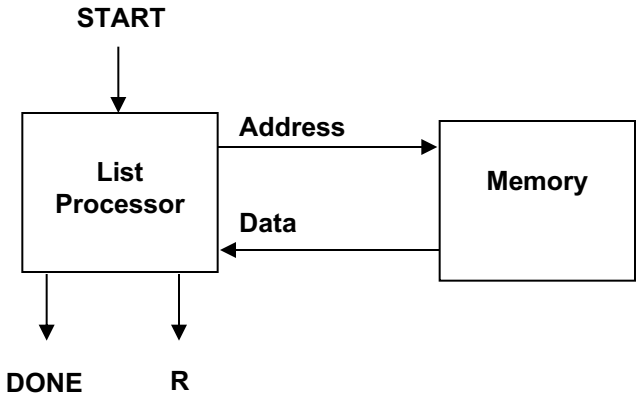
Example: List Processor

- Design a circuit that forms the sum of all the 2's complements integers stored in a linked-list structure starting at memory address 0:
- All integers and pointers are 8-bit. The linked-list is stored in a memory block with an 8-bit address port and 8-bit data port
- The pointer from the last element in the list is 0.



I/Os:

- START resets to head of list and starts addition process.
- DONE signals completion
- R, Bus that holds the final result



Example: List processor algorithm specification

Algorithm Specification

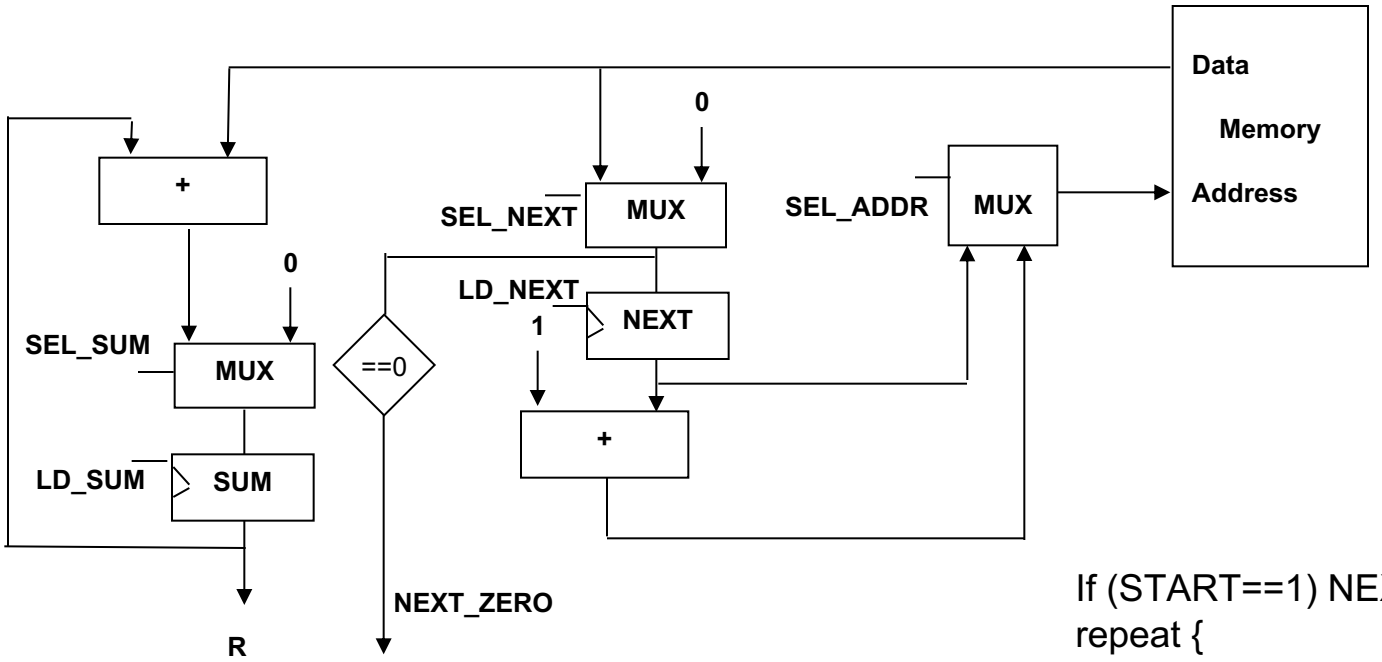
- In this case the memory only allows one access per cycle, so the algorithm is limited to sequential execution. If in another case more input data is available at once, then a more parallel solution may be possible

- Assume datapath state registers NEXT and SUM:
  - NEXT holds a pointer to the node in memory.
  - SUM holds the result of adding the node values to this point

```
If (START==1) NEXT←0, SUM←0;
repeat {
  SUM←SUM + Memory[NEXT+1];
  NEXT←Memory[NEXT];
 } until (NEXT==0);
R←SUM, DONE←1;
```

Example: List processor Solution 1

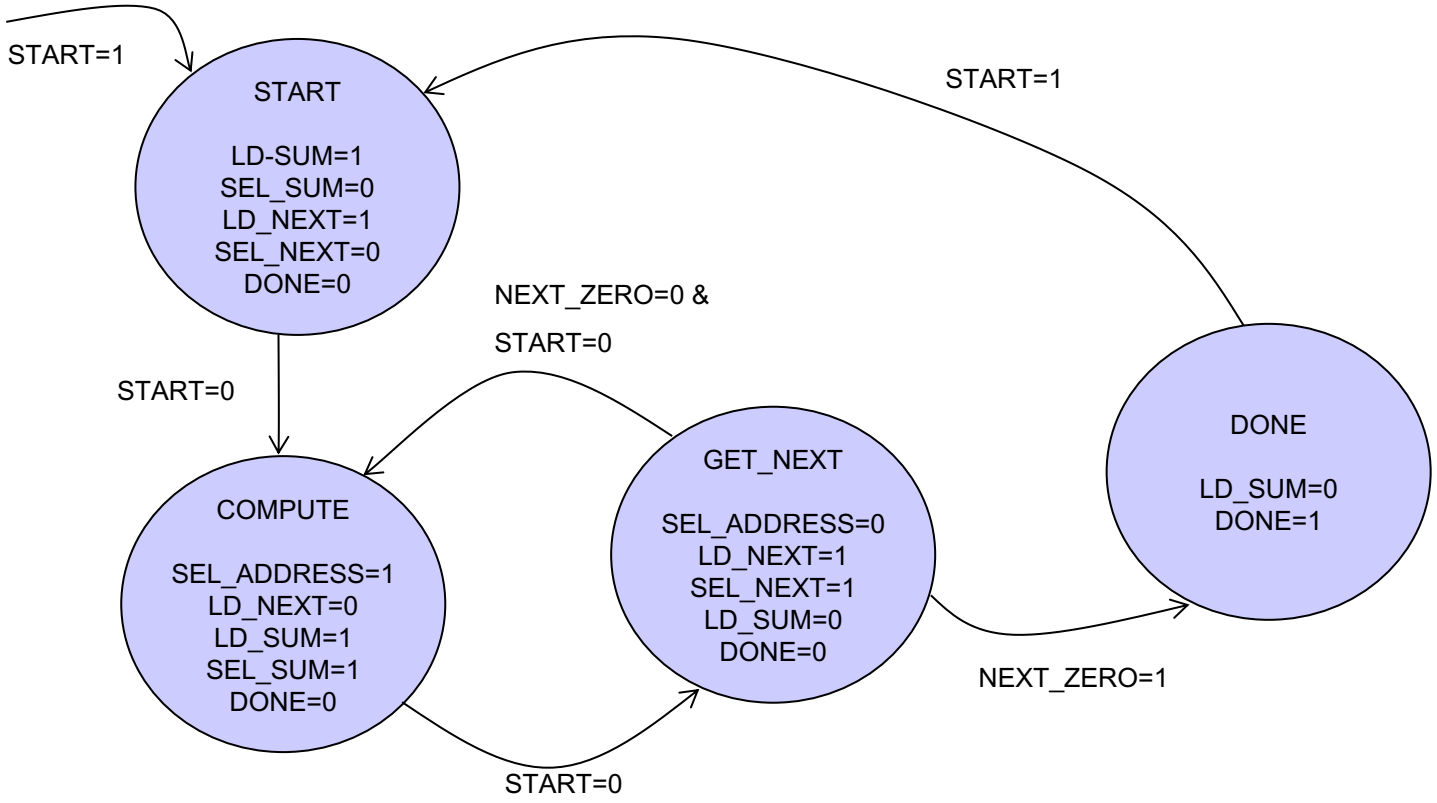
Datapath




```
If (START==1) NEXT←0, SUM←0;
repeat {
  SUM←SUM + Memory[NEXT+1];
  NEXT←Memory[NEXT];
 } until (NEXT==0);
R←SUM, DONE←1;
```

