CS 110
Computer Architecture
Datapath

Instructors:
Chundong Wang, Siting Liu & Yuan Xiao
Course website: https://toast-lab.sist.shanghaitech.edu.cn/courses/CS110@ShanghaiTech/Spring-2025/index.html
School of Information Science and Technology (SIST)
ShanghaiTech University

2024/3/25

Administratives

- Lab 5 available, please prepare in advance, to check this week! Lab 6 released!
- HW 3 available, ddl April 1st, start early!
- Proj 1.1 ddl TODAY, Mar. 27th!
- Proj 1.2 will be released.
- Discussion this week on digital circuits.

Mid-term I

- Midterm I
  - April 10th 8:00 am - 10:00 am
    - We start sharp at 8:00 am!
    - Arrive 7:45 am to check-in (Venue: TBD on your egate system; Seat: TBD on-site)
    - Arrive later then 8:30 am will get 0 mark.
- Contents:
  - Everything till April 8th lecture
- Switch cell phones **off**!!! (not silent mode)
  - Put them in your bags.
- Bags in the front. On the table: nothing but pen, exam paper, 1 drink, 1 snack, your student ID card and your cheat sheet!

Mid-term I requirements

- You can bring a cheatsheet (**handwritten only**). **1**-page **A4**, **double-sided** (2-page for the mid-term II and 3-page for the final). Put it on your desk at exam. Cheatsheet that does not apply to the rules would be taken away.
- Greencard shown on the course website is provided with the exam paper.
- No other electronic devices are allowed!
  - No ear plugs, music, smartwatch, calculator, computer…
- Anybody touching any electronic device will **FAIL** the course!
- Anybody found cheating (copy your neighbors answers, additional material, ...) will **FAIL** the course!

