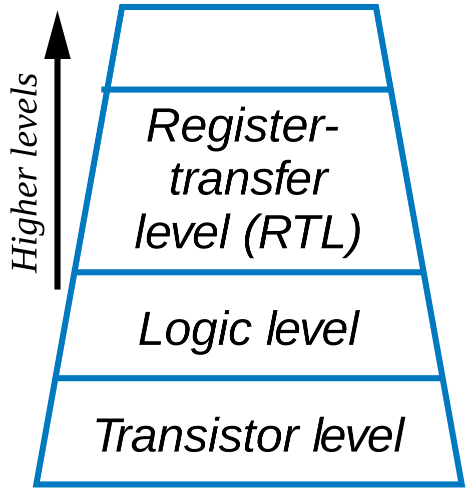
B38DB: Digital Design and Programming
Simple Data Path

Mustafa Suphi Erden

Heriot-Watt University
School of Engineering & Physical Sciences
Electrical, Electronic and Computer Engineering
Room: EM 2.01
Phone: 0131-4514159
E-mail: m.s.erden@hw.ac.uk

Introduction

- Chapter 2 (of the Text Book) *(covered)*
  - <u>Capture</u> Comb. behavior: Equations, truth tables
  - <u>Convert</u> to circuit: AND + OR + NOT → Comb. logic
- Chapter 3 *(covered)*
  - <u>Capture</u> sequential behavior: FSMs
  - <u>Convert</u> to circuit: Register + Comb. logic → Controller
- Chapter 4 *(have been and will be covering)*
  - Datapath components, **<u>simple datapaths</u>**
- Chapter 5 *(will touch, but topic of next semester)*
  - <u>Capture</u> behavior: High-level state machine
  - <u>Convert</u> to circuit: Controller + Datapath → Processor
  - Known as “RTL” (register-transfer level) design

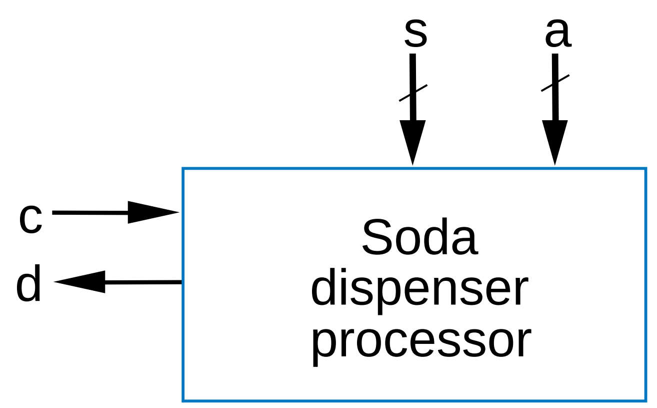


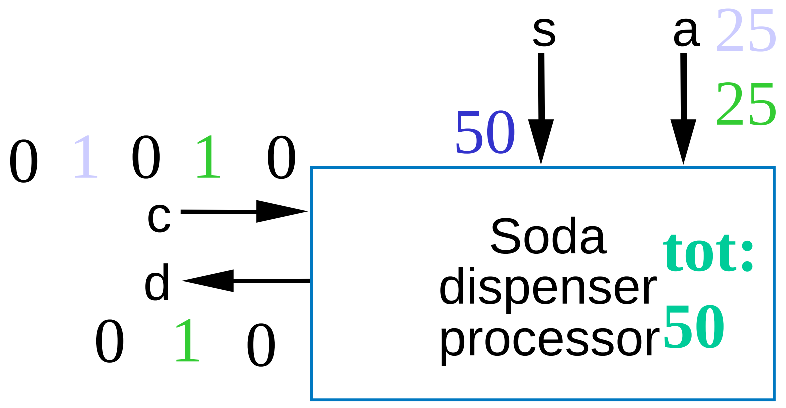
Levels of digital design abstraction

Processors:
- *Programmable (microprocessor)*
- *Custom*

High-Level State Machines (HLSMs)

- Some behaviors too complex for equations, truth tables, or FSMs
- Ex: Soda dispenser
  - *c*: bit input, 1 when coin deposited
  - *a*: 8-bit input having value of deposited coin
  - *s*: 8-bit input having cost of a soda
  - *d*: bit output, processor sets to 1 when total value of deposited coins equals or exceeds cost of a soda
- FSM can't represent…
  - 8-bit input/output
  - Storage of current total
  - Addition (e.g., 25 + 10)

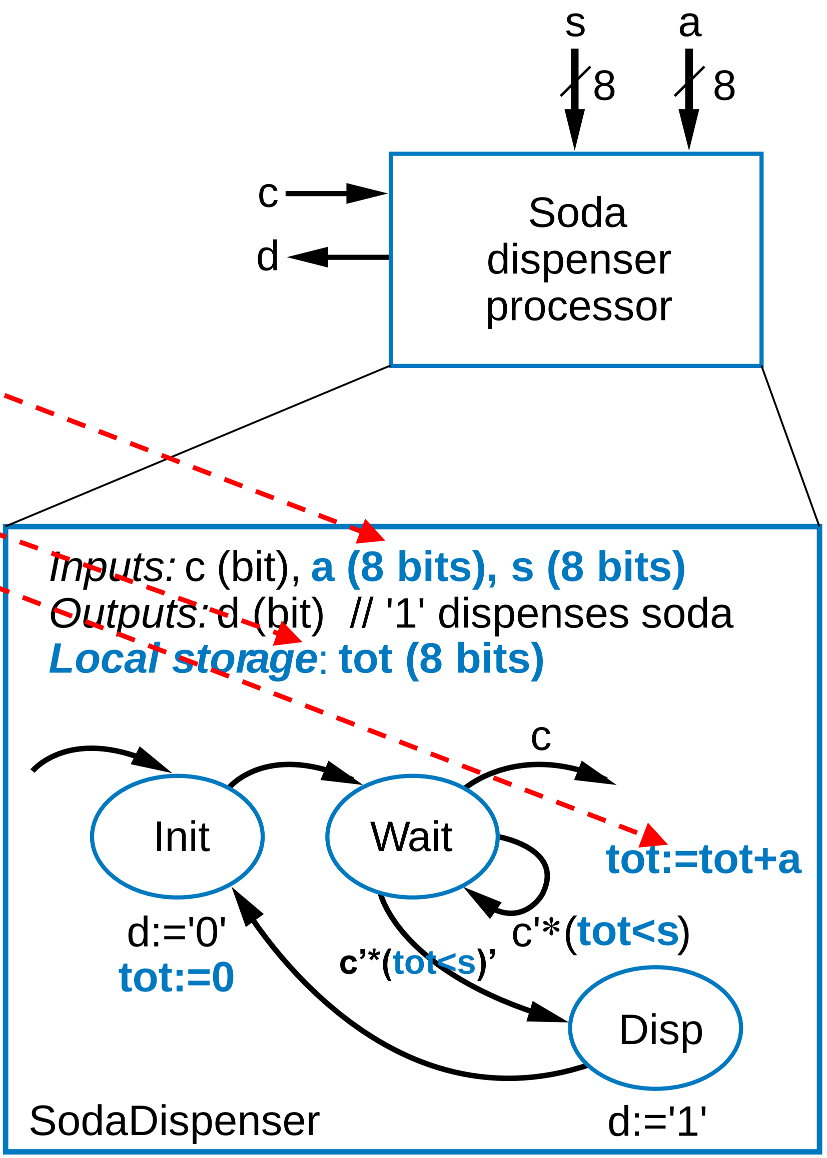




HLSMs

- High-level state machine (HLSM) extends FSM with:
  - Multi-bit input/output
  - Local storage
  - Arithmetic operations

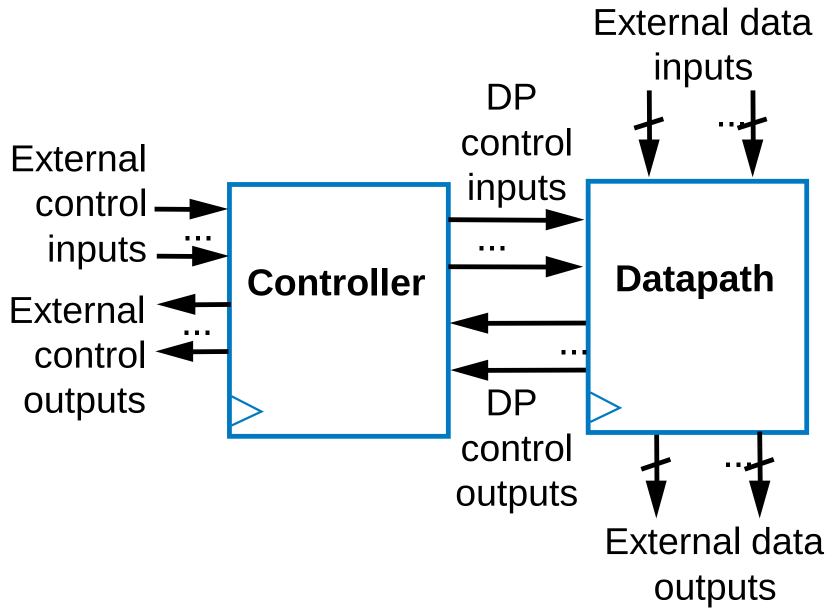
- Conventions
  - Numbers:
    - Single-bit: '0' (single quotes)
    - Integer: 0 (no quotes)
    - Multi-bit: “0000” (double quotes)
  - == for equal, := for assignment
  - Multi-bit outputs *must* be registered via local storage
  - // precedes a comment