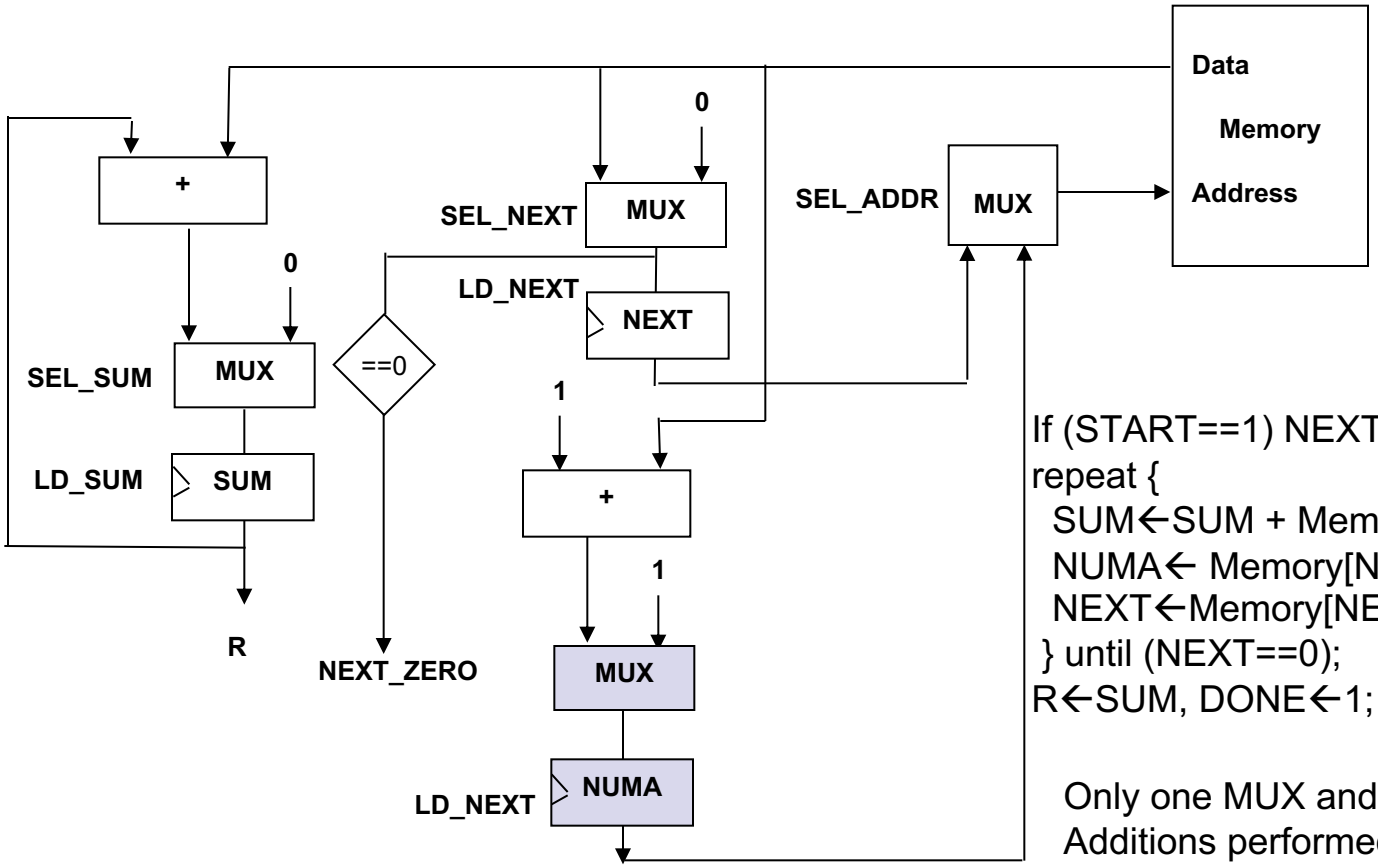
Example: List processor Solution 1

Control Unit FSM



Example: List processor Solution 2

Datapath

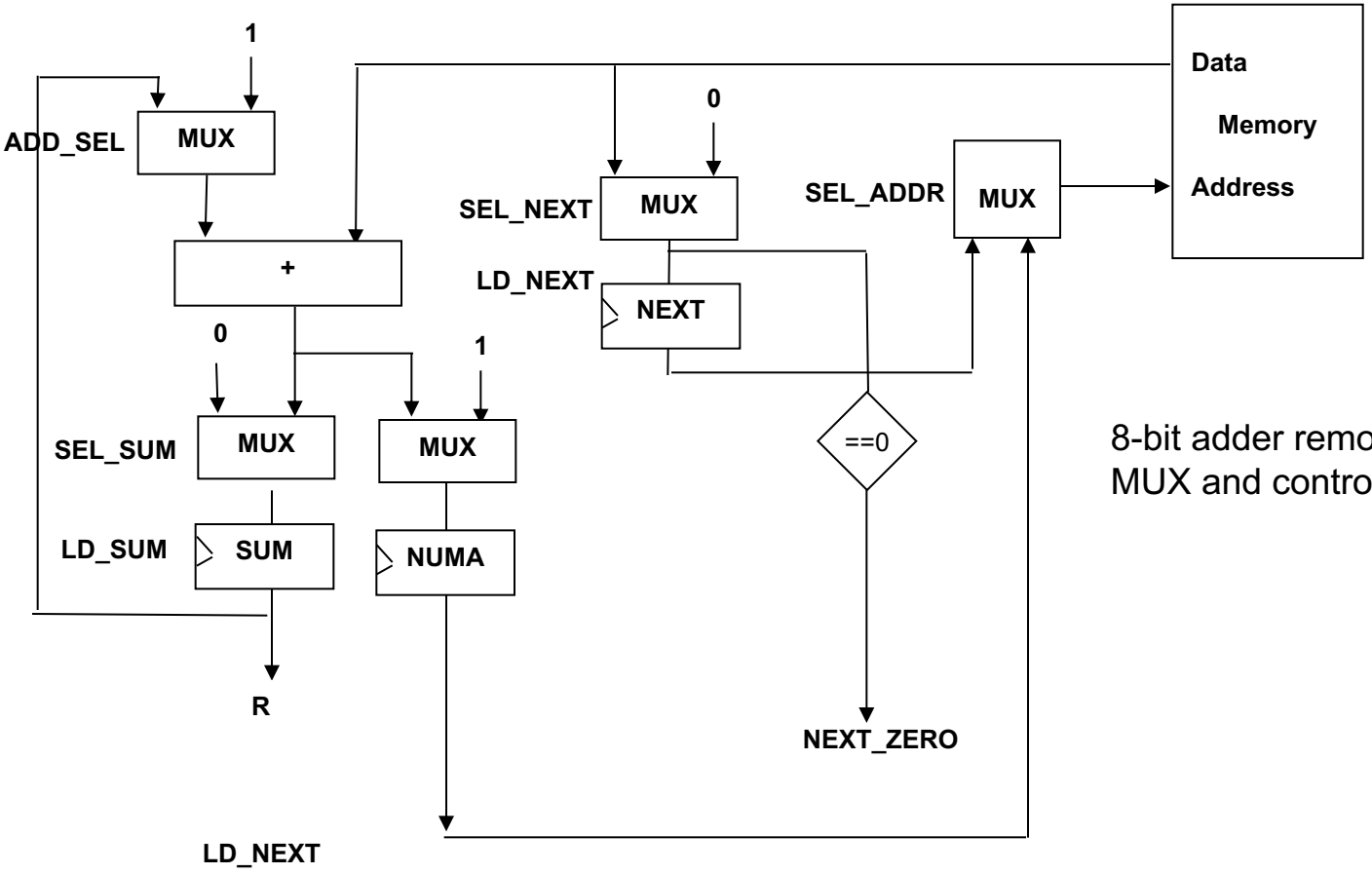



```
If (START==1) NEXT←0, SUM←0, NUMA←1;
repeat {
  SUM←SUM + Memory[NUMA];
  NUMA← Memory[NEXT]+1;
  NEXT←Memory[NEXT];
 } until (NEXT==0);
R←SUM, DONE←1;
```

Only one MUX and one register added
Additions performed in separate cycles =>
Use only one adder!

Example: List processor Solution 3

Datapath

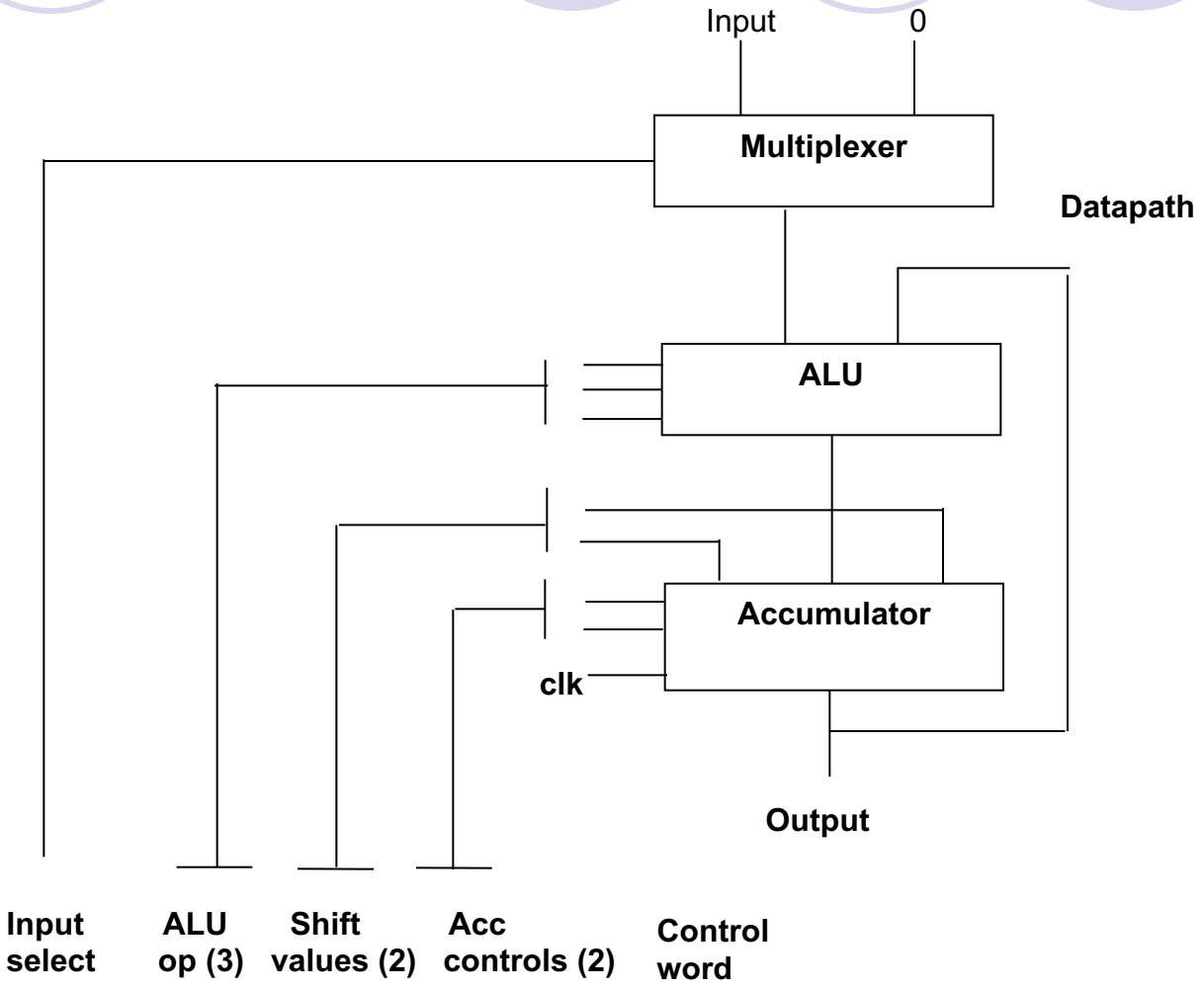


8-bit adder removed
MUX and control added

Simple datapath example 2

$$sum = \sum_{i=1}^{100} x_i$$

```
sum = 0;
Loop:
    For i =1 to 100
    sum= sum+xi;
end loop
```



More general datapath example (cnt'd)

Register file – single-write, dual-read port
Register file supplies two operands and stores one result in every clock cycle

Examples of ALU and shifter operation:

Mode	S_1	S_0	ALU operations
0	0	0	Complement A
0	0	1	AND
0	1	0	EXOR
0	1	1	OR
1	0	0	Decrement A
1	0	1	Add
1	1	0	Subtract
1	1	1	Increment A

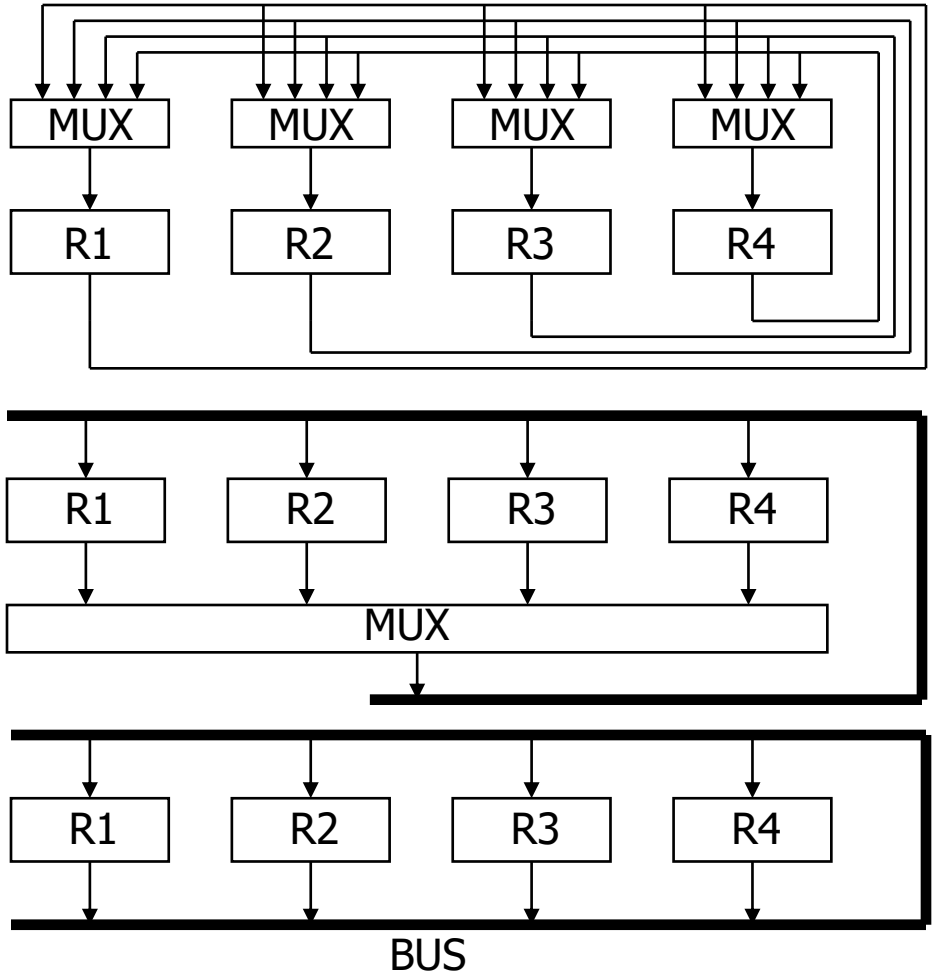
S_2	S_1	S_0	Shifter operations
0	0	0	Pass
0	0	1	Pass
0	1	0	Not used
0	1	1	Not used
1	0	0	Shift left
1	0	1	Rotate left
1	1	0	Shift right
1	1	1	Rotate right