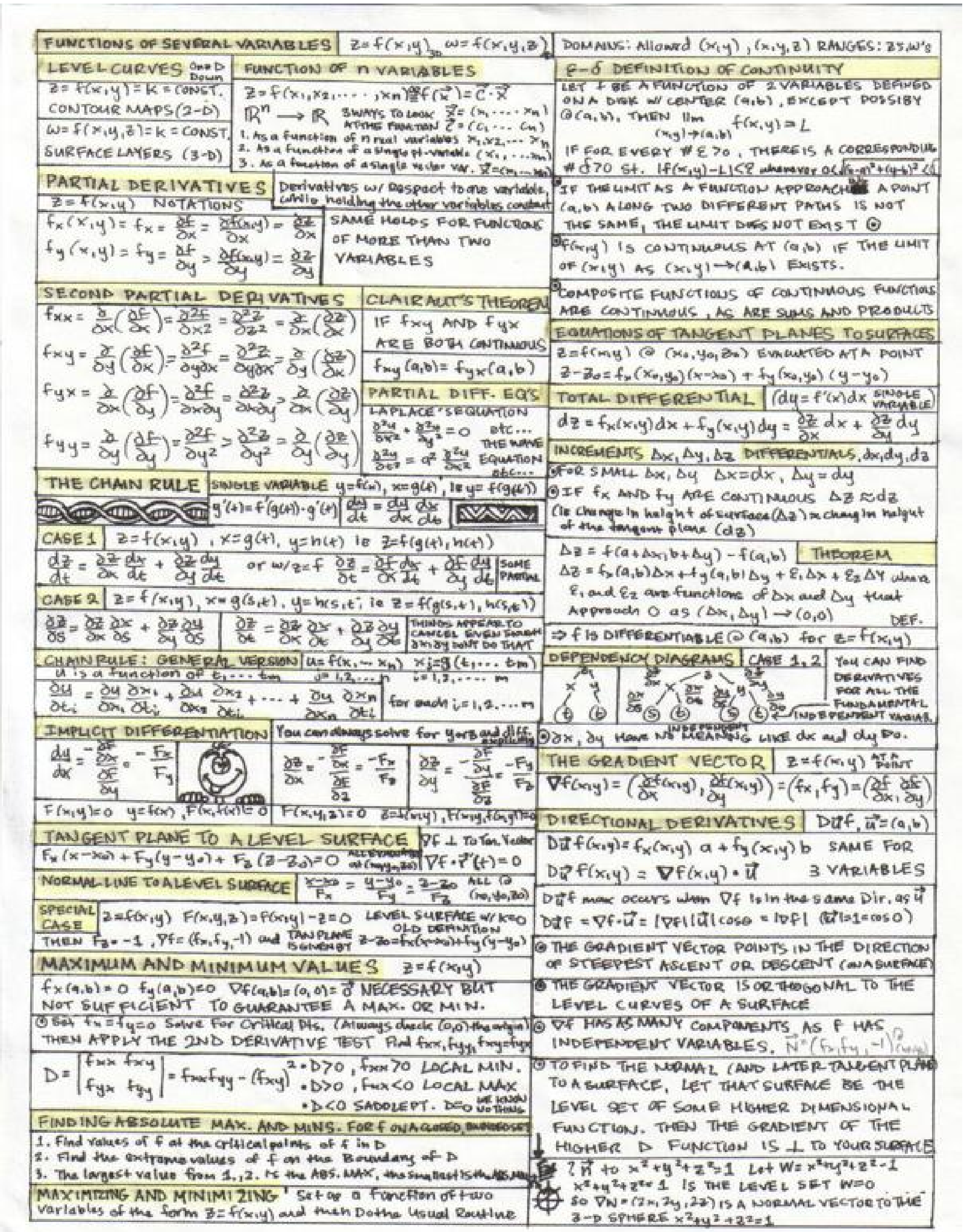
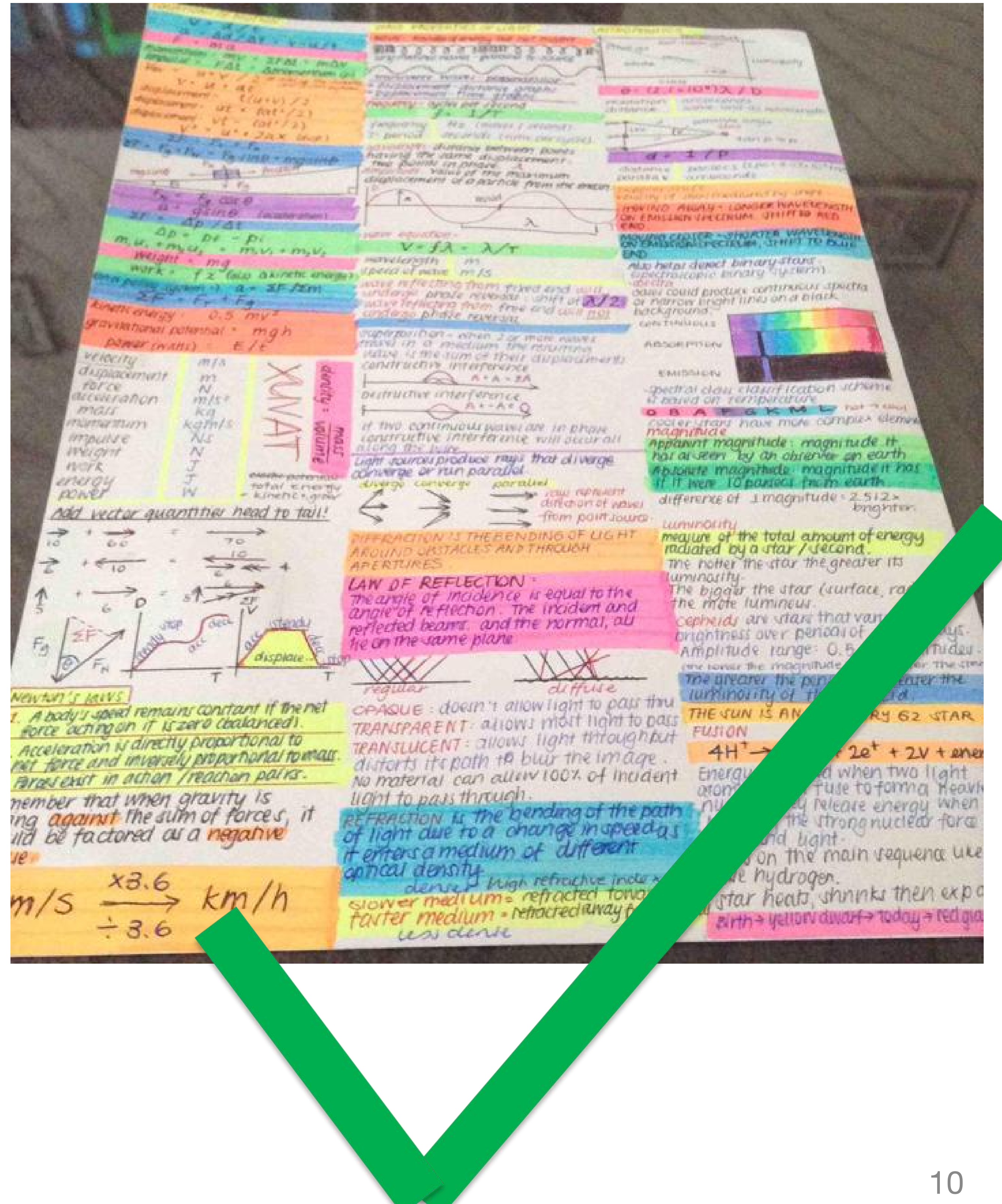

CASIO
JF-120TM
TWO WAY POWER
12 DIGITS
F CUT 5/4
4 3 2 1 0
+/-
MRC
TAX-
TAX RATE
TAX+
RATE SET %
GT
C
ON
AC
7
4
9
6
1
3
0
00
×
−
=
+