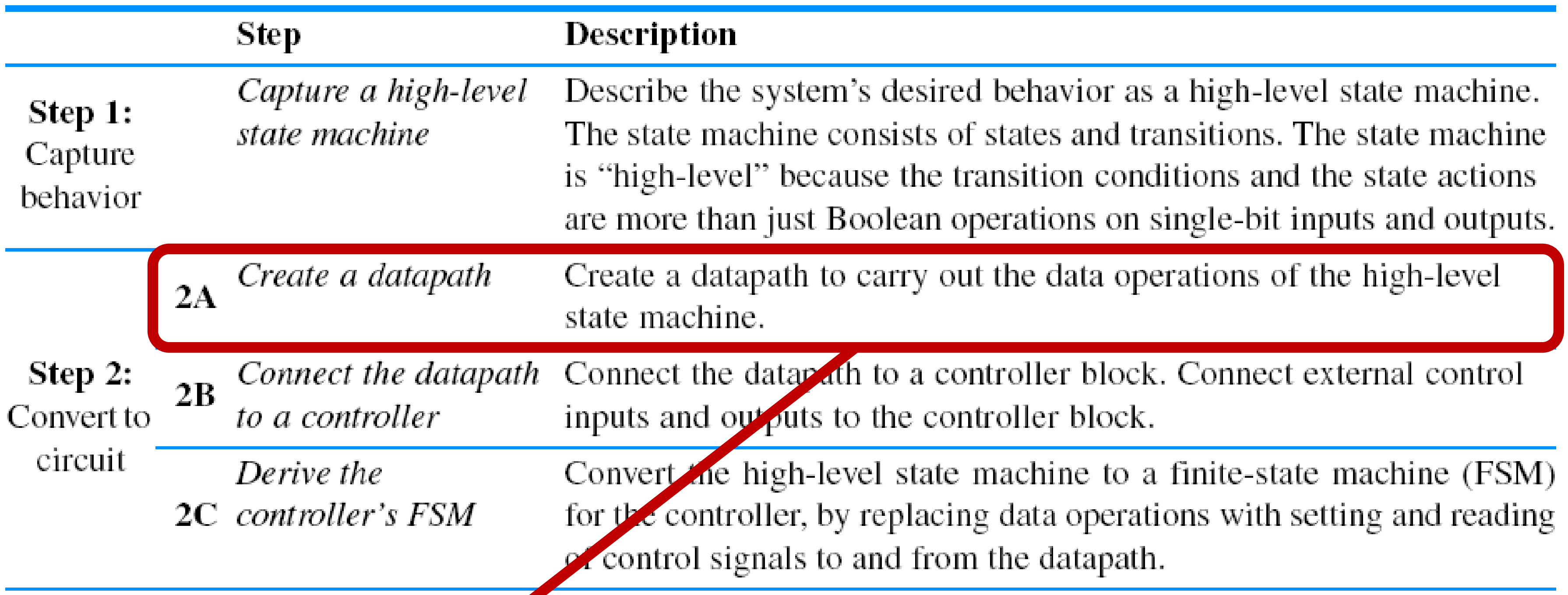
s
a
8
8
c
d
Soda dispenser processor
Inputs: c (bit), a (8 bits), s (8 bits)
Outputs: d (bit) // '1' dispenses soda
Local storage: tot (8 bits)
Init
Wait
Disp
c
tot:=tot+a
d:='0'
tot:=0
c'*(tot<s)
c'*(tot<s)'
SodaDispenser
d:='1'

RTL Design Process

- Capture behavior
- Convert to circuit
  - Need target architecture
  - Datapath capable of HLSM's data operations
  - Controller to control datapath



RTL Design Process

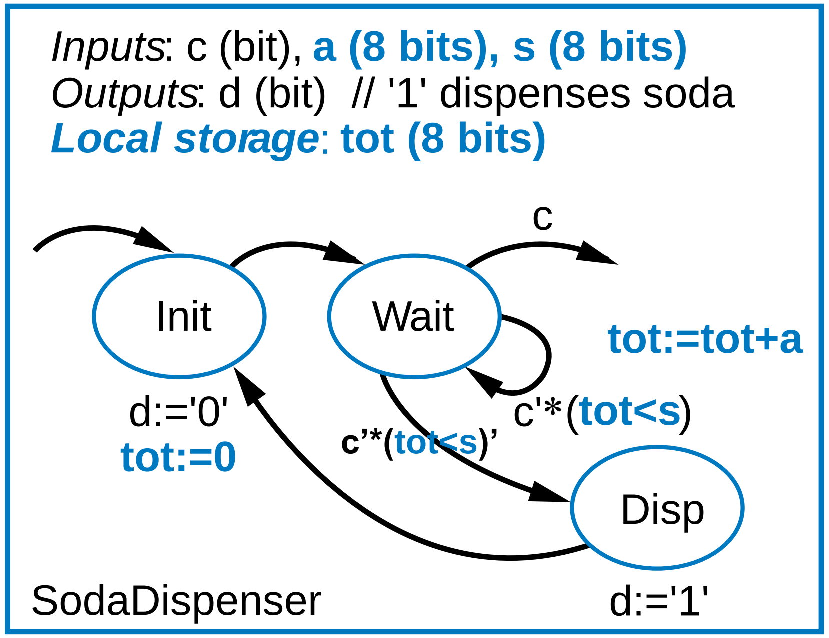


		Step	Description
Step 1: Capture behavior		*Capture a high-level state machine*	Describe the system's desired behavior as a high-level state machine. The state machine consists of states and transitions. The state machine is "high-level" because the transition conditions and the state actions are more than just Boolean operations on single-bit inputs and outputs.
Step 2: Convert to circuit	**2A**	*Create a datapath*	Create a datapath to carry out the data operations of the high-level state machine.
	2B	*Connect the datapath to a controller*	Connect the datapath to a controller block. Connect external control inputs and outputs to the controller block.
	2C	*Derive the controller's FSM*	Convert the high-level state machine to a finite-state machine (FSM) for the controller, by replacing data operations with setting and reading of control signals to and from the datapath.

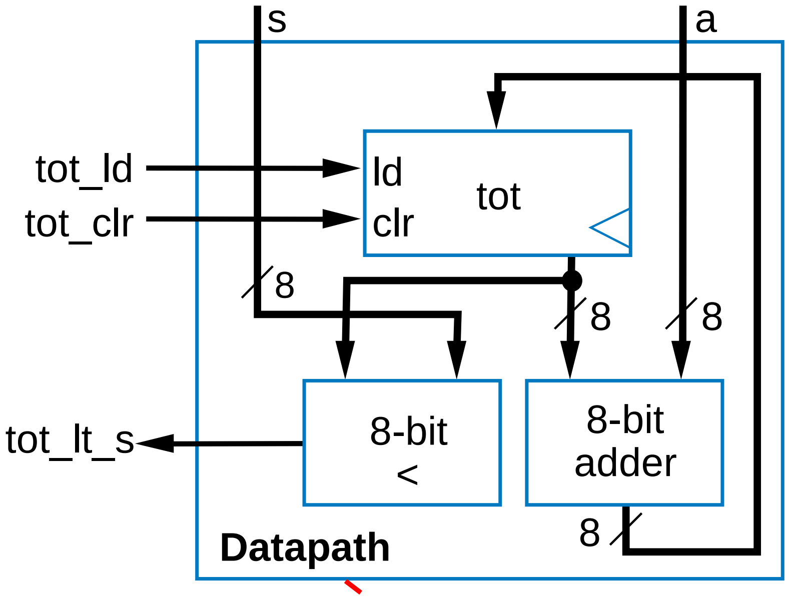
Datapath is our topic!

Example: Soda Dispenser

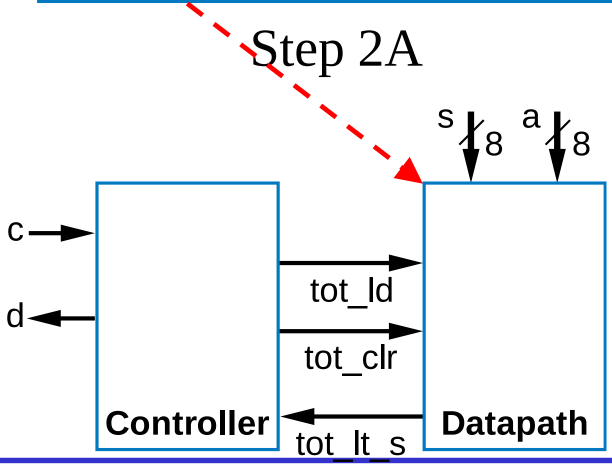
- Quick overview example.


Inputs: c (bit), **a (8 bits), s (8 bits)**
Outputs: d (bit) // '1' dispenses soda
***Local storage*: tot (8 bits)**
Init
Wait
Disp
c
tot:=tot+a
d:='0'
tot:=0
c'*(tot<s)'
c'*(tot<s)
d:='1'
SodaDispenser