General datapaths

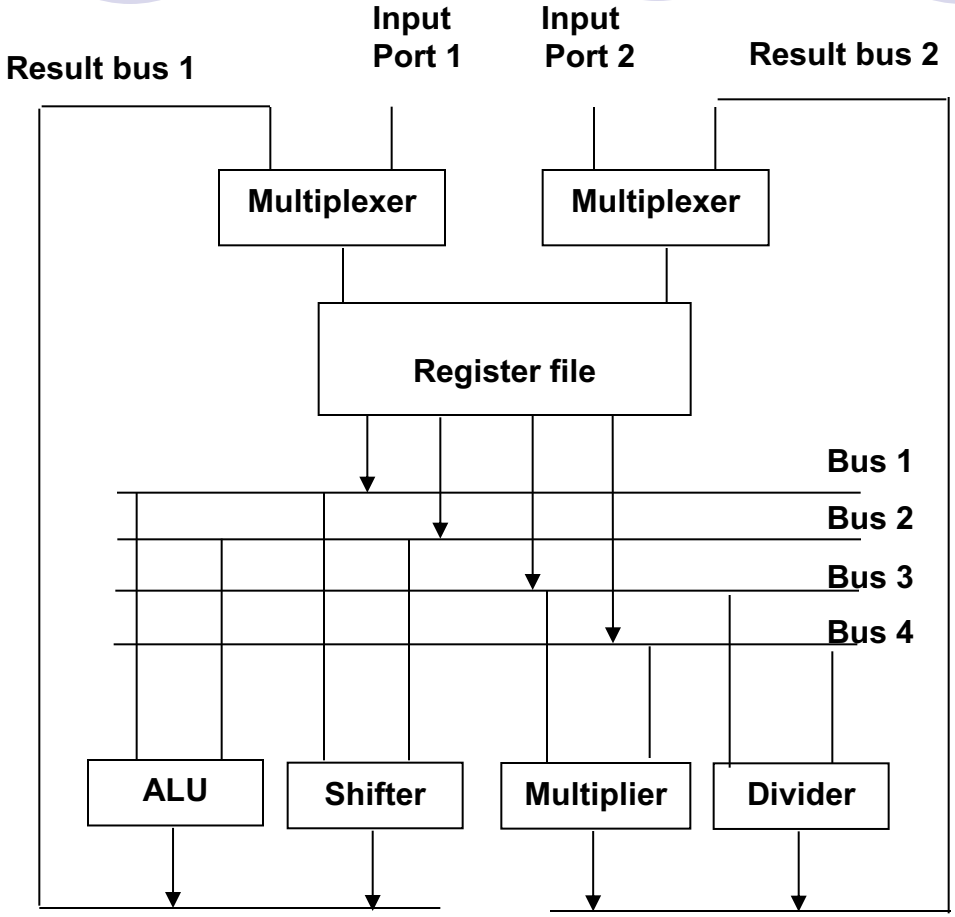
- In order to speed-up system operation and increase system performance we design datapaths that enable multiple operations at the same time (concurrently)
- They are parallel datapaths
- Data are usually stored in a multi-port register file and use several processing elements (functional units)
- Datapath in the example below can perform two operations in parallel (one in the ALU or shifter and the other in the multiplier or divider)
- However, it does not provide all kinds of parallelism (e.g. not two additions at the same time)

Creating Buses

- Point-to-point connection
  - Dedicated wires
  - Muxes on inputs of each register
- Common input from multiplexer
  - Load enables for each register
  - Control signals for multiplexer
- Common bus with output enables
  - Output enables and load enables for each register



General datapaths



Generic Datapaths

- Datapaths often have a number of
  - storage registers
  - processing units that perform arithmetic/logic operations
  - buses
  - multiplexers
  - decoders

- Microoperations specify:
  - source register(s)
  - destination register
  - operation

Generic Datapath Example

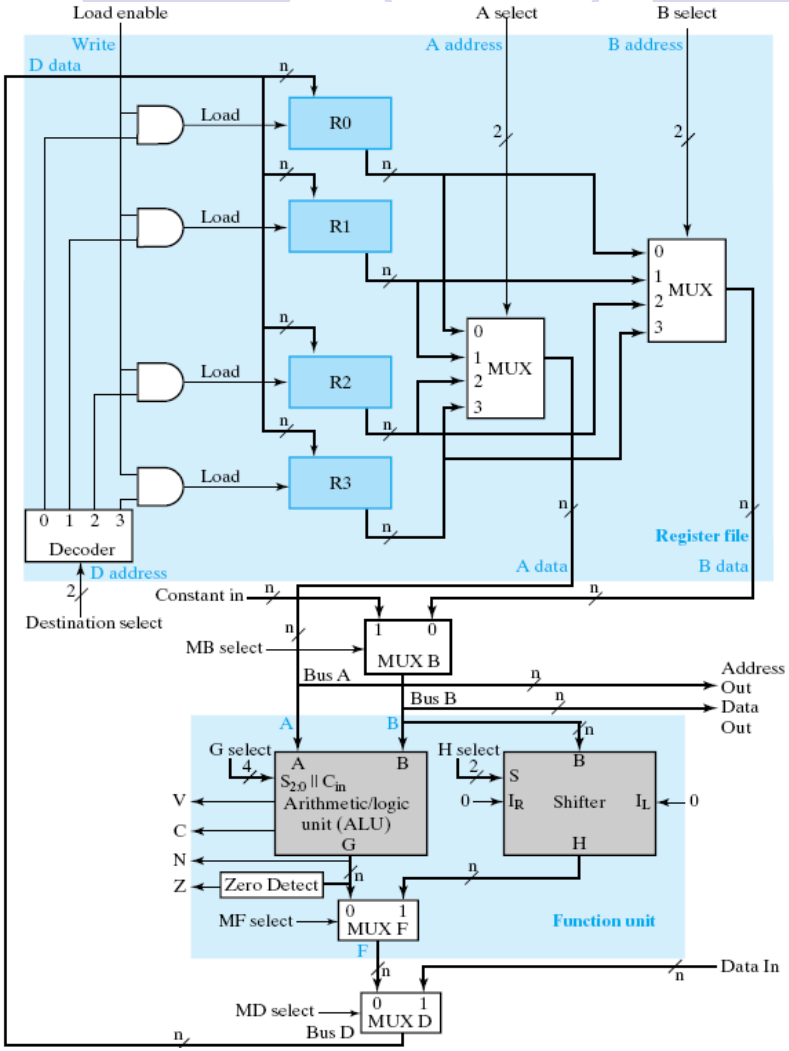
A select, B select for source data
Destination select for dest. reg.
MB select – operand B
MF select (from ALU or Shifter)

Microoperation example:

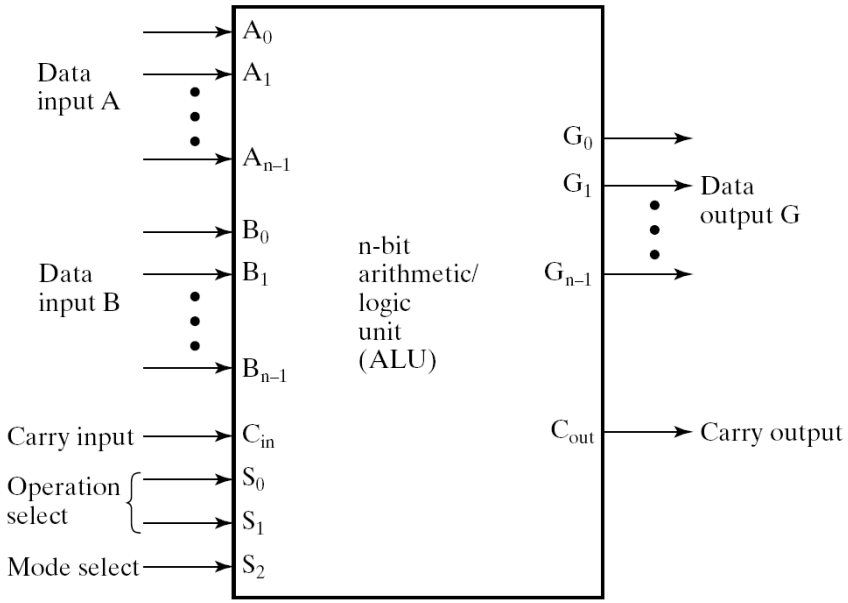
R1←R2+R3

Requires following control signals:

“A select”, “B select”, “G select” (+),
“MF select” (from G), “MD select”,
“Load 1” (R1)



Generic Datapath ALU

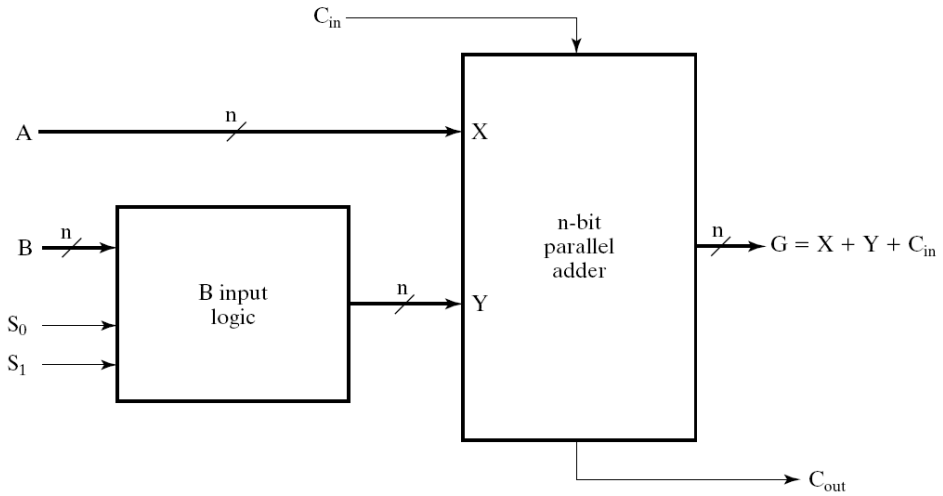


Function Table for ALU

Operation Select					
S_2	S_1	S_0	C_{in}	Operation	Function
0	0	0	0	$G \leftarrow A$	Transfer A
0	0	0	1	$G \leftarrow A + 1$	Increment A
0	0	1	0	$G \leftarrow A + B$	Addition
0	0	1	1	$G \leftarrow A + B + 1$	Add with carry input of 1
0	1	0	0	$G \leftarrow A + \overline{B}$	A plus 1's complement of B
0	1	0	1	$G \leftarrow A + \overline{B} + 1$	Subtraction
0	1	1	0	$G \leftarrow A + 1$	Decrement A
0	1	1	1	$G \leftarrow A$	Transfer A
1	X	0	0	$G \leftarrow A \wedge B$	AND
1	X	0	1	$G \leftarrow A \vee B$	OR
1	X	1	0	$G \leftarrow A \oplus B$	XOR
1	X	1	1	$G \leftarrow \overline{A}$	NOT (1's complement)