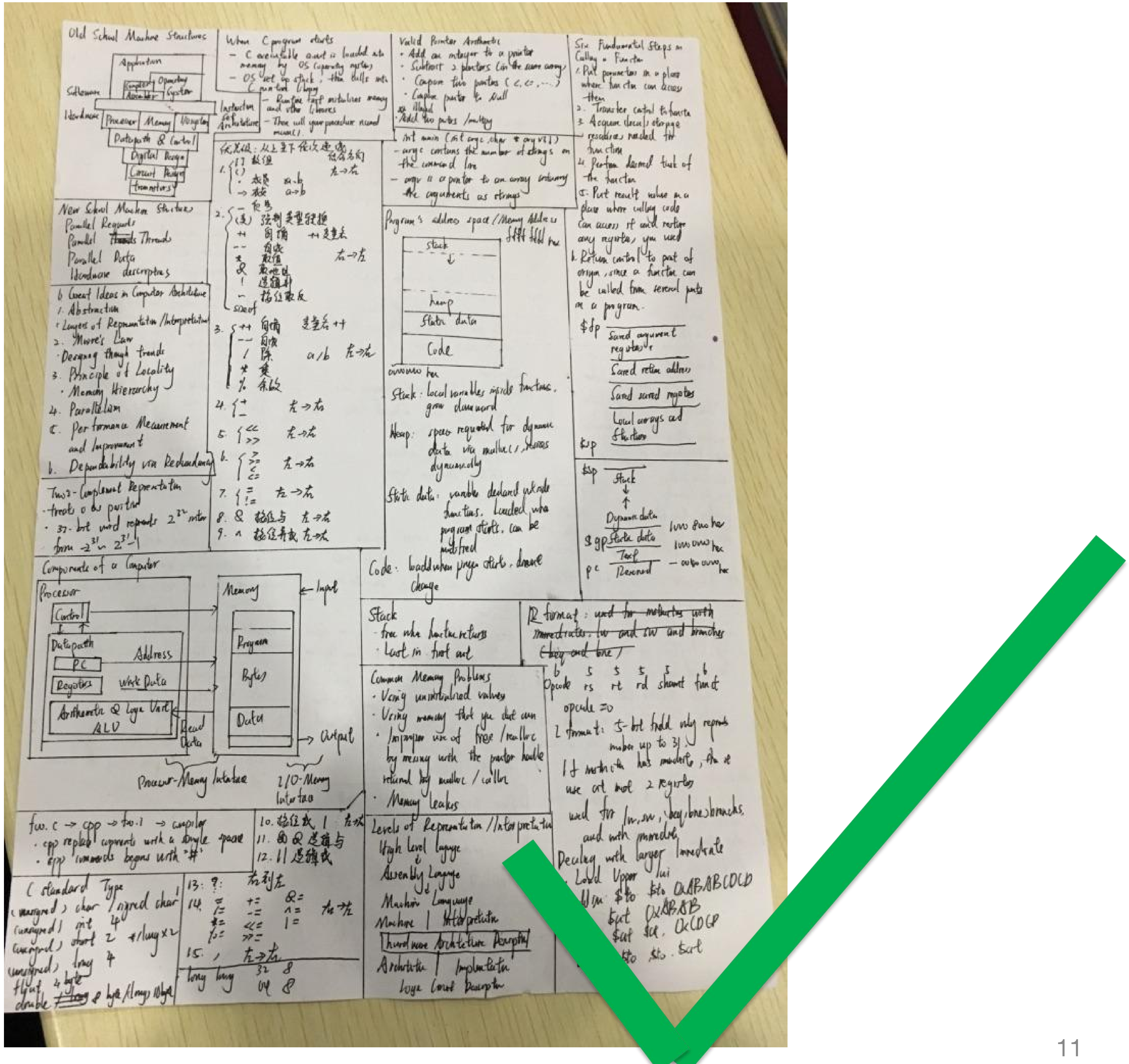

08:08
Thu, Sep 1
HUAWEI

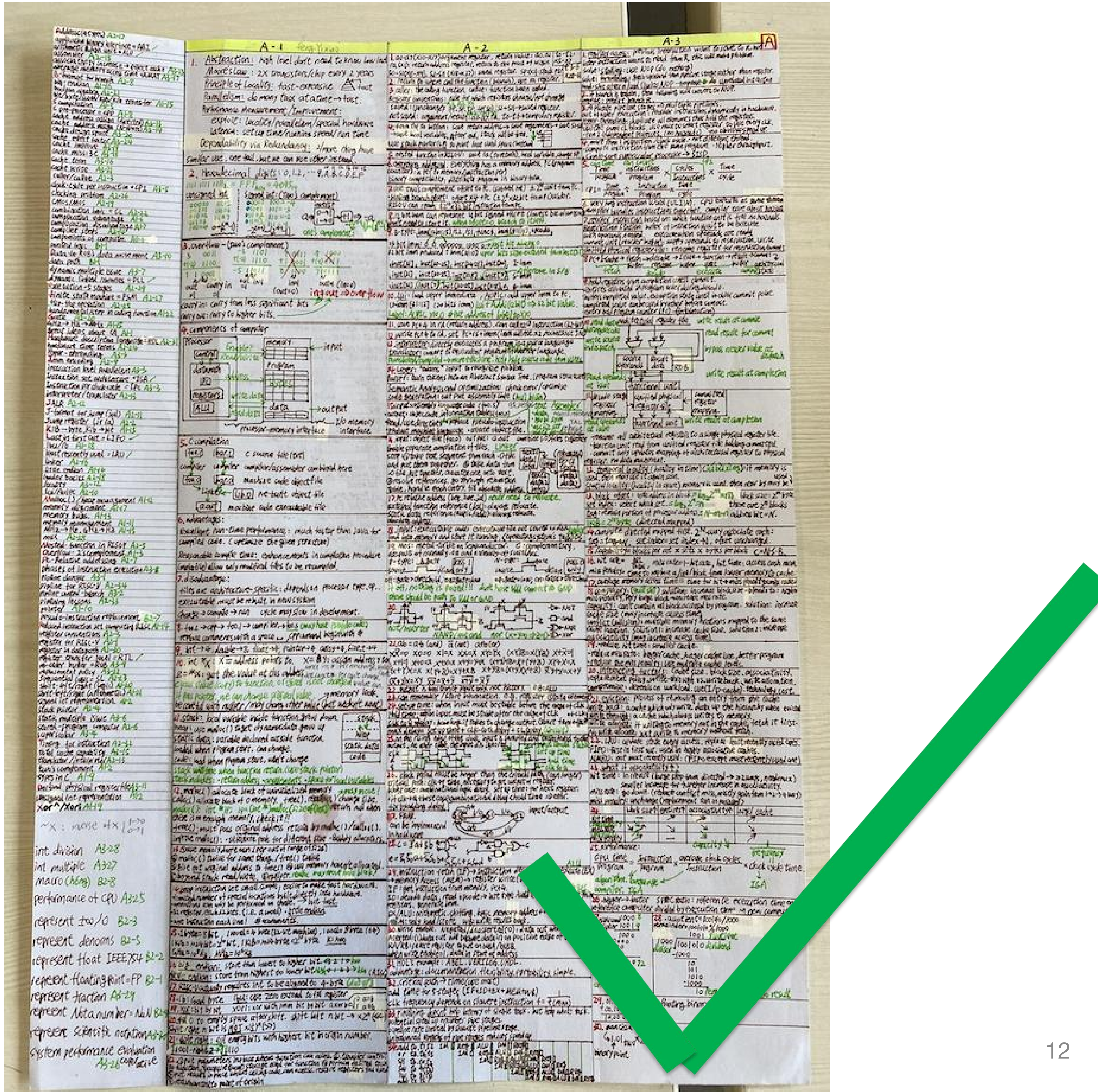

COMPUTER
INTERFACE
EDITION
MORGAN KAUFMANN
DAVID A. PATTERSON
JOHN L. HENNESSY

Cheat Sheet

- 1 A4 Cheat Sheet allowed (double sided)
  - Midterm II: 2 pages
  - Final: 3 pages
- Rules:
  - *Hand-written* – **not printed/photocopied!**
  - Your **name** in pinyin on the top!
  - Cheat Sheets not complying to this rule will be **confiscated**!


FUNCTIONS OF SEVERAL VARIABLES
LEVEL CURVES
FUNCTION OF n VARIABLES
ε-δ DEFINITION OF CONTINUITY
PARTIAL DERIVATIVES
SECOND PARTIAL DERIVATIVES
CLAIRAUT'S THEOREM
EQUATIONS OF TANGENT PLANES TO SURFACES
PARTIAL DIFF. EQ'S
TOTAL DIFFERENTIAL
INCREMENTS
THE CHAIN RULE
THEOREM
DEPENDENCY DIAGRAMS
IMPLICIT DIFFERENTIATION
THE GRADIENT VECTOR
DIRECTIONAL DERIVATIVES
TANGENT PLANE TO A LEVEL SURFACE
NORMAL LINE TO A LEVEL SURFACE
MAXIMUM AND MINIMUM VALUES
FINDING ABSOLUTE MAX. AND MINS.
MAXIMIZING AND MINIMIZING