Step 1



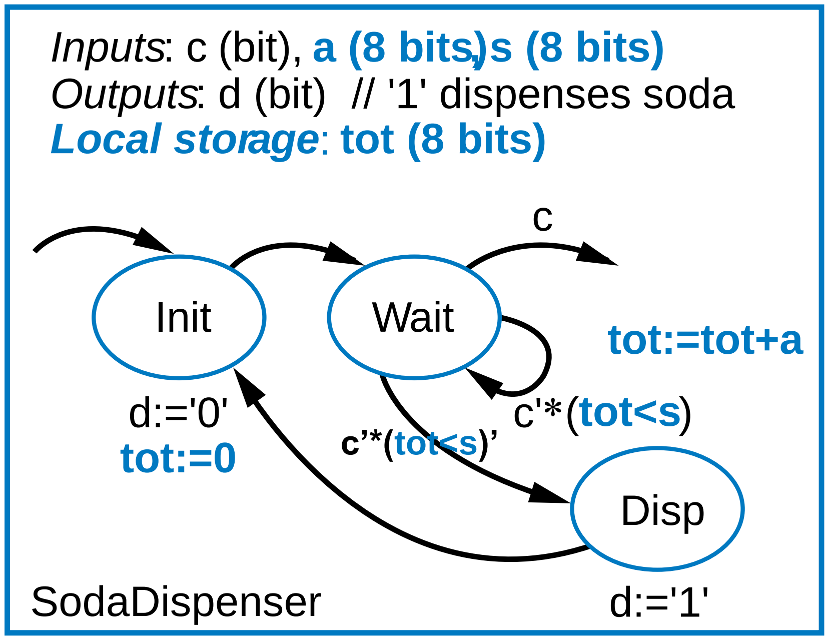
Step 2A



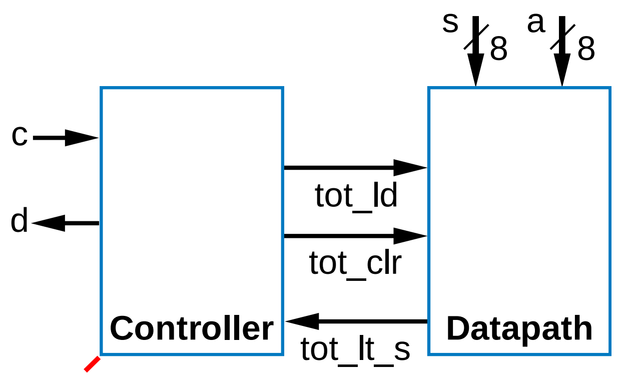
Step 2B

Example: Soda Dispenser

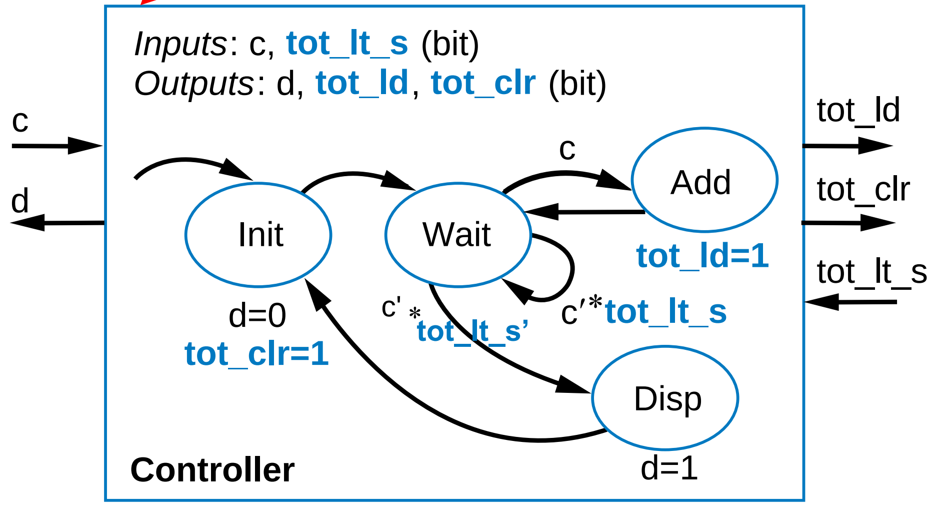
- Quick overview example.


Inputs: c (bit), a (8 bits), s (8 bits)
Outputs: d (bit) // '1' dispenses soda
Local storage: tot (8 bits)
Init
Wait
Disp
c
tot:=tot+a
c'*(tot<s)
c'*(tot<s)'
d:='0'
tot:=0
d:='1'
SodaDispenser

Step 1



Step 2B

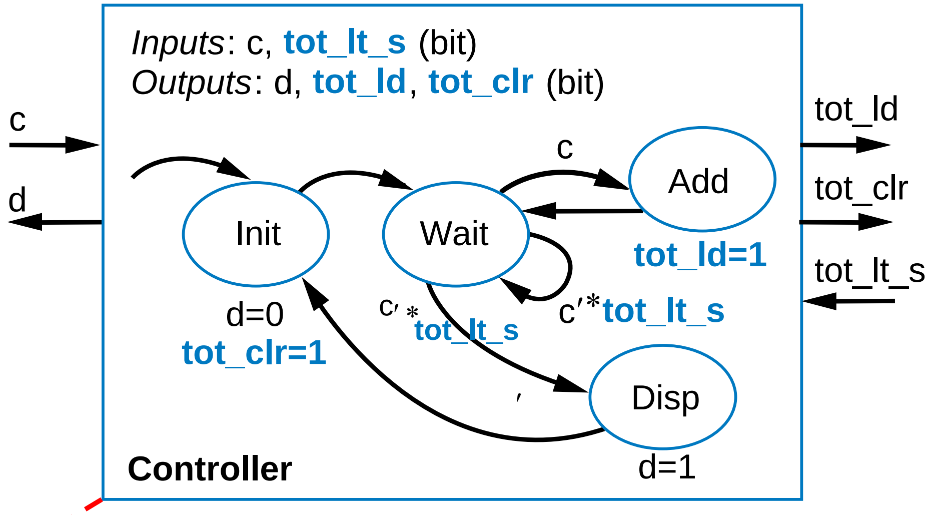


Step 2C

Example: Soda Dispenser

- Quick overview example.

	s1	s0	c	tot_lt_s	n1	n0	d	tot_ld	tot_clr
Init	0	0	0	0	0	1	0	0	1
	0	0	0	1	0	1	0	0	1
	0	0	1	0	0	1	0	0	1
	0	0	1	1	0	1	0	0	1
Wait	0	1	0	0	1	1	0	0	0
	0	1	0	1	0	1	0	0	0
	0	1	1	0	1	0	0	0	0
	0	1	1	1	1	0	0	0	0
Add	1	0	0	0	0	1	0	1	0
		• • •				• • •			
Disp	1	1	0	0	0	0	1	0	0
		• • •				• • •			



Step 2C

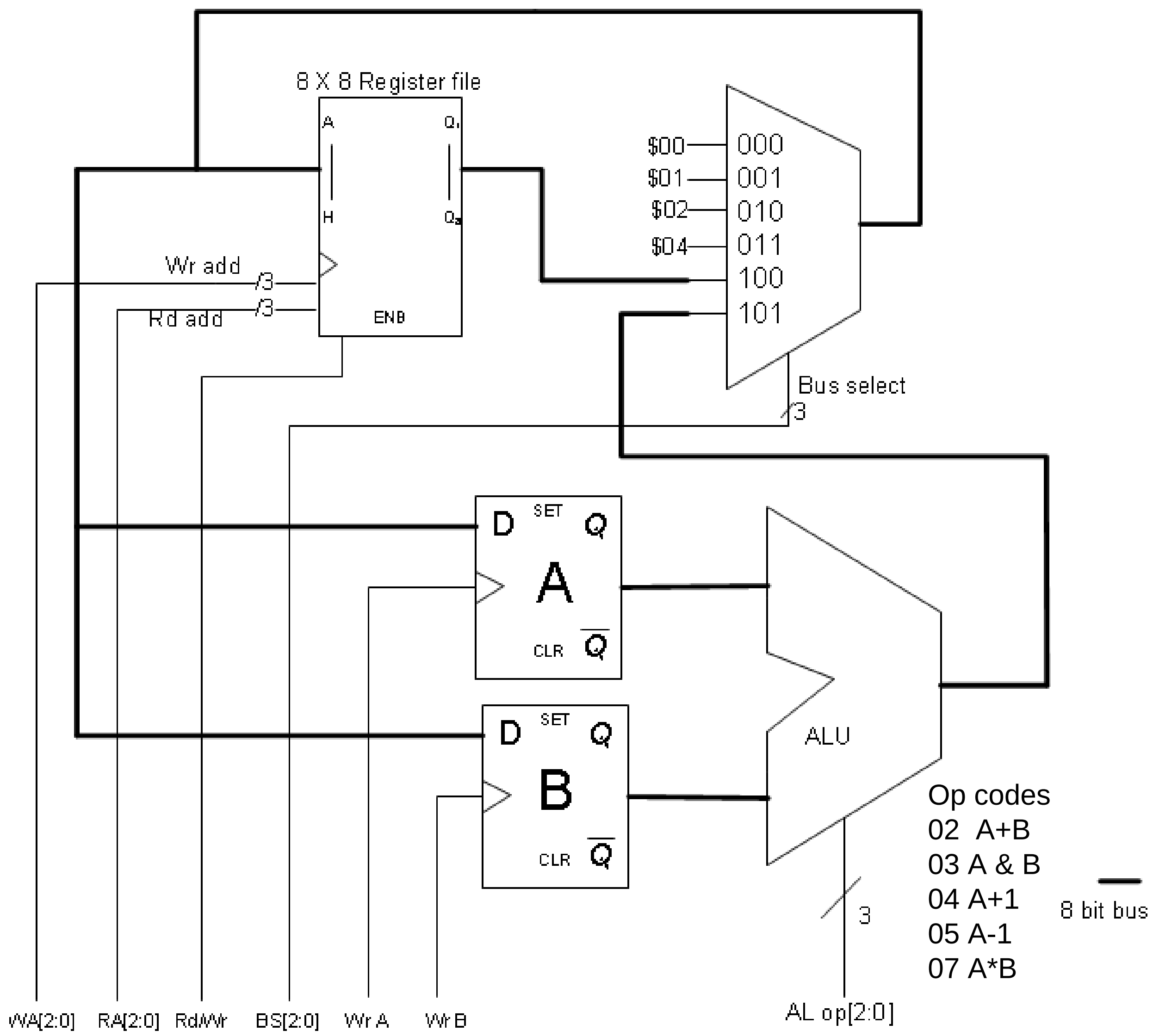
Controller design process (Ch3) applied to complete the design.