Generic Datapath ALU

Arithmetic unit

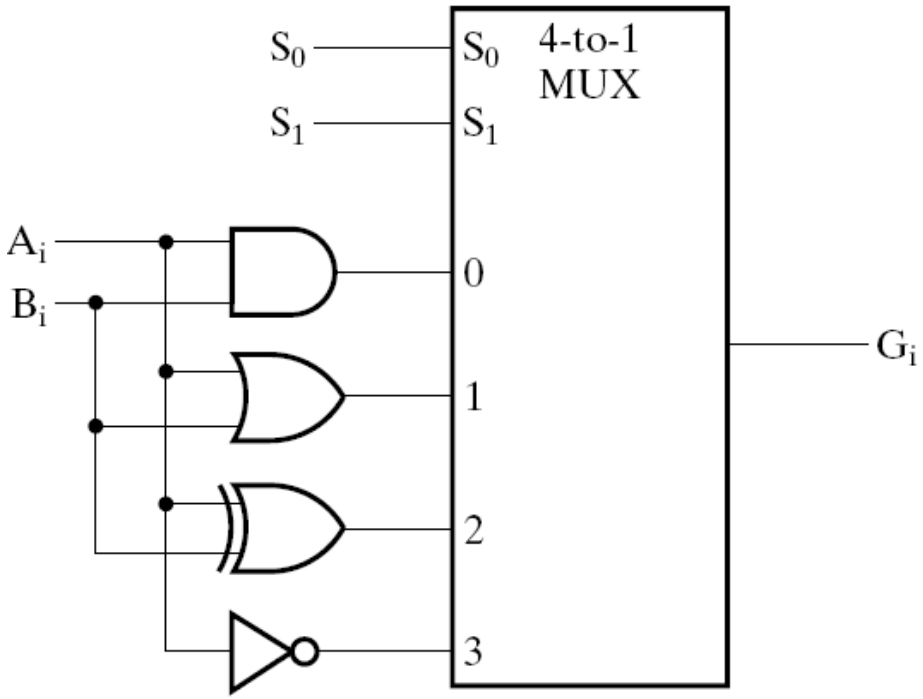


Function Table for Arithmetic Circuit

Select		Input	$G = A + Y + C_{in}$	
S_1	S_0	Y	C_{in} 0	C_{in} 1
0	0	all 0's	$G = A$ (transfer)	$G = A + 1$ (increment)
0	1	B	$G = A + B$ (add)	$G = A + B + 1$
1	0	$\overline{B}$	$G = A + \overline{B}$	$G = A + \overline{B} + 1$ (subtract)
1	1	all 1's	$G = A - 1$ (decrement)	$G = A$ (transfer)

Generic Datapath ALU

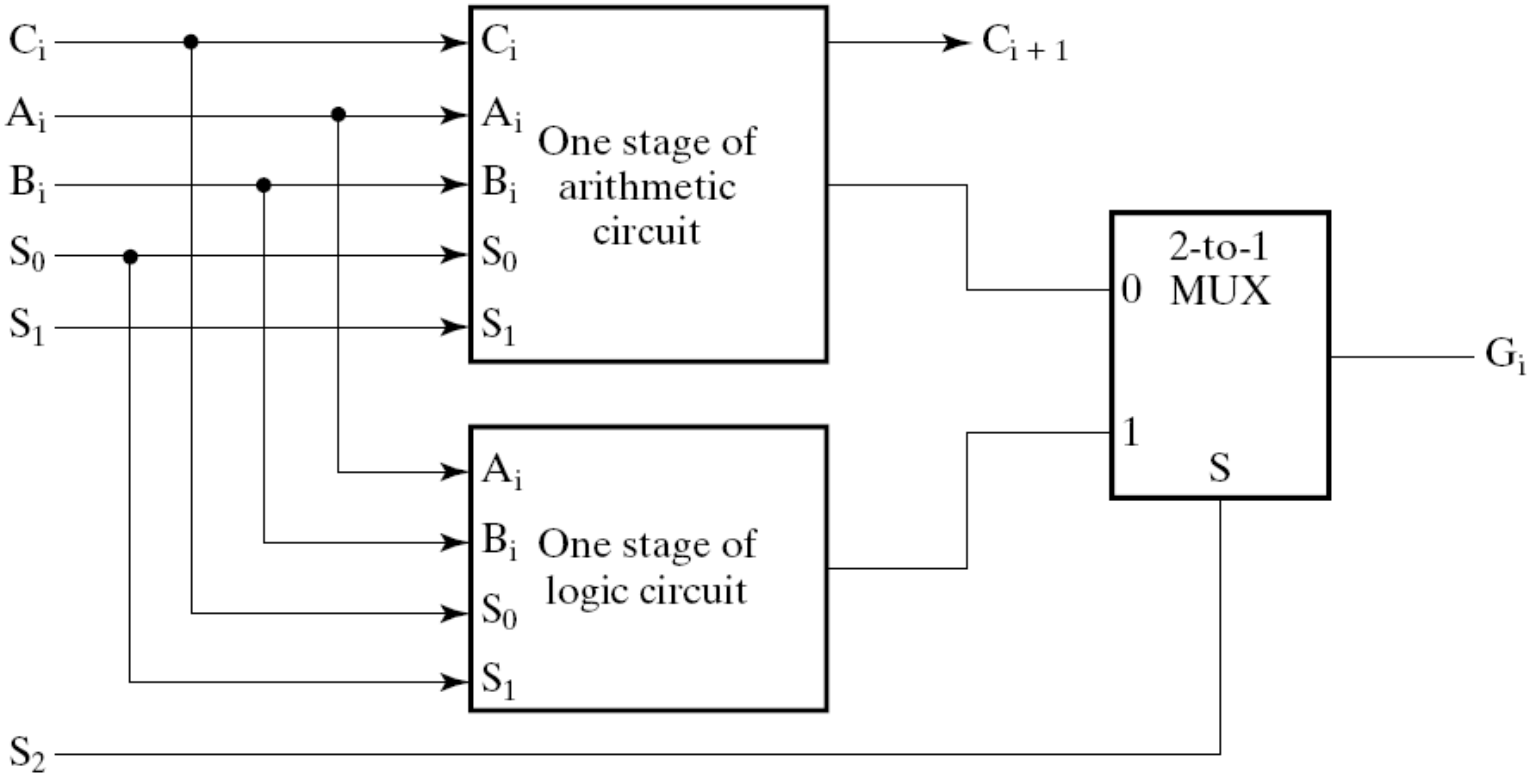
Logic Unit



S_1	S_0	Output	Operation
0	0	$G = A \wedge B$	AND
0	1	$G = A \vee B$	OR
1	0	$G = A \oplus B$	XOR
1	1	$G = \overline{A}$	NOT

Generic Datapath ALU

Single-bit stage of ALU



More general datapath example

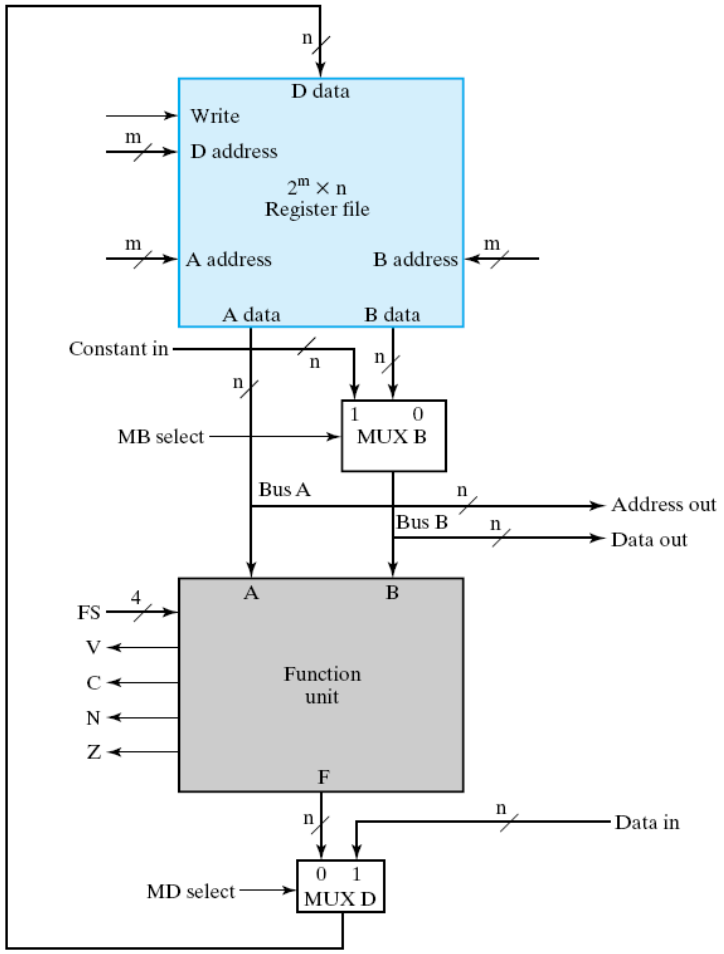
- Register file generalised
- Can be implemented using different technologies (registers, memories)
- Microoperations selected by selection variables

For m=3 (8 registers in register file) control variables make the control word

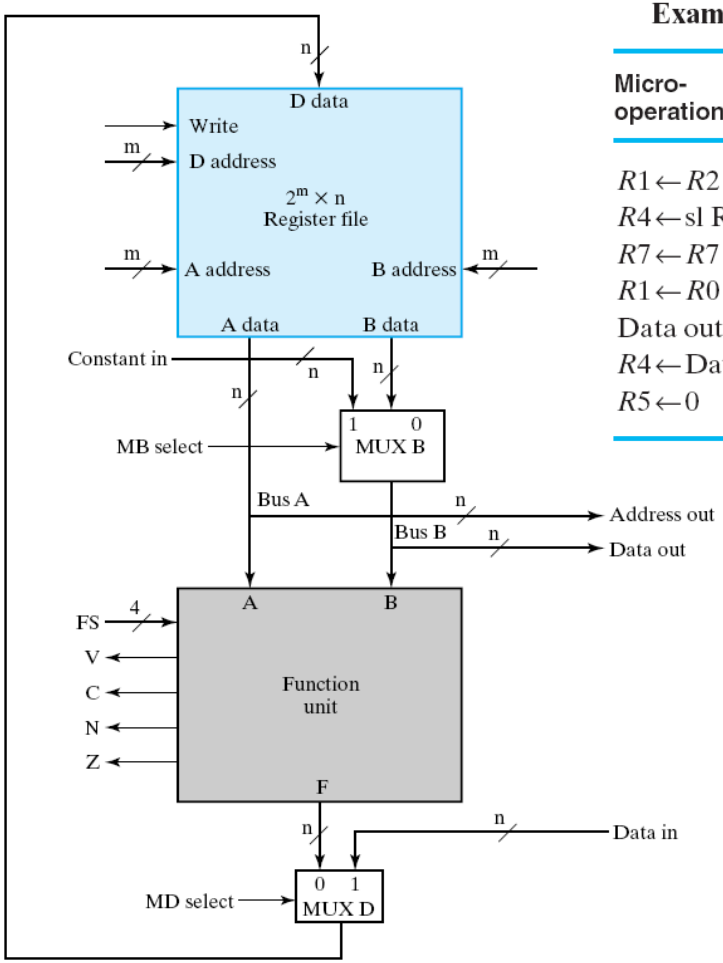
Results of microoperation started in one clock cycle appear in the following clock cycle

16 bits long control word

DA(3)	AA(3)	BA(3)	MB(1)	FS(4)	MD(1)	RW(1)