XUVAT
LAW OF REFLECTION
Newton's laws
OPAQUE
TRANSPARENT
TRANSLUCENT
REFRACTION
m/s
x3.6
km/h
÷3.6
THE SUN IS AN
FUSION


Old School Machine Structures
Application
Software
Hardware
Processor
Memory
Datapath & Control
Digital Design
Circuit Design
transistors
When C program starts
Valid Pointer Arithmetic
Add an integer to a pointer
Six Fundamental Steps in Calling a Function
优先级：从上至下优先递减
New School Machine Structures
Parallel Requests
Parallel Threads
Parallel Data
Hardware descriptions
6 Great Ideas in Computer Architecture
1. Abstraction
2. Moore's Law
3. Principle of Locality
Memory Hierarchy
4. Parallelism
5. Performance Measurement and Improvement
6. Dependability via Redundancy
Two's-Complement Representation
Program's address space
stack
heap
Static data
Code
Components of a Computer
Processor
Control
Datapath
PC
Registers
Arithmetic & Logic Unit ALU
Memory
Program
Bytes
Data
Input
Output
Processor-Memory Interface
I/O-Memory Interface
Common Memory Problems
Memory leaks
Levels of Representation/Interpretation
High Level Language
Assembly Language
Machine Language
Hardware Architecture Description
Logic Circuit Description
R format
opcode rs rt rd shamt funct


A-1
A-2
A-3

SIFT REFERENCE GUIDE (V.1.1) – CREATING TIMELINES WITH THE SIFT WORKSTATION

SANS COMPUTER FORENSICS and INCIDENT RESPONSE

THE PURPOSE OF THIS REFERENCE [illegible]UIDE IS TO WALK THROUGH THE [illegible]CESS OF BOOTING THE SIFT [illegible]ORKSTATION, CREATING A TIMELINE ("SUPER" OR "MICRO") AND REVIEWING IT.

1. VISIT: http://computer-forensics11.sans.org/community/downloads

1

[illegible]load: SIFT Workstation VM Appliance

Do[illegible]FT Workstation Installation

2

2. BOOT SIFT VM

Login: *sansforensics*
Password: *forensics*

3

$ sudo su

3. ELEVATE PRIVS

4

4. CONNECT IMAGE TO SIFT

Plug hard drive to physical host and attach to SIFT VM

5

5. HARD DRIVE MOUNTING *(if you are using log2timeline-sift and Single DD you can skip to 7-A)*

EWF/[illegible]

SINGLE OR SPLIT IMAGE (2 options):

[illegible]nt_ewf.py image.E01 /mnt/ewf

[illegible]image.E01 /mnt/ewf/

Not Needed For 7-A

# mount -t ntfs -o ro,loop,show_sys_files,streams_interface=wind[illegible]
offset=#### /mnt/ewf/<image> /mnt/windows_mount/

MOUNT TO MOUNT POINT

6

DD

SINGLE IM[illegible]

mount -t ntfs -o [illegible]w_sys_files,streams_interface=windows,offset=#### image.dd /m[illegible]ws_mount/

SPLIT IMAGE (2 step pro[illegible]

affuse image.001 /mnt/aff
mount –t ntfs-3g –o loop,ro,show[illegible]/mnt/aff/<image> /mnt/windows_[illegible]

HOW TO CALCULATE THE OFFSET FOR MOUNTING

1. Run mmls to query partition layout
 # mmls image.E01
2. Identify partition and byte offset
3. (Partition byte offset) x (bytes per sector) = **offset #### to use!**
Example: 63 X 512 = 32256

Note: If needed, repeat for each partition. Make new mount point:
 # mkdir /mnt/windows_mount2/

6. log2timeline default timezone is set to examiner local host. To change use -z [TIMEZONE] option. To list all available timezones: # log2timeline -z list

7-A & 7-B

log2timeline PARSING PLUGINS

apache2_error - Apache2 error log file
chrome - Chrome history file
encase_dirlisting - CSV file that is exported from encase
evt - Windows 2k/XP/2k3 Event Log
evtx - Windows Event Log File (EVTX)
exif - Metadata information from files using ExifTool
ff_bookmark - Firefox bookmark file
firefox2 - Firefox 2 browser history
firefox3 - Firefox 3 history file
ftk_dirlisting - CSV file that is exported from FTK Imager (dirlisting)
generic_linux - Generic Linux logs that start with MMM DD HH:MM:SS
iehistory - index.dat file containg IE history
iis - IIS W3C log file
isatxt - ISA text export log file
jp_ntfs_change - CSV output file from JP (NTFS Change log)
mactime - Body file in the mactime format
mcafee - Log file
mft - NTFS MFT file
mssql_errlog - ERRORLOG file produced by MS SQL server
ntuser - NTUSER.DAT registry file
opera - Opera's global history file
oxml - OpenXML document pcap
pcap - PCAP file
pdf - Available PDF document metadata
prefetch - Prefetch directory
recycler - Recycle bin directory
restore 0.9 - Restore point directory
safari - Safari History.plist file
sam - SAM registry file
security - SECURITY registry file
setupapi - SetupAPI log file in Windows XP
skype_sql - Skype database
software - SOFTWARE registry file
sol - .sol (LSO) or a Flash cookie file
squid - Squid access log (http_emulate off)
syslog - Linux Syslog log file
system - SYSTEM registry file
tln - Body file in the TLN format
volatility - Volatility output files (psscan2, sockscan2, ...)
win_link - Windows shortcut file (or a link file)
wmiprov - wmiprov log file
xpfirewall - XP Firewall log

List plugins # log2timeline -f list
...HELP EXPAND THIS LIST. BUILD PLUGINS!!!