Simple Data Path

It will consist of

1. <u>Memory</u> as a register file
2. A <u>bus selector</u> (multiplexer) to select data paths
3. An <u>arithmetic-logic-unit</u> that provides simple computation

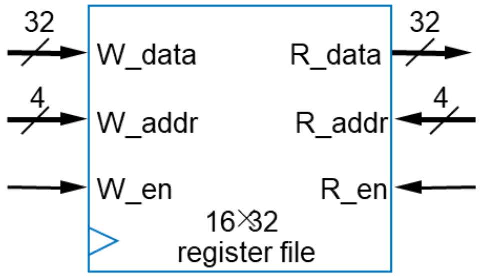
Data Path

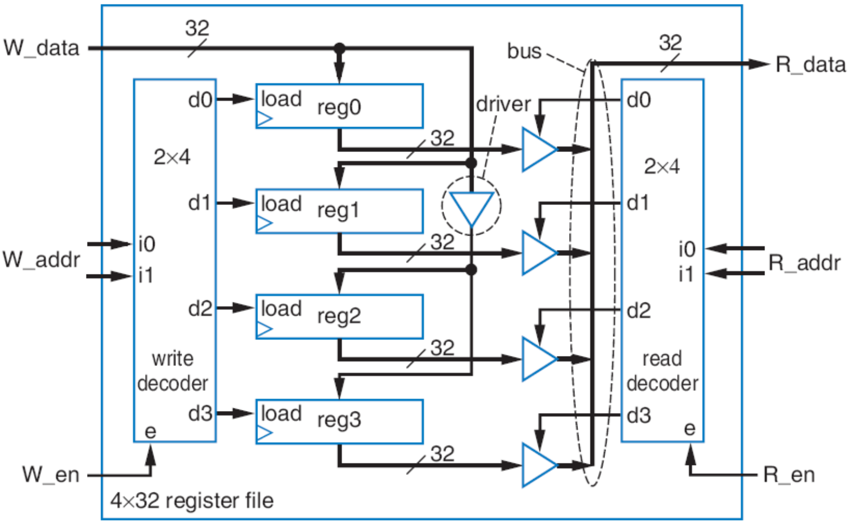


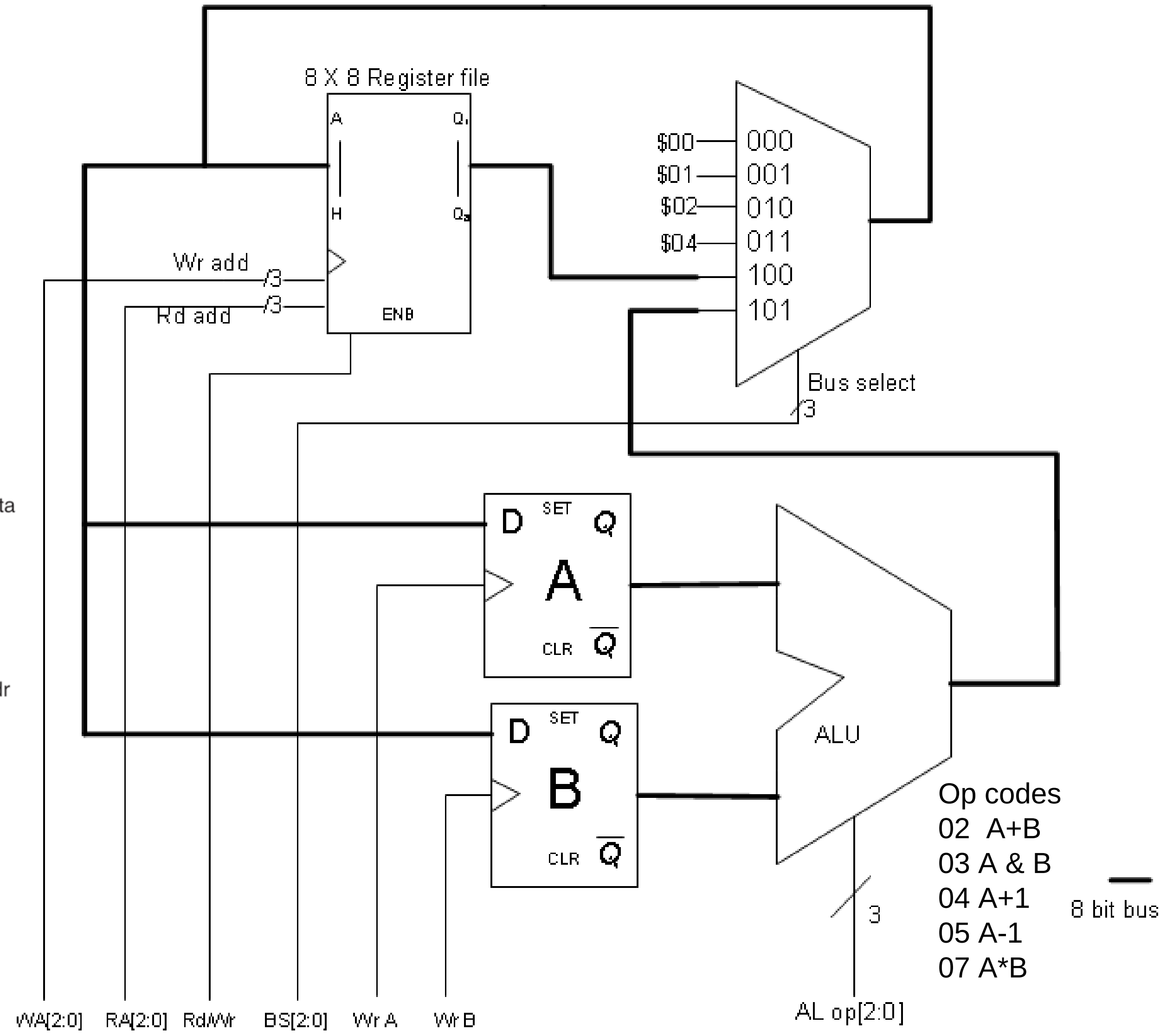
Op codes

02	A+B
03	A & B
04	A+1
05	A-1
07	A*B

Register File in Data Path