More general datapath – examples of microoperations



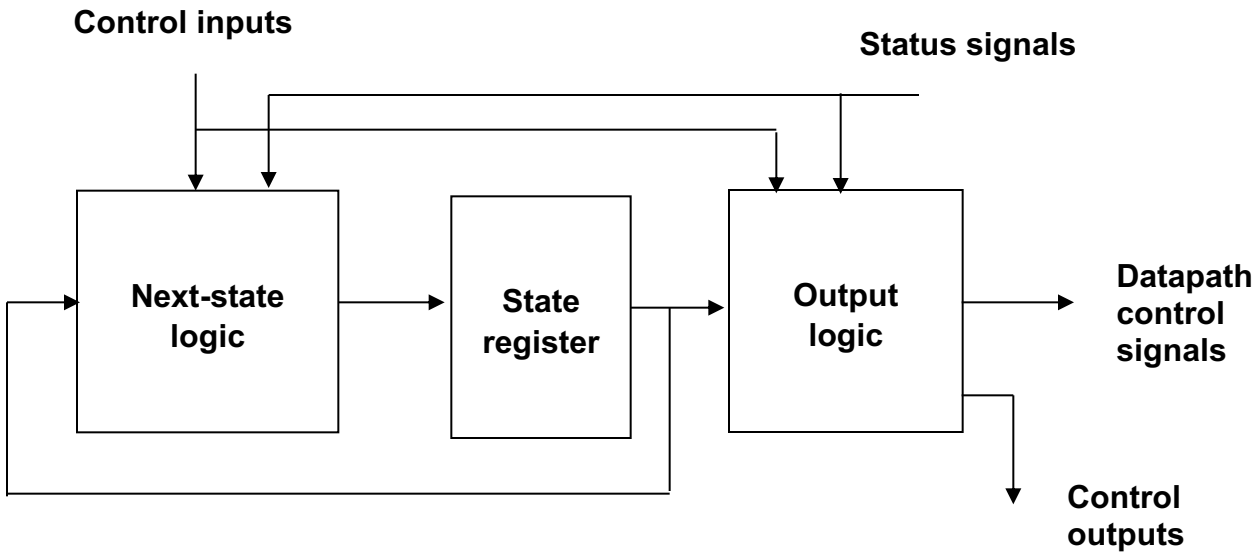
Examples of Microoperations for the Datapath, Using Symbolic Notation

Micro-operation	DA	AA	BA	MB	FS	MD	RW
$R1 \leftarrow R2 - R3$	$R1$	$R2$	$R3$	Register	$F = A + \overline{B} + 1$	Function	Write
$R4 \leftarrow$ sl R6	$R4$	—	$R6$	Register	$F =$ sl 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

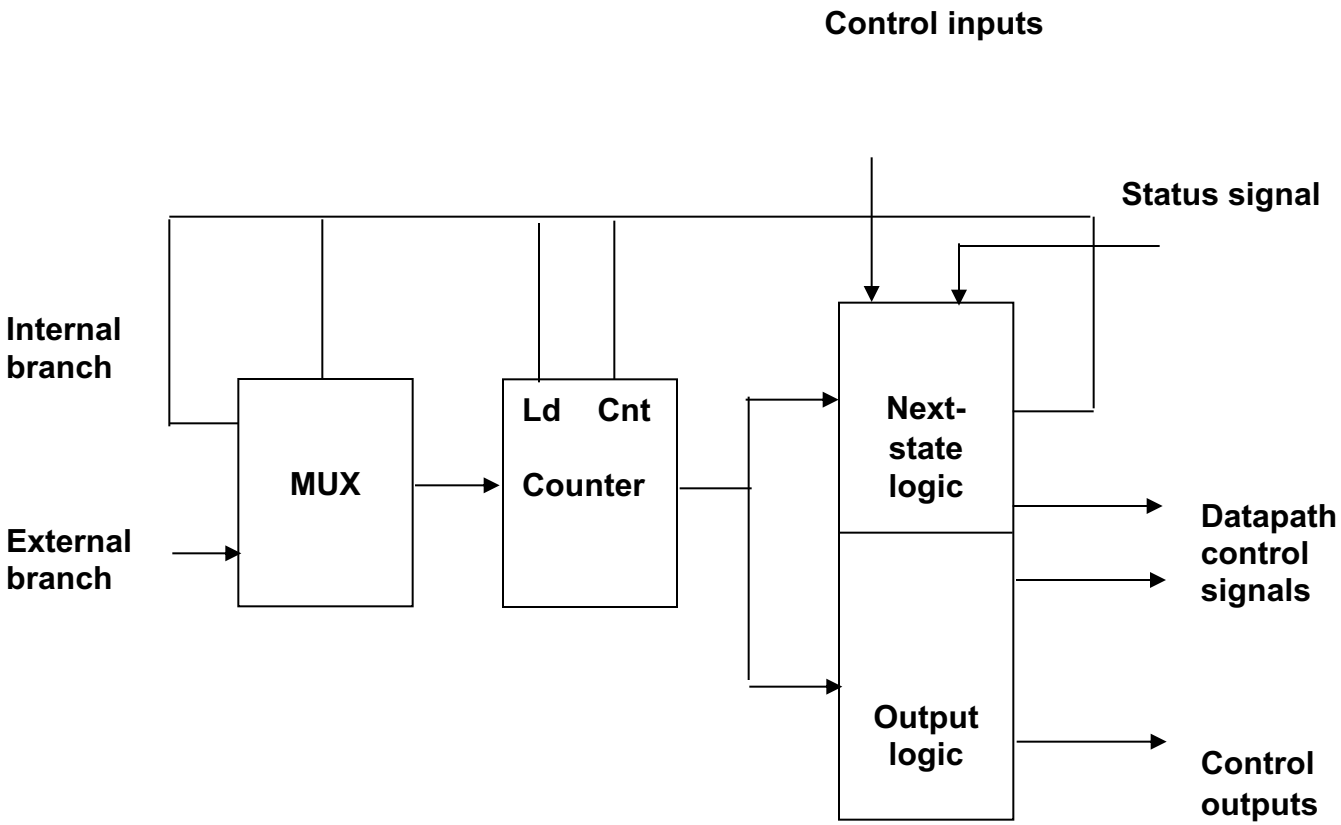
Control units

- Most often control unit is designed by following FSM model consisting of
  - a next-state logic,
  - a state register, and
  - an output logic
- This model is used mostly for application-specific and custom designs
- The amount of next-state logic can be reduced if each state in the sequence has one successor
- The state register can be replaced with a counter which has two more control signals (load and count)
- Load signal is used to branch out of sequence
- The branch can be supplied either internally or externally through control inputs

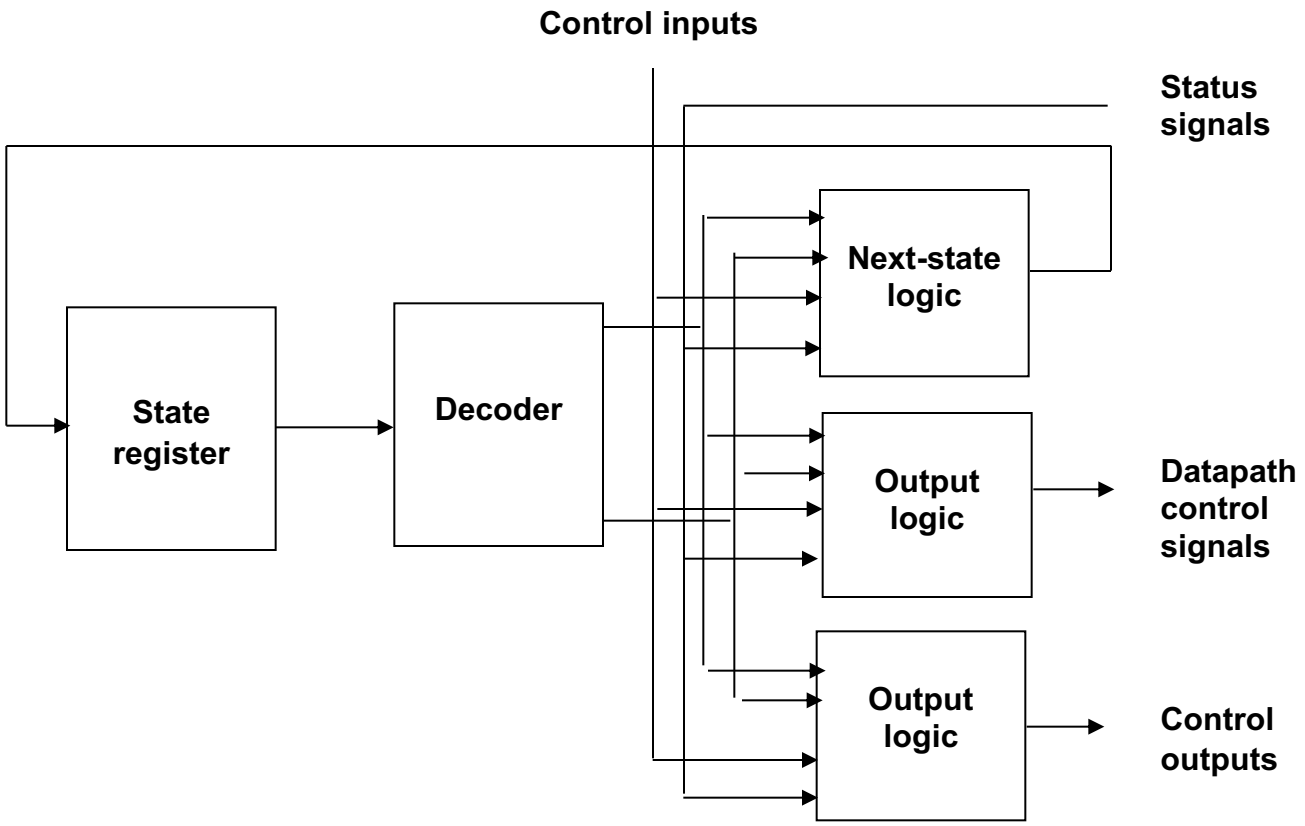
Control units – General model



Control unit – Model with counter



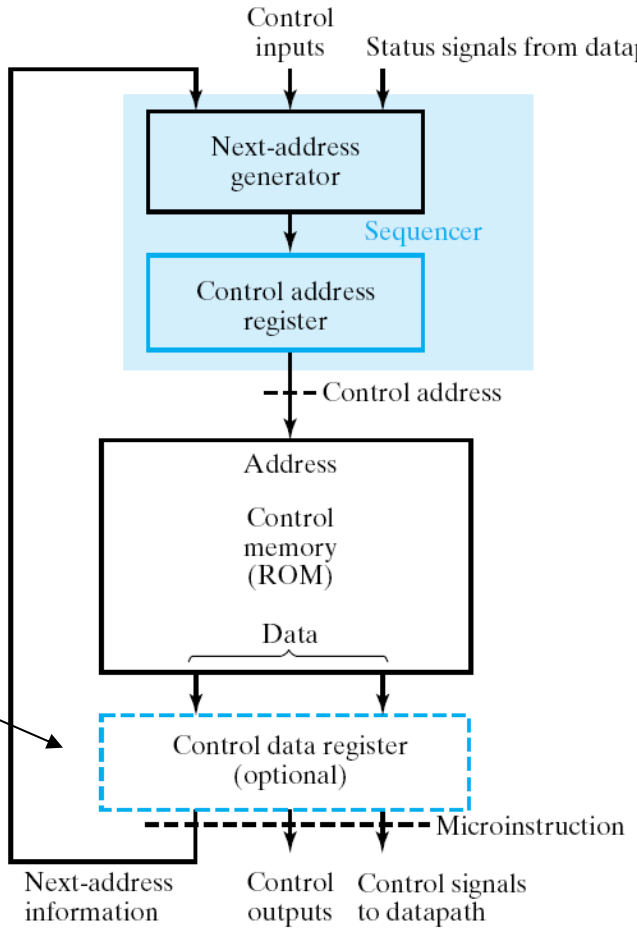
Control unit – Model with state register