8

7-A: AUTOMATED SUPER TIMELINE CREATION

log2timeline-sift -o –z [TIMEZONE] -p [PARTITION #] -i [IMAGE FILE]

DISK IMAGE (prompt for partition, mount, and run):

XP # log2timeline-sift –z EST5EDT -i image
WIN7 # log2timeline-sift -win7 -z EST5EDT -i image

FOR PARTITION (mount and run using all applicable pl[illegible]

XP # log2timeline-sift –z EST5EDT -p 0 -i partiti[illegible]
WIN7 # log2timeline-sift -win7 -z EST5EDT [illegible]tition

OTHER USAGE EXAMPLES:

Display list of available pl[illegible]
log2timeline -f list
Run log2timeline use[illegible]use only specific plugins:
log2timeline-sif[illegible]preftch –z EST5EDT -i image.dd
Help (man pag[illegible]
log2timeli[illegible]

[illegible]NUAL "MICRO" TIMELINE CREATION

[illegible]TIONS] [-f FORMAT] [-z TIMEZONE] [-o OUTPUT MODULE] [-w [illegible]G_FILE/LOG_DIR [–] [FORMAT FILE OPTIONS]

[illegible]EM METADATA (using log2timeline or fls)

[illegible]stem data w/log2timeline from mounted file system:
log[illegible]f mft -o mactime –r -z EST5EDT -w mft.bo[illegible]lume/
OR Extract M[illegible]ge using Sleuthkit:
fls -m "" -o offs[illegible].dd > fls.body
Convert body file for[illegible]format w/ mactime:
mactime –b fls.body –d[illegible]

ARTIFACTS (run l2l on mounted file s[illegible] plugins recursively)

Extract artifacts w/ log2timeline and run[illegible]ed file system:
log2timeline -f firefox3,chrome -o mactim[illegible]EDT -w web.body /mnt/volume/
Convert body file format to CSV format w/ mactime:
mactime –b log2timeline.body –d > log2timeline.csv

HELP? OPTIONS? USAGE?
log2timeline -help
Log2timeline-sift -help
L2t_process -help

OTHER log2timeline OUTPUT FORMATS
Note: CSV is Default Output
-BeeDocs - Mac OS X visualization tool
-CEF - Common Event Format - ArcSight
-CFTL - XML file- CyberForensics TimeLab visualization tool
-CSV - comma separated value file
-Mactime - Both older and newer version of the format supported for use by TSK's mactime
-SIMILE - XML file - SIMILE timeline visualization widget
-SQLite - SQLite database
-TLN - Tab Delimited File
-TLN - Format used by some of H Carvey tools, expressed as a ASCII output
-TLNX - Format used by some of H Carvey tools, expressed as a XML document

10

9

8. CSV [illegible]PUT (/cases/timeline-output-folder)

-dat[illegible]he event, in the format of MM/DD/YYYY
[illegible]of day, expressed in a 24h format, HH:MM:SS
[illegible]: the timezone that was used to call the tool with.
[illegible]: MACB meaning of the fields, comp w/ mactime format.
[illegible]ce: Source short name (i.e. registry entries are REG)
-sourcetype: Desc of the source ("Internet Explorer" instead of WEBHIST)
-type: Timestamp type (i.e. "Last Accessed", "Last Written")
-user: Username associated with the entry, if one is available.
-host: Hostname associated with the entry, if one is available.
-short: Contains less text than the full description field.
-desc: where majority info is stored, the actual parsed desc of the entry.
-version: Version number of the timestamp object.
-filename: Filename with the full path that contained the entry
-inode: inode number of the file being parsed.
-notes: Some input modules insert additional information in the form of a note, which comes here. Or it can be used during the review.
-format: Input module name used to parse the file.
-extra: Additional information parsed is joined together and put here.

9. FILTER TIMELINE

Filter timeline with date range to include only:
l2t_process -b timeline.csv MM-DD-YYYY..MM-DD-YYYY > filtered.csv
Filter timeline with keyword list (one term per line in keywords.txt):
l2t_process -b timeline.csv -k keywords.txt > filtered.csv
What sources are in your timeline?
awk–F , '{print $6;}' timeline.csv| grep–v sourcetype|sort | uniq
Find all LNK files that reference E Drive
grep"Shortcut LNK" timeline.csv| grep"E:"
FiindMountPoints2 entries that reference E Drive
grep"MountPoints2 key" timeline.csv} | grep"E drive"
grepUSB timeline.csv| grep"SetupAPILog"

File System	M	A	C	B
Ext2/3	Modified	Accessed	Changed	N/A
FAT	Written	Accessed	N/A	Created
NTFS	File Modified	Accessed	MFT Modified	Created
UFS	Modified	Accessed	Changed	N/A