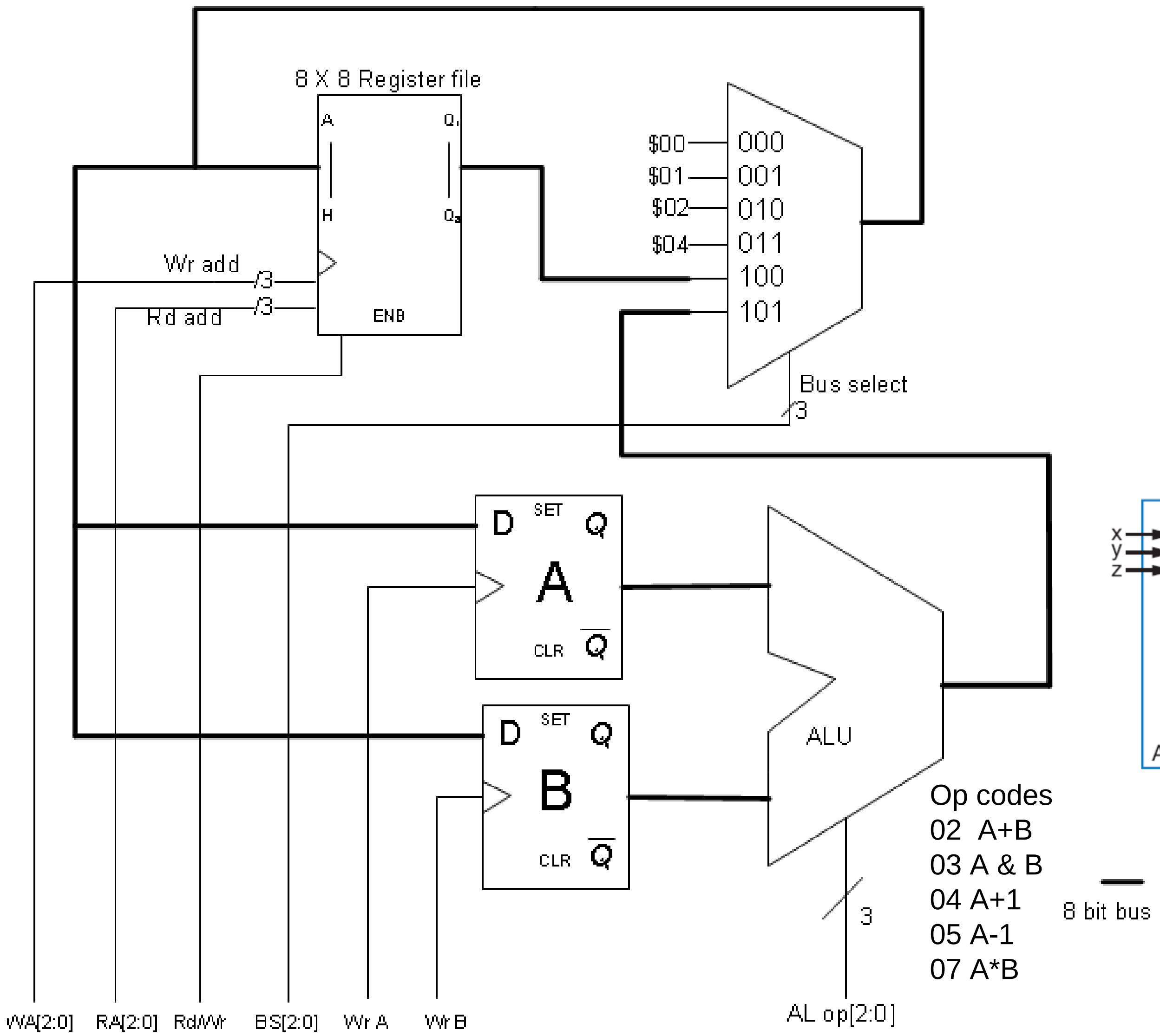




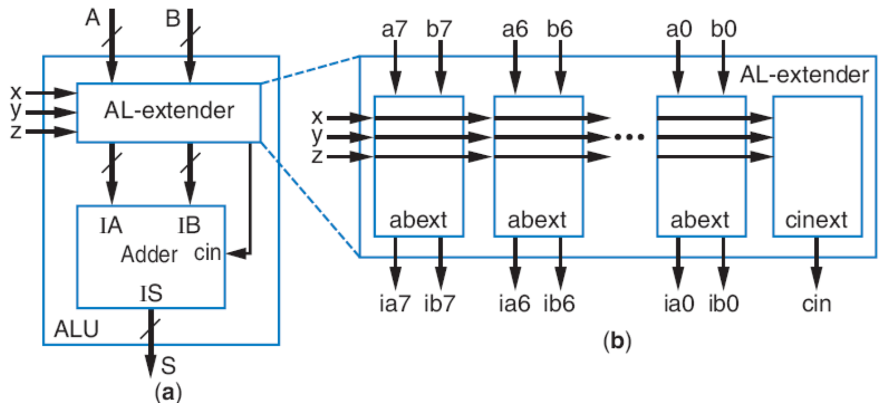


Op codes
02 A+B
03 A & B
04 A+1
05 A-1
07 A*B

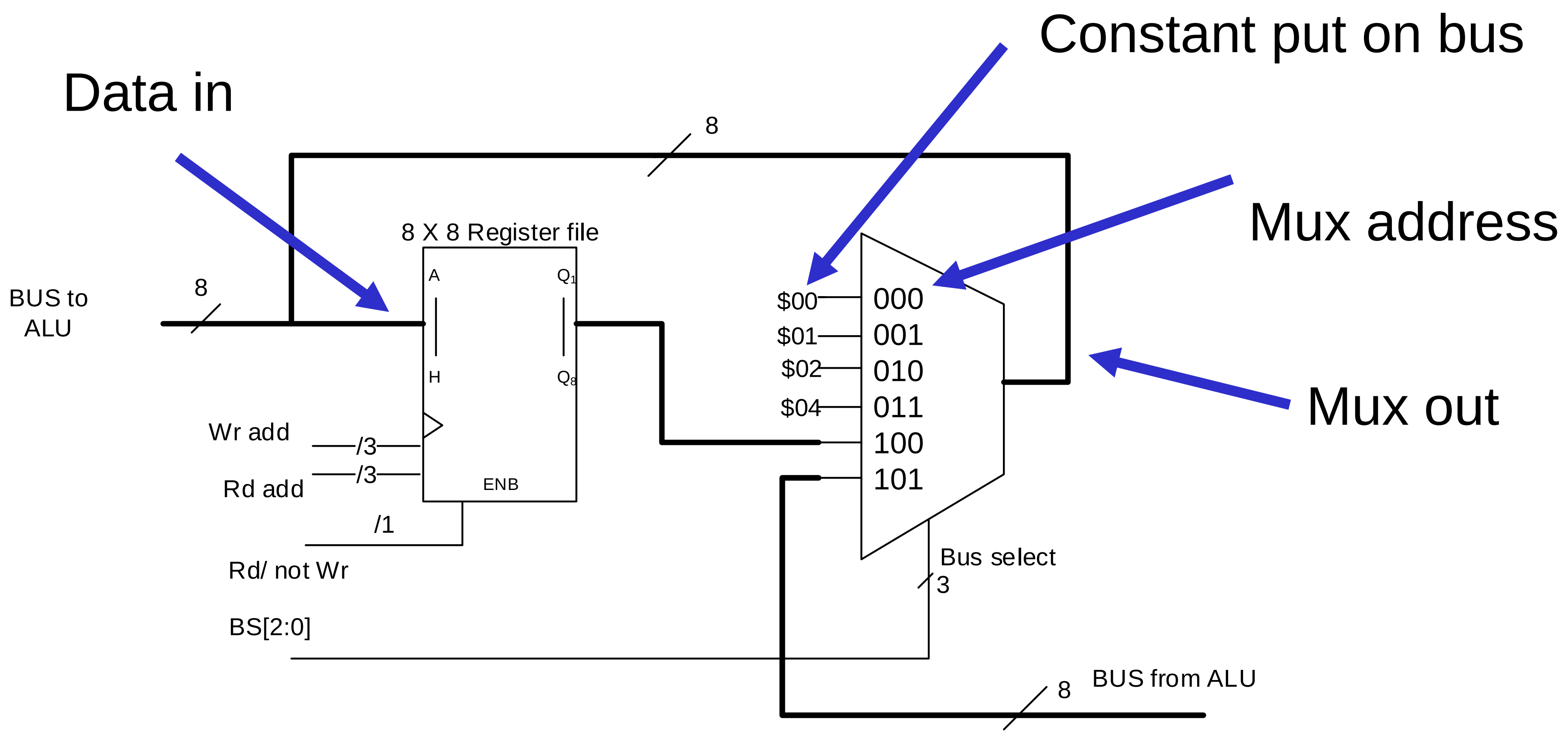
ALU in Data Path



Op codes
02 A+B
03 A & B
04 A+1
05 A-1
07 A*B

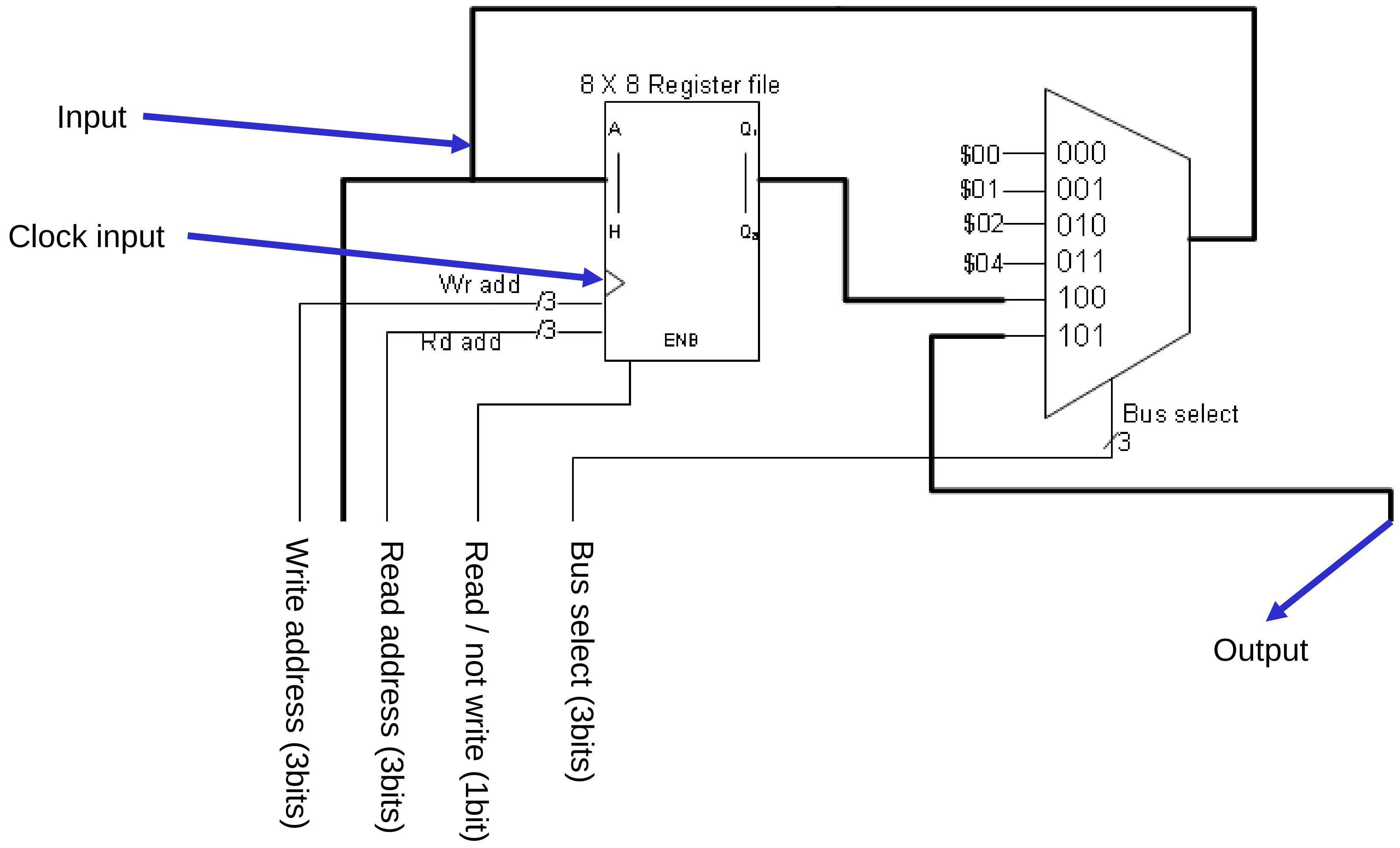


Register - Bus Selector

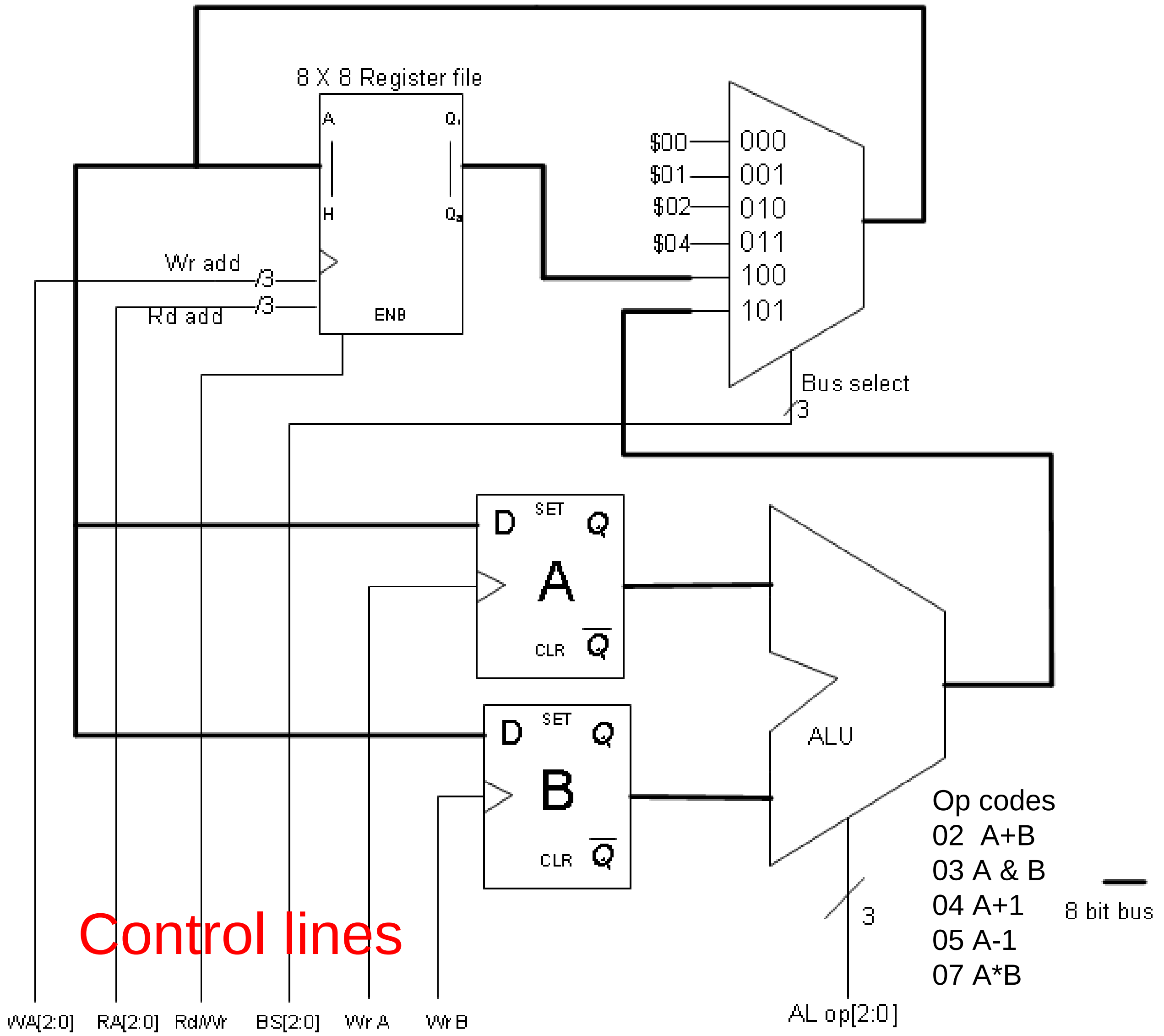


What could you add to allow data to come in from outside the data path ?