Control unit - Microprogramming

- Binary control values stored as words in memory
- Each word contains a microinstruction that specifies one or more microoperations for the system
- A sequence of microinstructions make a microprogram
- Microoperations are performed in both datapath and control unit
- Control memory can be ROM or RAM

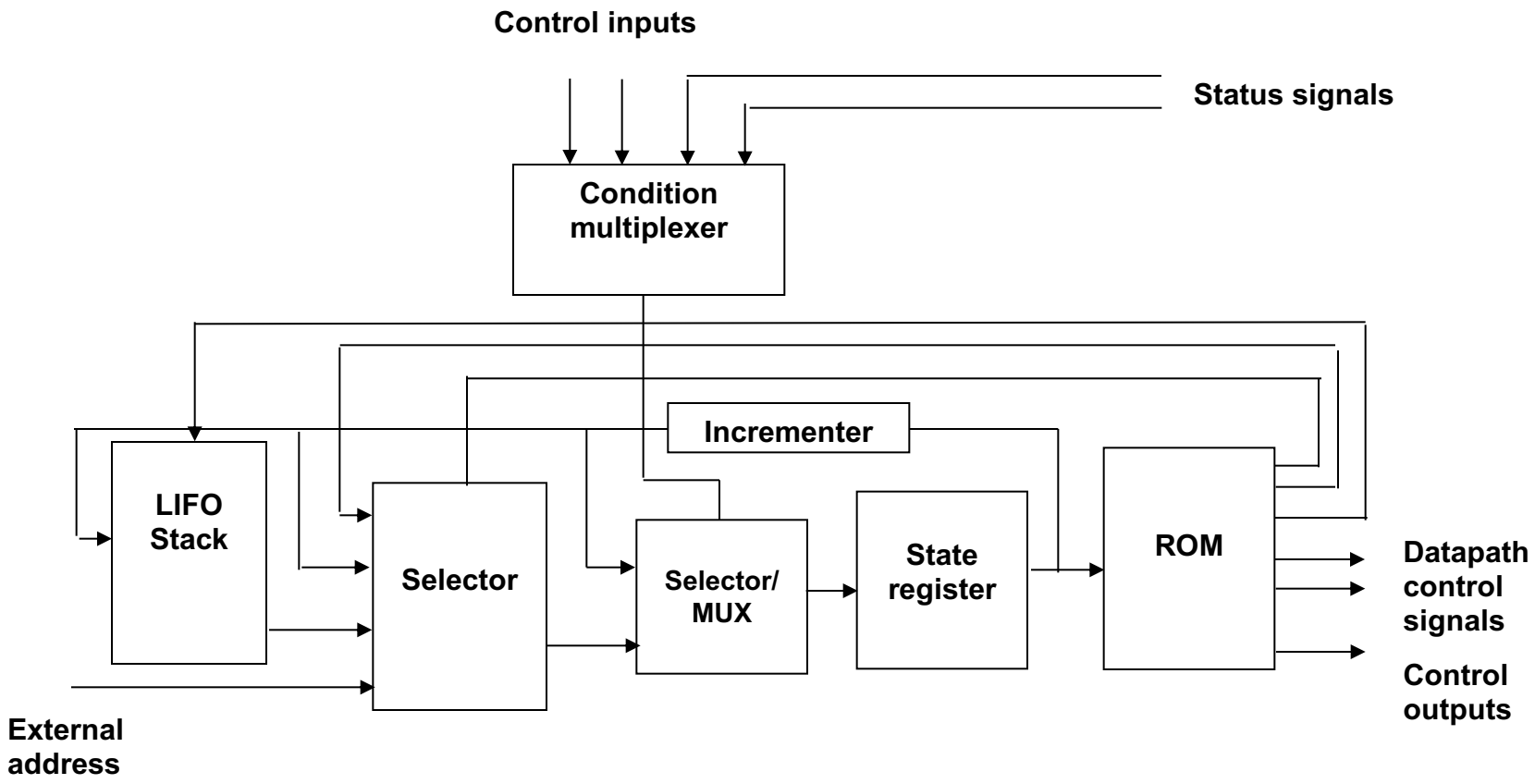
  - Breaks up combinational delay
  - One clock cycle = one microinstruction
  - Status bits enter control unit =>
  - Control unit Moore type machine



Control unit - Microprogramming

- The state register is an address register to the control memory (ROM)
- It is important to limit number of control and status signals for the next-state selection as the cost of control memory doubles for each new address line
- Only one signal used to select the next state - this limits the branching capability to two-way branching
- The next address is either an incremented present address or one of the branching addresses

Control unit – Microprogramming

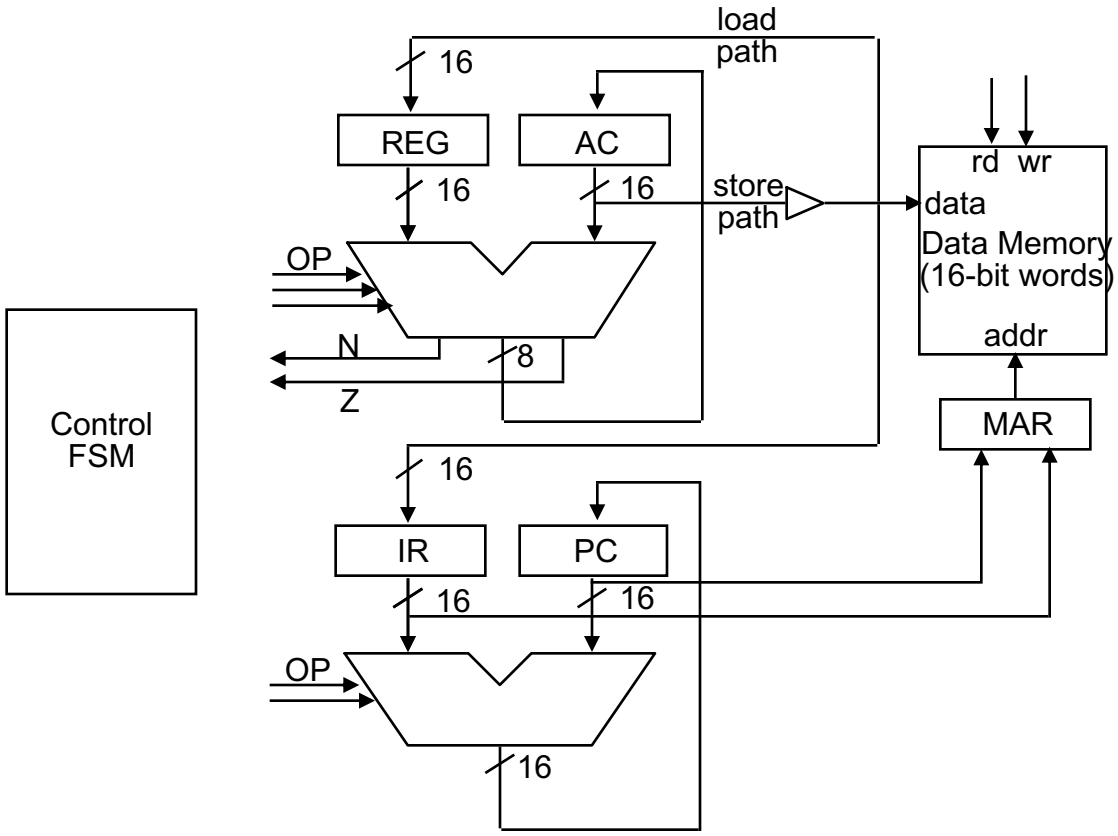


Simple Processor - Datapath Memory Interface

- Memory
  - Separate data and instruction memory (Harvard architecture)
    - Two address busses, two data busses
  - Single combined memory (von Neumann architecture)
    - Single address bus, single data bus
- Separate memory
  - ALU output goes to data memory input
  - Register input from data memory output
  - Data memory address from instruction register
  - Instruction register from instruction memory output
  - Instruction memory address from program counter
- Single memory
  - Address from PC or IR
  - Memory output to instruction and data registers
  - Memory input from ALU output

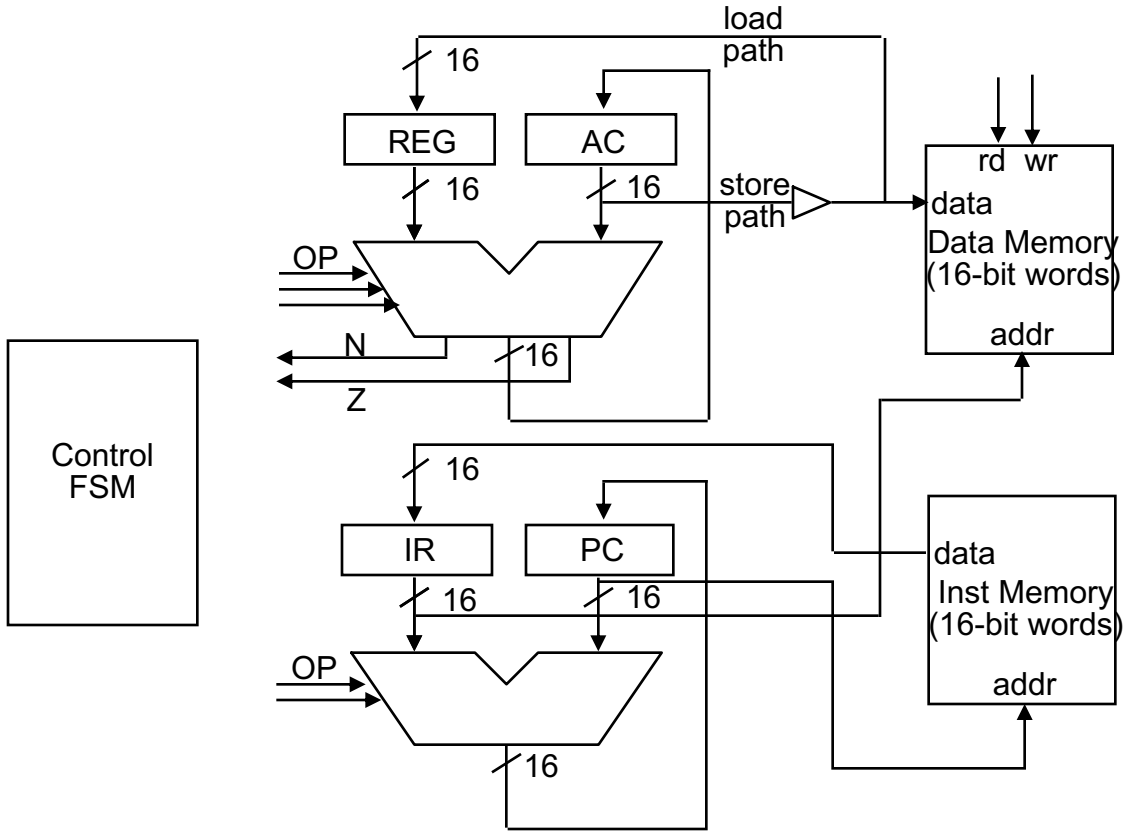
Block Diagram of a Simple Processor

- Register Transfer View of von Neumann Architecture
  - Which register outputs are connected to which register inputs
  - Arrows represent data-flow, other are control signals from control FSM
  - MAR may be a simple multiplexer
    rather than separate register
  - MBR is split in two
    (REG and IR)
  - Load control
    for each register



Block Diagram of a Simple Processor

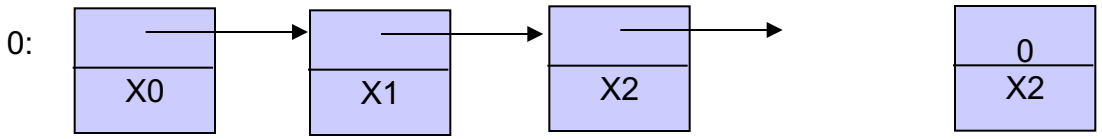
- Register transfer view of Harvard architecture
  - Which register outputs are connected to which register inputs
  - Arrows represent data-flow, other are control signals from control FSM
  - Two MARs (PC and IR)
  - Two MBRs (REG and IR)
  - Load control for each register



- Example of RTL Design – List processor

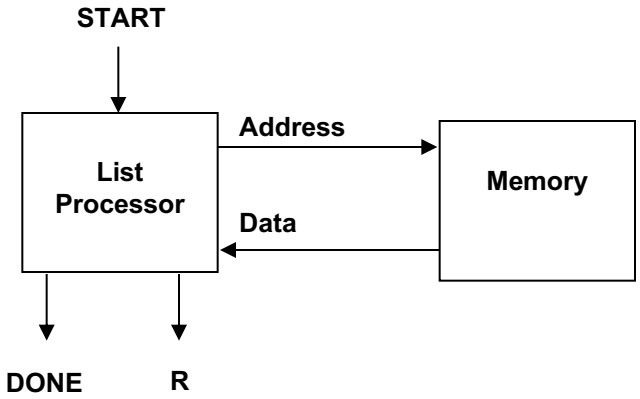
Example: List Processor

- Design a circuit that forms the sum of all the 2's complements integers stored in a linked-list structure starting at memory address 0:
- All integers and pointers are 8-bit. The linked-list is stored in a memory block with an 8-bit address port and 8-bit data port
- The pointer from the last element in the list is 0.