10. CONNECT TO SIFT

✓ [illegible]> SETTINGS -> OPTIONS -> Shared Fo[illegible]rs -> Always Enabled (Check)
✓ 2. SIFT Desktop > VMware-Shared-Drive
✓ Access from a Win Machine \\SIFTWORKSTATION

11

11. REVIEW TIMELINE

Review timelines using:
- Open, Soft, Filter with Excel
- Import into SPLUNK
- SIMILE
- Tapestry

BY DAVID NIDES (12/16/2011)
TWITTER: @DAVNADS
BLOG: DAVNADS.BLOGSPOT.COM
EMAIL: DNIDES@KPMG.COM
CREDITS TO: ED GOINGS, ROB LE[illegible]
KRISTINN GUDJONSSON, KPMG &[illegible]
QUESTIONS/FEEDBACK–CONTACT U[illegible]

KEY
Red text – image/source
Blue text – mount point
Purple text – output file
Green text – log2timeline plugins
Brown text - TimeZone

Outline

- Useful building blocks
  - ALU design
  - Register file
  - Memory considerations
- Datapath
- Design of the controller

Warm-up

- A classic problem: sequence detection for “010” (non-overlapping)

```
Input:  0 1 0 0 1 0 1 0 1 1 0
Output: 0 0 0 1 0 0 1 0 0 0 0
```

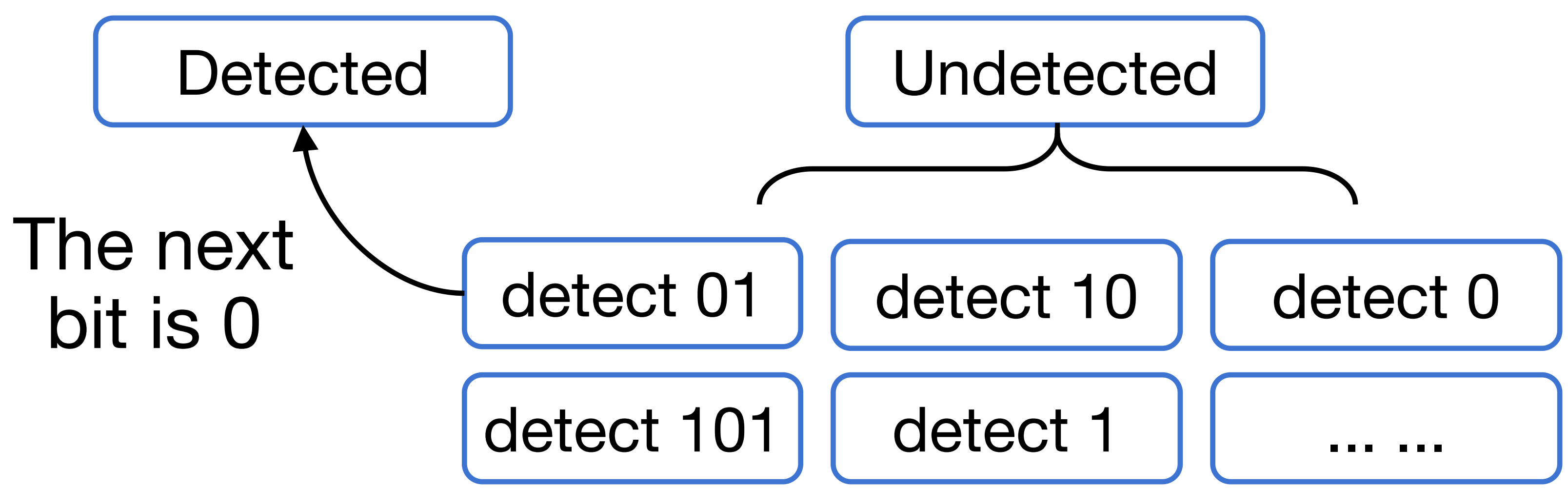
- Step 1: Draw finite state machine of the desired function (we ignore the initialization)
- Step 2: Define/assign binary numbers to represent the states, the inputs and the outputs
- Step 3: Write down the truth table (enumerate input/previous state (and current state) and their corresponding current state (and output))
- Step 4: Use template and decide the combinational block for state transition and output logic

Warm-up

- A classic problem: sequence detection for “010” (non-overlapping)

```
Input:  0 1 0 0 1 0 1 0 1 1 0
Output: 0 0 0 1 0 0 1 0 0 0 0
```