Register - Multiplexer



Control Lines



What sequence of the control lines is needed to move data through this path ?

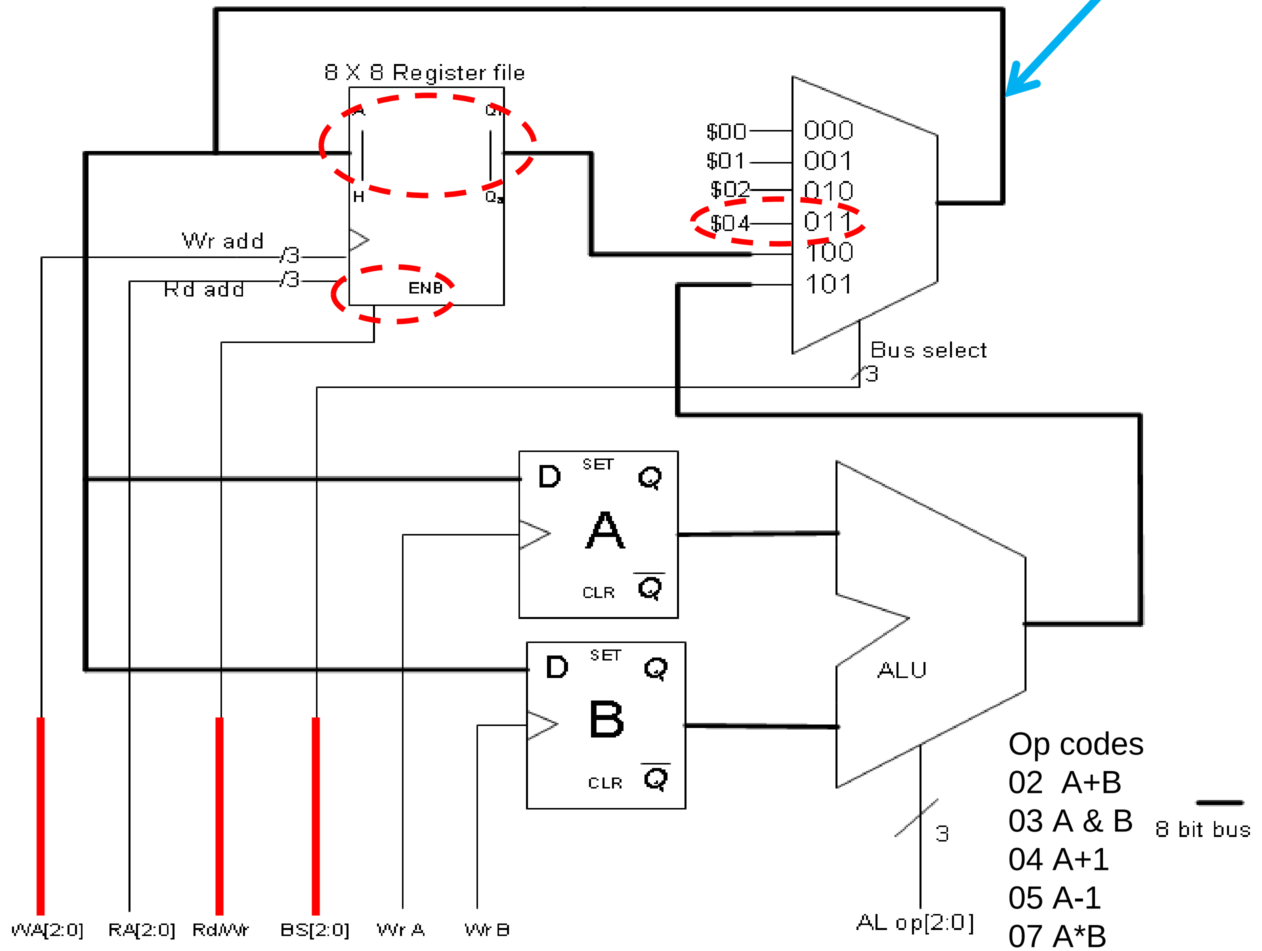
Load Data Into Register

8 bit data bus

Operation : R1 = 4

Sequence:

1. Set BS = 3
2. Set WA = 1
3. Set Rd/Wr = 0
4. Set Rb/Wr = 1



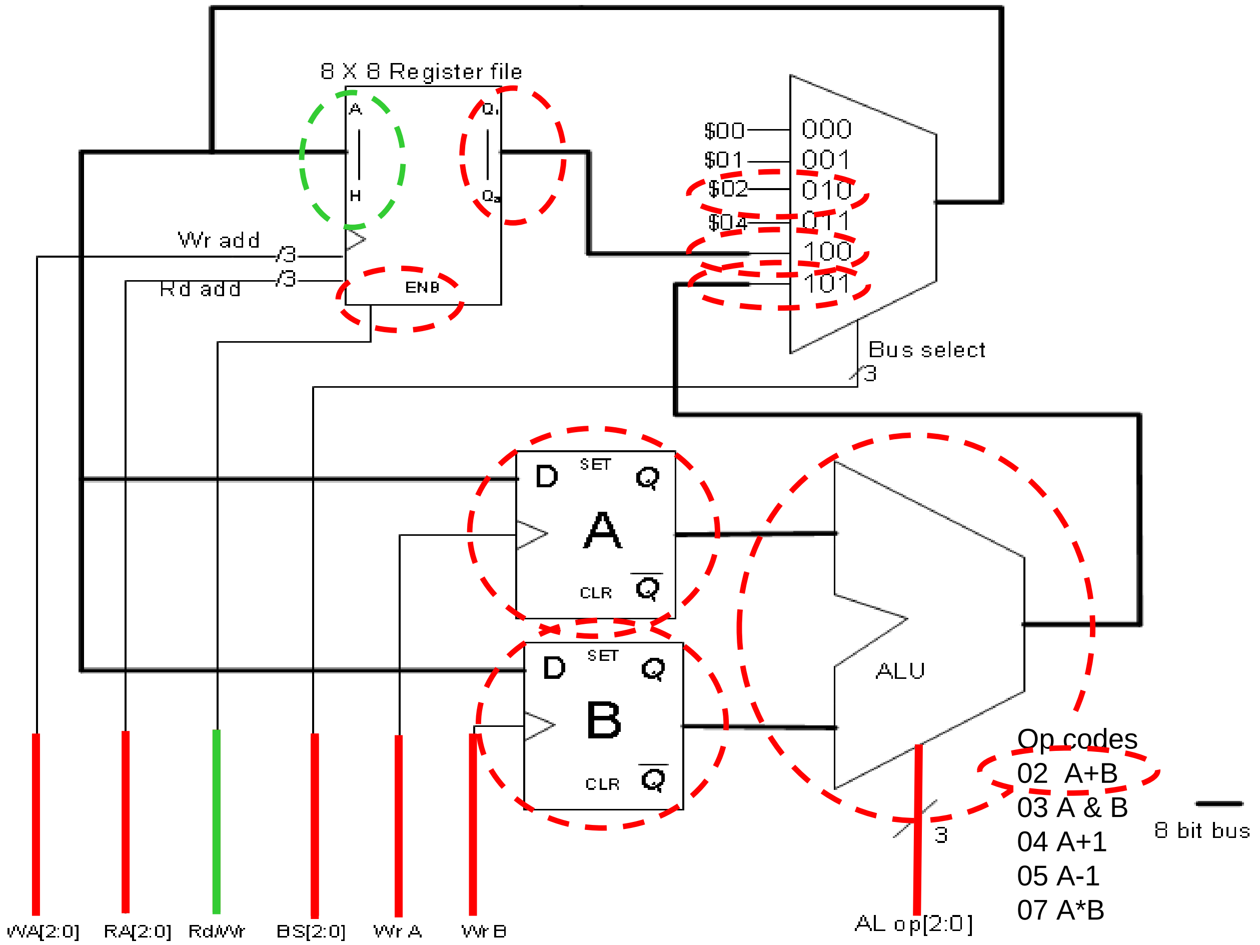
Op codes
02 A+B
03 A & B
04 A+1
05 A-1
07 A*B