I/Os:

- START resets to head of list and starts addition process.
- DONE signals completion
- R, Bus that holds the final result



Example: List processor algorithm specification

Algorithm Specification

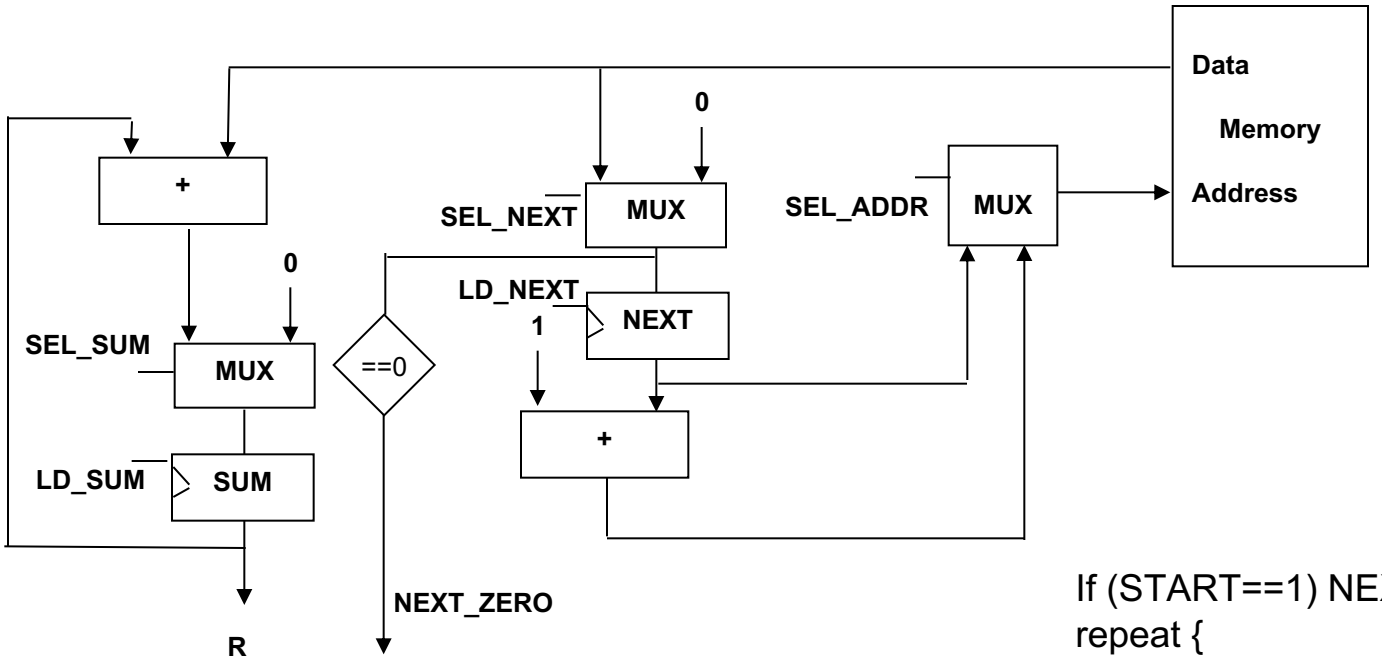
- In this case the memory only allows one access per cycle, so the algorithm is limited to sequential execution. If in another case more input data is available at once, then a more parallel solution may be possible

- Assume datapath state registers NEXT and SUM:
  - NEXT holds a pointer to the node in memory.
  - SUM holds the result of adding the node values to this point

```
If (START==1) NEXT←0, SUM←0;
repeat {
  SUM←SUM + Memory[NEXT+1];
  NEXT←Memory[NEXT];
 } until (NEXT==0);
R←SUM, DONE←1;
```

Example: List processor Solution 1

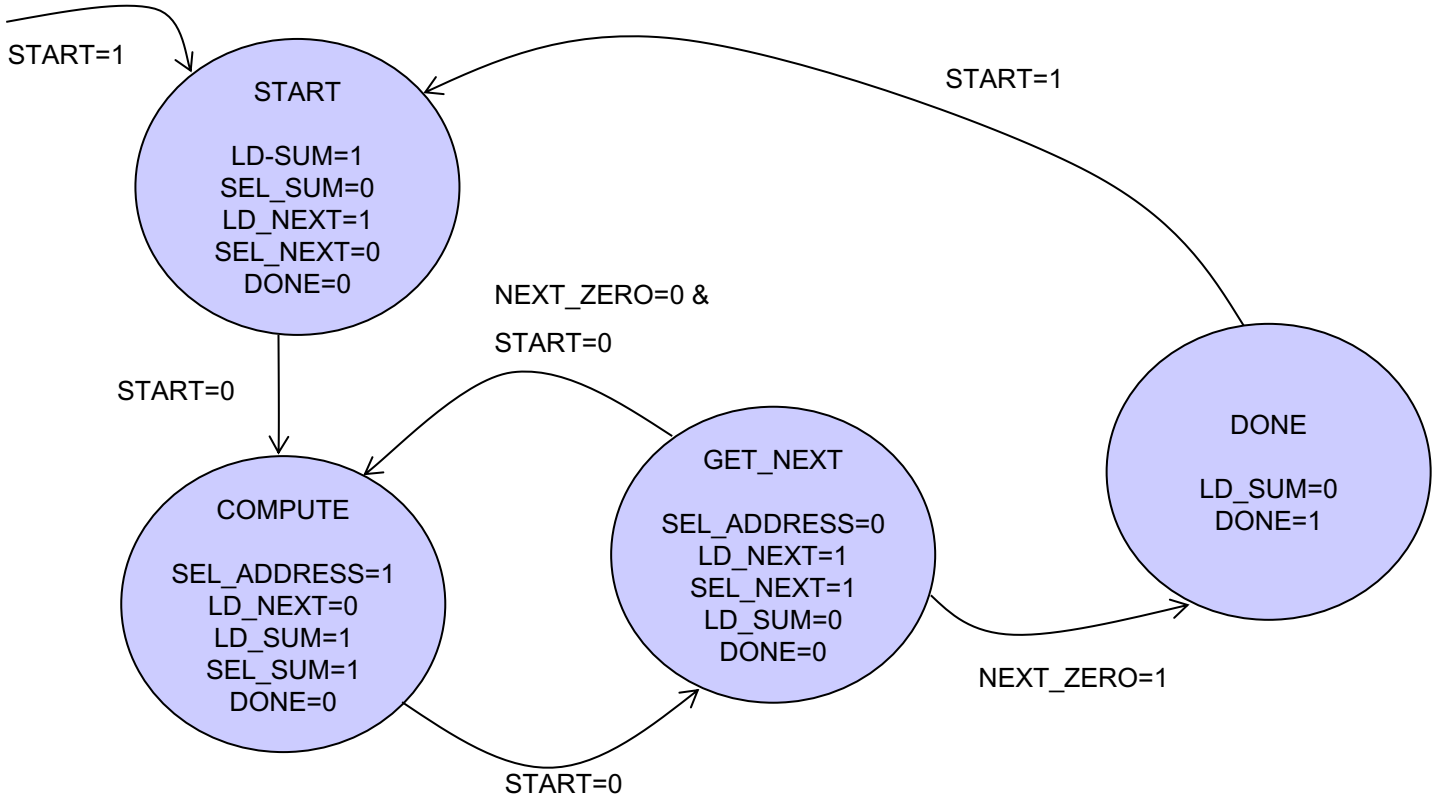
Datapath



If (START==1) NEXT←0, SUM←0;
repeat {
  SUM←SUM + Memory[NEXT+1];
  NEXT←Memory[NEXT];
 } until (NEXT==0);
R←SUM, DONE←1;