ALU Operation – Arithmetic

Operation : R2 = R3 + 2

Sequence:

1. Set BS = 4 (reg out connected to bus)
2. Set RA = 3 (bus connected to reg out connected to R3)
3. Set Rd/Wr = 1 (read R3)
4. Set WrA = 1 (write R3 contents into A), then Set WrA = 0
5. Set BS = 2 (output constant to bus)
6. Set Wr B = 1 (write constant 2 into B) then Set WrB = 0
7. Set AL op =2 (add A + B)
8. Set WA = 2
9. Set BS = 5 (alu result on bus)
10. Set Rd/Wr = 0 (write Alu output to R2)

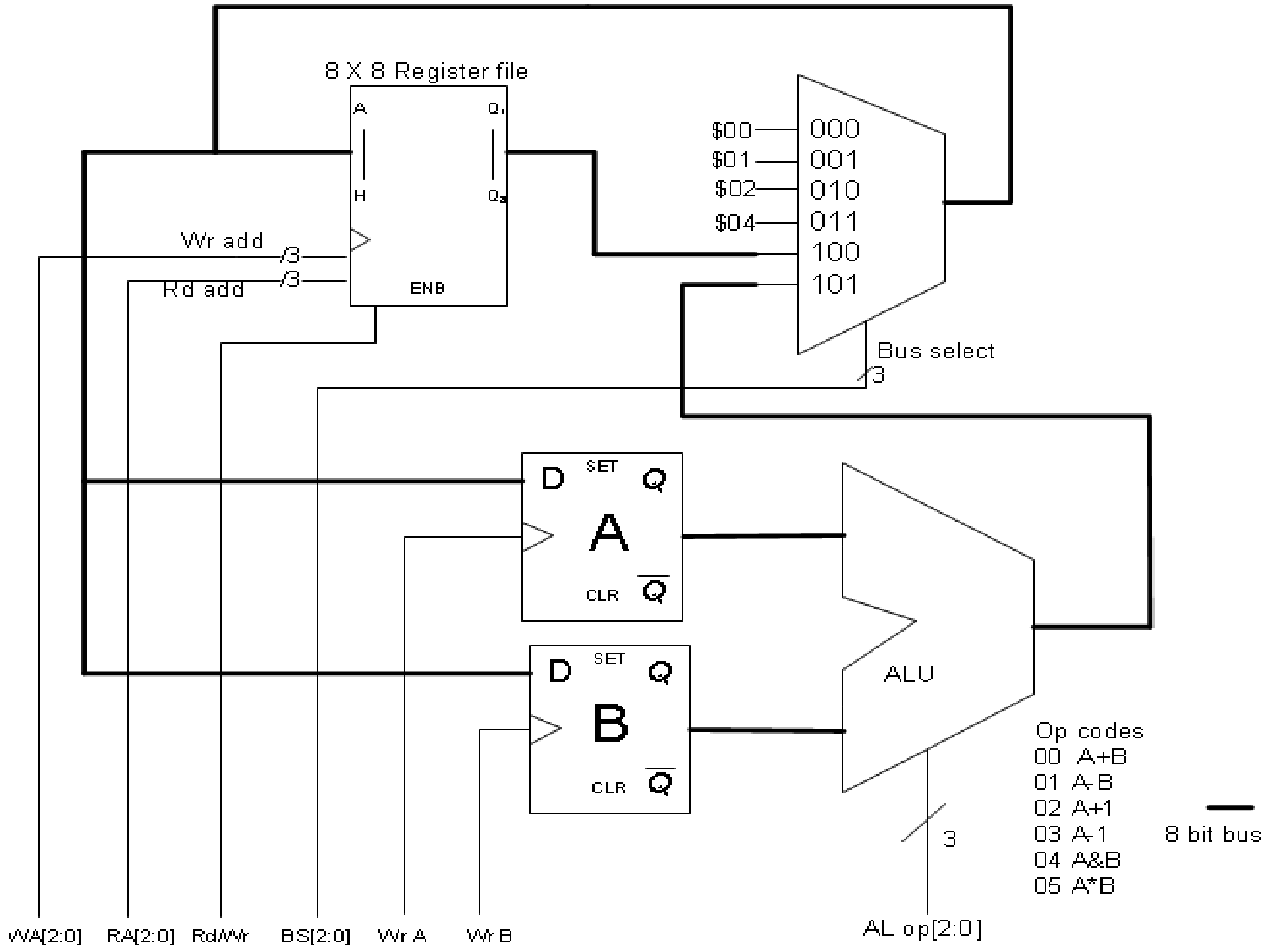


Move Data Between Registers

Operation : R2 = R3

Sequence:

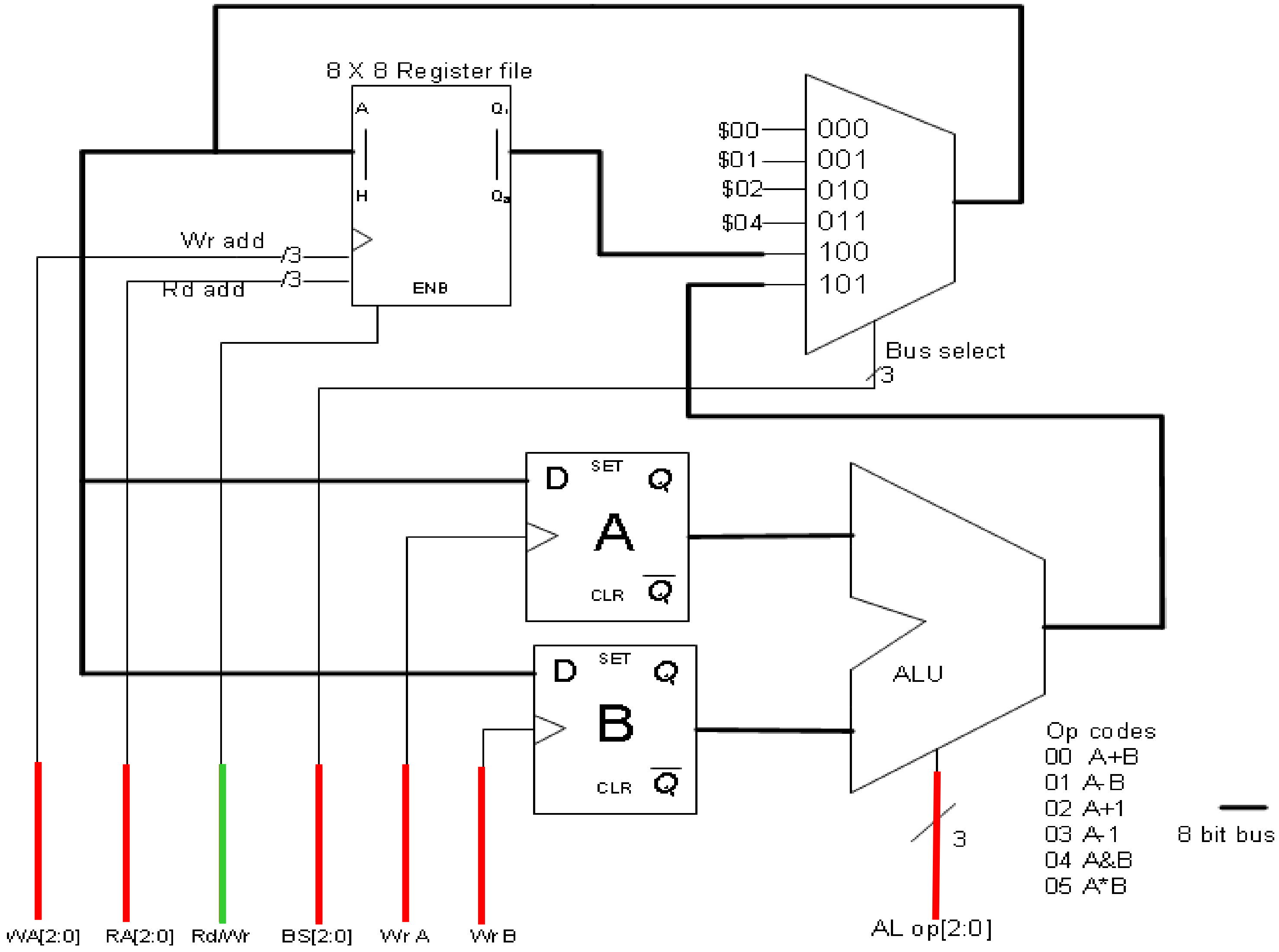
1. Set BS = 4 (reg out connected to bus)
2. Set RA = 3 (bus connected to R3)
3. Rd/Wr = 1 (read R3)
4. Set WA = 2 (bus connected to R2)
5. Rd/Wr = 0 (write R2)