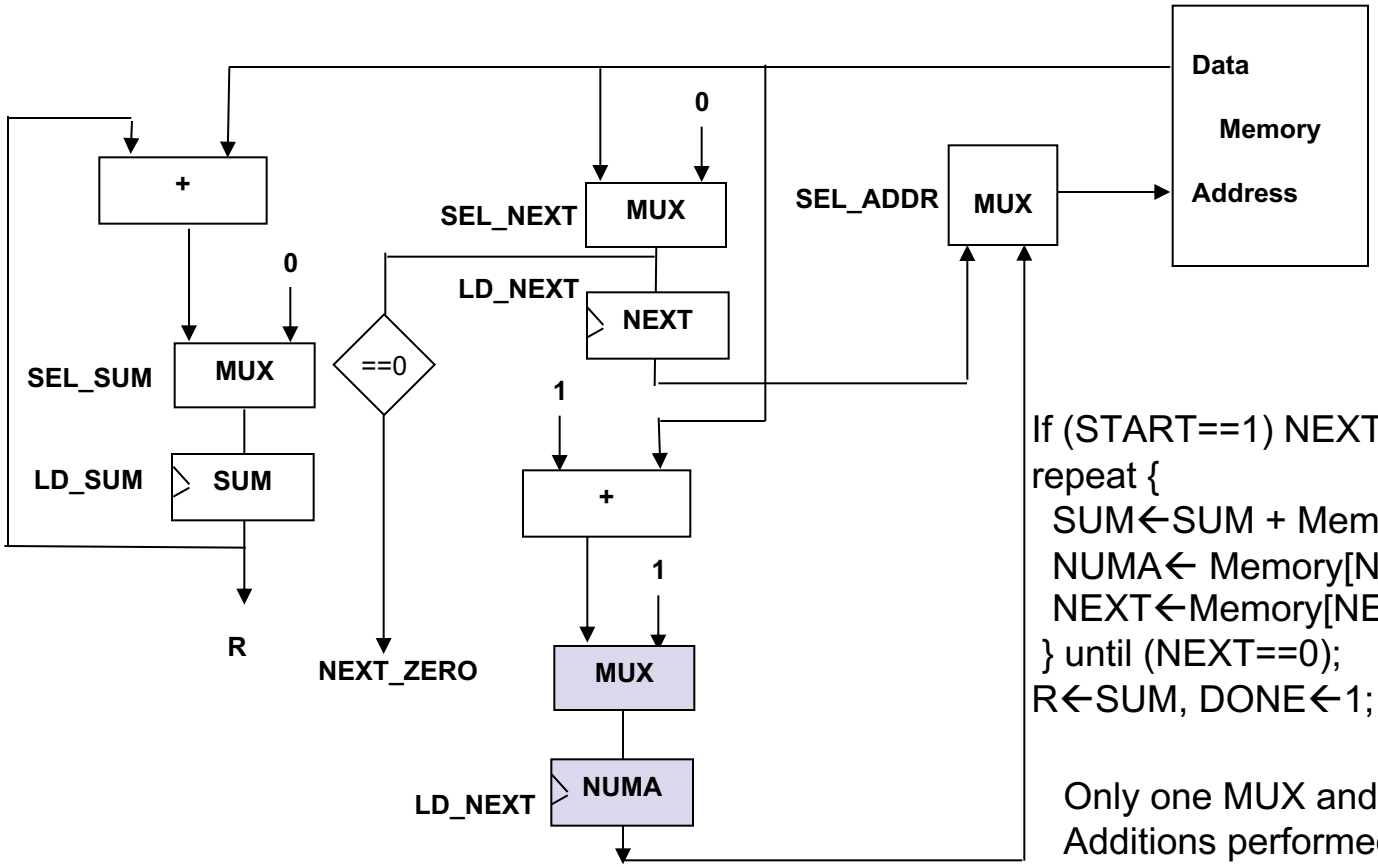
Example: List processor Solution 1

Control Unit FSM



Example: List processor Solution 2

Datapath

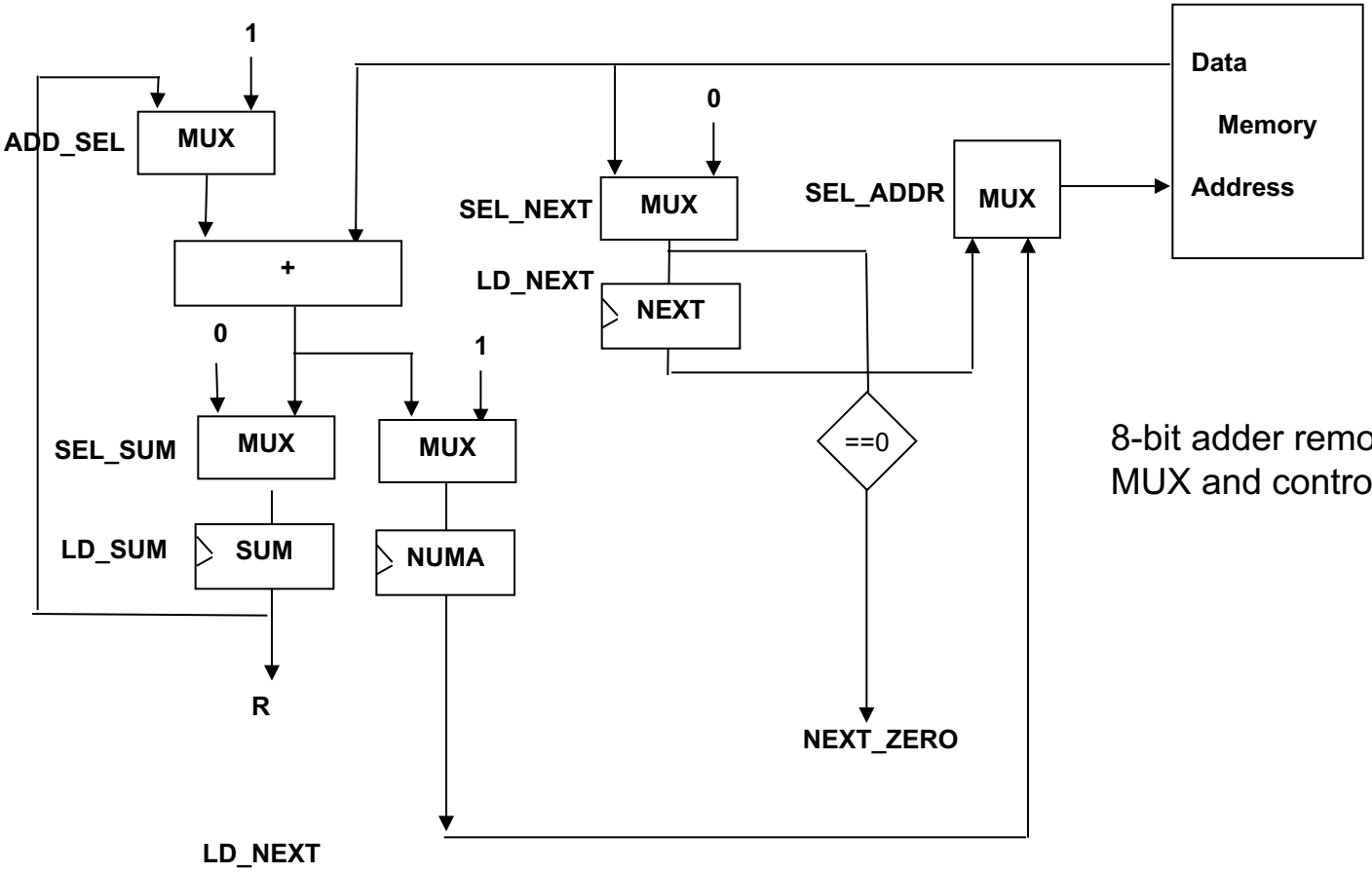



```
If (START==1) NEXT←0, SUM←0, NUMA←1;
repeat {
  SUM←SUM + Memory[NUMA];
  NUMA← Memory[NEXT]+1;
  NEXT←Memory[NEXT];
 } until (NEXT==0);
R←SUM, DONE←1;
```

Only one MUX and one register added
Additions performed in separate cycles =>
Use only one adder!

Example: List processor Solution 3

Datapath



8-bit adder removed
MUX and control added

Example: List processor performance evaluation

Timing design constraints

Component	Propagation delay
Logic gate	0.5 ns
n-bit register	Clk-to-Q = 0.5 ns Setup time = 0.5 ns
n-bit 2-to-1 multiplexer	1 ns
n-bit adder	(2log(n)+2) ns
Memory	10 ns read (asynchronous)
Zero compare	0.5 log(n)

Performance:

- 2 cycles per number added
- What is minimum clock period?
- Control unit may be on critical path!

Example: List processor performance evaluation

Performance analysis – Solution 1

Compute state:

(CLK-Q=0.5)+(8-bit add=8)+(MUX=1)+(Memory=10)+(15-bit add=10)+(MUX=1)+(setup=0.5)=31ns

Get_Next state:

(ctrlout delay=0.5)+(MUX=1)+(Memory=10)+(MUX=1)+(Compare=1.5)+(ctrin delay=1.5)=15.5ns

Clock period = max for each state= 31ns, f<32MHz

Solution 2: T=23 ns, f=43 MHz

Solution 3: T=24 ns, f=41.67 MHz

Example: List processor performance evaluation

Resource (operation) usage chart

- Used to help schedule operations on shared resources (memory and adder)
- Operations on y axis, time (cycles) on x axis
- Example:

Cycle	1	2	3	4	5	6
Memory	Fetch A1		Fetch A2			
Bus	Fetch A1		Fetch A2			
Register file		Read B1		Read B2		
Adder			A1+B1		A1+B2	

Example: List processor performance evaluation

Resource (operation) usage chart

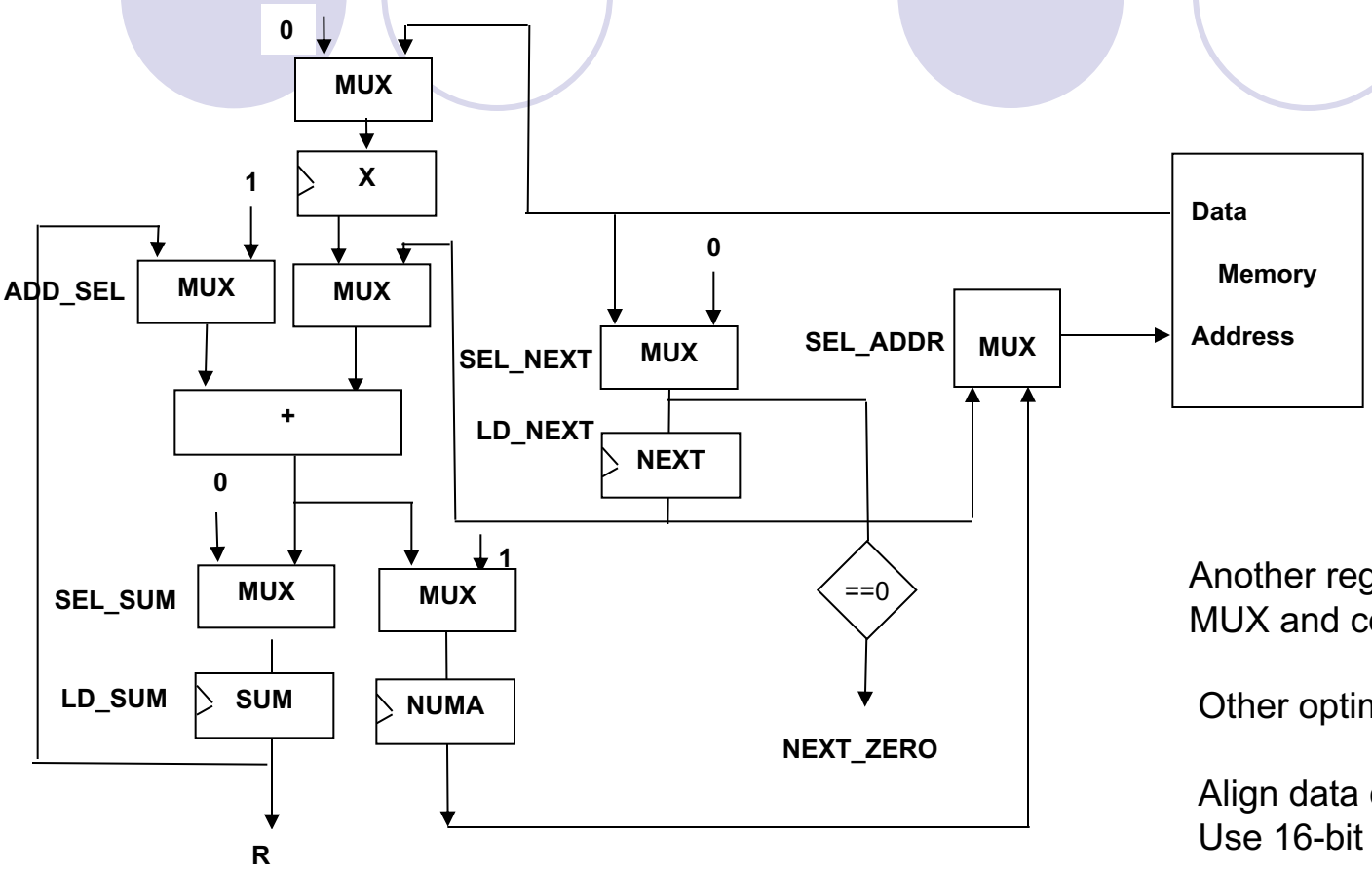
- Unoptimised solution: 1. SUM←SUM+M[NEXT+1] 2. NEXT←M[NEXT]

Cycle	1	2	1	2
Memory	Fetch X	Fetch NEXT	Fetch X	Fetch NEXT
Adder1	NEXT+1		NEXT+1	
Adder2	SUM		SUM	

- Optimised solution: 2. SUM←SUM+M[NEXT+1] 2. NEXT←M[NEXT]

Cycle	1	2	1	2
Memory	Fetch X	Fetch NEXT	Fetch X	Fetch NEXT
Adder1	SUM	NUMA	SUM	NUMA

Example: Optimised Solution 3



Another register, MUX, adder MUX and control added

Other optimisations possible:

Align data on even addresses
Use 16-bit wide memory

- Is this solution feasible:
  1. X←M[NUMA], NUMA←NEXT+1; T=0.5+10+1+1+0.5=14 ns, f=71 MHz
  2. NEXT← M[NEXT], SUM←SUM+X;