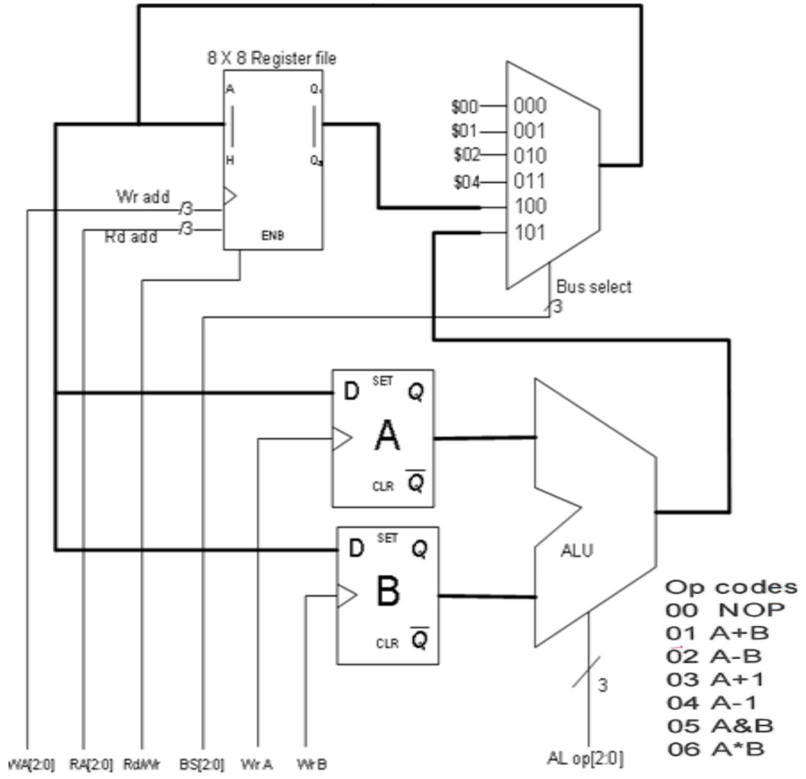
ALU Operation – Logic

Operation : R2 = R2 & 2

Sequence
1. Set BS = 4 (reg out connected to bus)
2. Set RA = 2 (bus connected to reg out connected to R2)
3. Set Rd/Wr = 1 (read R2)
4. Set WrA = 1 (write contents R2 to A) then Set WrA =0
5. Set BS = 2
6. Set WrB = 1 Set Wr B =0
7. Set ALU op = 4
8. Set WA = 2
9. Rd/Wr = 0 (write R2)



Process Tables – Example 1



Process tables

Process tables provide a similar means of describing what is occurring in the CPU datapath. The idea is to use a column in the table to stipulate what is occurring on each control line.

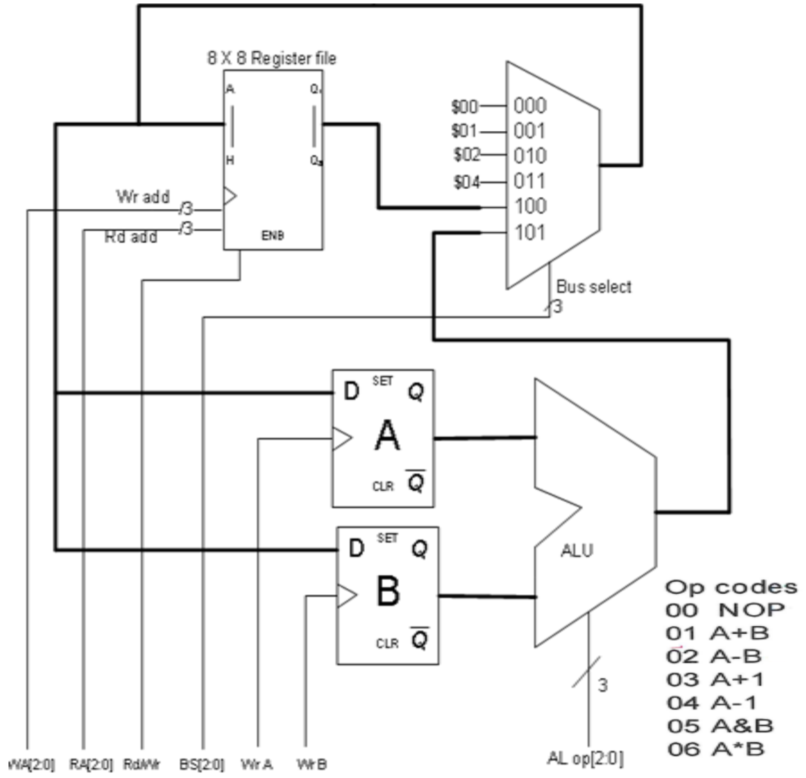
Consider the operation: R5 = R2 & R3.

Cycle	WA[2:0]	RA[2:0]	Rd/Wr	BS[2:0]	wrA	wrB	ALU op	Operation
0		010	1	100	1	0		A = R2 (read from register memory, RA=010 and write to A).
1		011	1	100	0	1		B = R3 (read from register memory, RA=011 and write to B).
2	101		0	101	0	0	101	R5 = R2 & R3 (perform AND operation, ALU op = 101 and write to R5, WA = 101, BS = 101).

Process Tables – Example 2

Consider the operation: R2 = R3 * R4.

Note This is very similar to the previous example. All that changed were the WA and RA addresses (to reflect different register memory addresses) and the ALU op (to reflect multiplication rather than the AND operation)

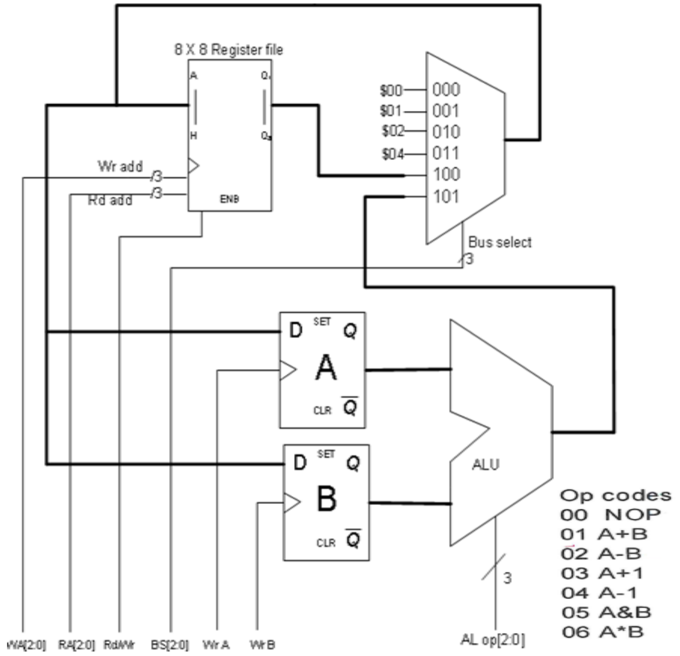


Cycle	WA[2:0]	RA[2:0]	Rd/Wr	BS[2:0]	wrA	wrB	ALU op	Operation
0		011	1	100	1	0		A = R3.
1		100	1	100	0	1		B = R4.
2	010		0	101	0	0	110	R2 = R3 * R4.

Process Tables – Example 3

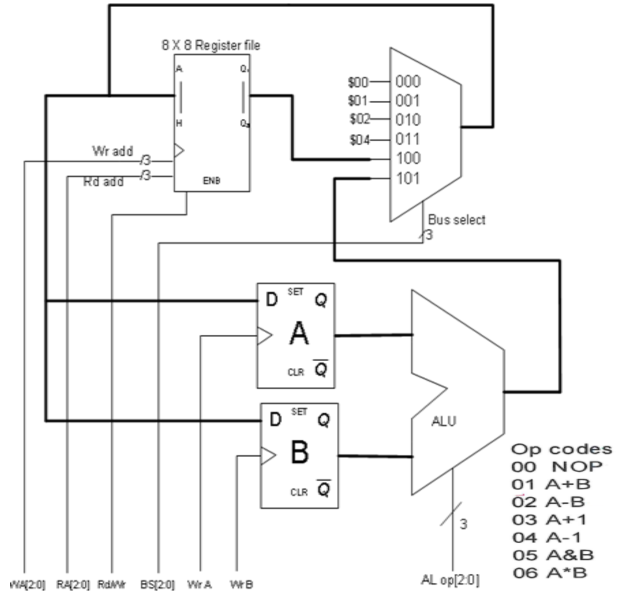
Consider the operation: R1 = R1 – 9.

Note 9 is larger than any of the nominal numbers provided, i.e., 0,1,2,4 thus it will be necessary to call these numbers numerous times using multiple instances of bus select.



Cycle	WA[2:0]	RA[2:0]	Rd/Wr	BS[2:0]	wrA	wrB	ALU op	Operation
0		001	1	100	1	0		A = R1
1				011	0	1		B = 4
2				011	0	0	010	ALUo = R1 – 4
3				011	1	0	010	A=R1 – 4
4				011	0	1	000	B = 4
5				101	0	0	010	ALUo = R1 – 4 – 4
6				101	1	0	010	A = R1 – 4 – 4
7				001	0	1	000	B = 1
8	001			101	0	0	010	R1 = R1 - 9

Process Tables – Example 3



Process tables

Consider the operation: R7 = R7^4 . In this example, the activation of WrA and WrB simultaneously is allowed.

Cycle	WA[2:0]	RA[2:0]	Rd/Wr	BS[2:0]	wrA	wrB	ALU op	Operation
0		111	1	100	1	1		A = B = R7 . BS set as 100 is telling RF to output R7 (RA = 111) into both A (wrA = 1) and B (wrB = 1).
1			1	101	0	0	110	A = R7 * R7 . A and B are being multiplied together using ALUop = 110 with the answer being written to A using BS = 101.
2		111	1	101	1	1		A = B = R7^2
3	111		0	101	0	0	110	R7 = R7^4 Write the answer to 111 using WA = 111, i.e., R7.