- Step 1: Draw finite state machine of the desired function

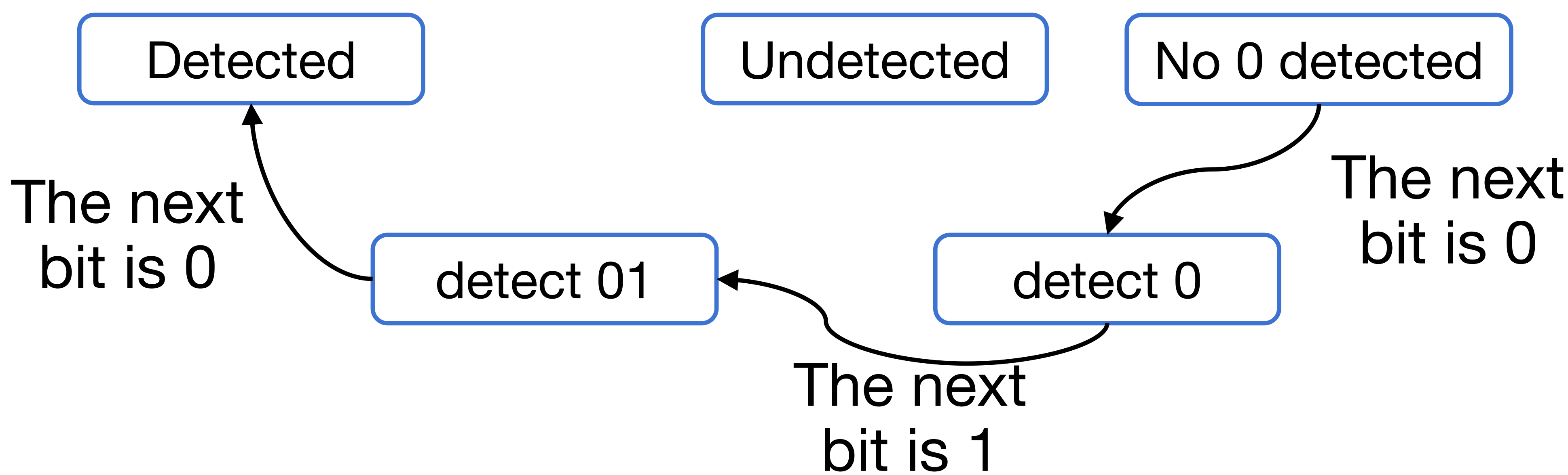


Warm-up

- A classic problem: sequence detection for “010” (non-overlapping)

```
Input:  0 1 0 0 1 0 1 0 1 1 0
Output: 0 0 0 1 0 0 1 0 0 0 0
```

- Step 1: Draw finite state machine of the desired function

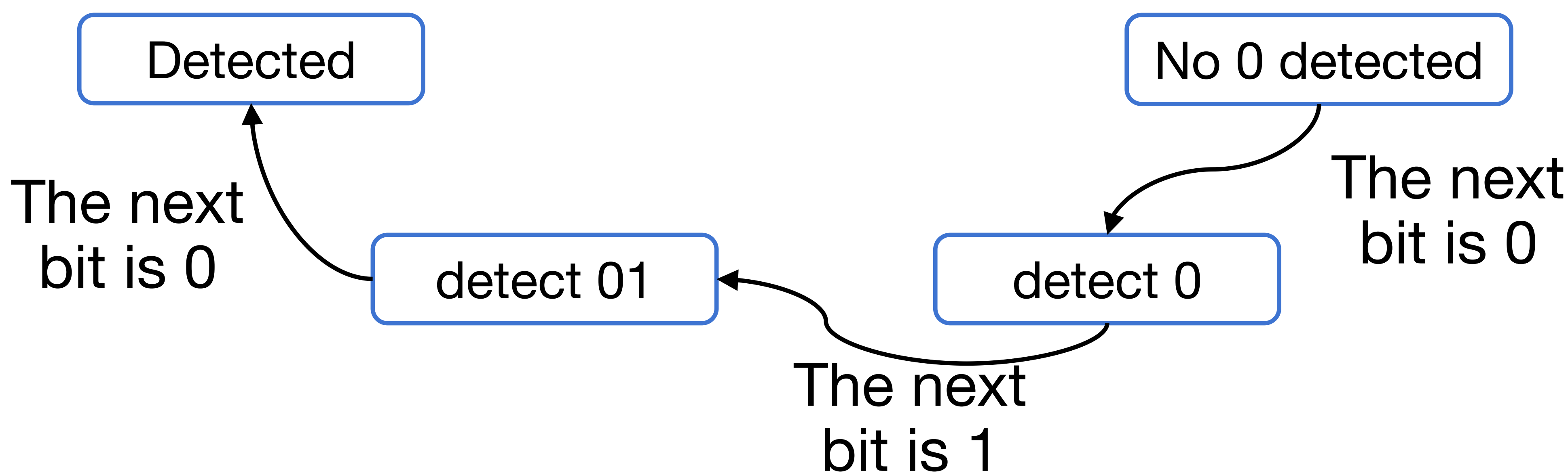


Warm-up

- A classic problem: sequence detection for “010” (non-overlapping)

```
Input:  0 1 0 0 1 0 1 0 1 1 0
Output: 0 0 0 1 0 0 1 0 0 0 0
```

- Step 1: Draw finite state machine of the desired function

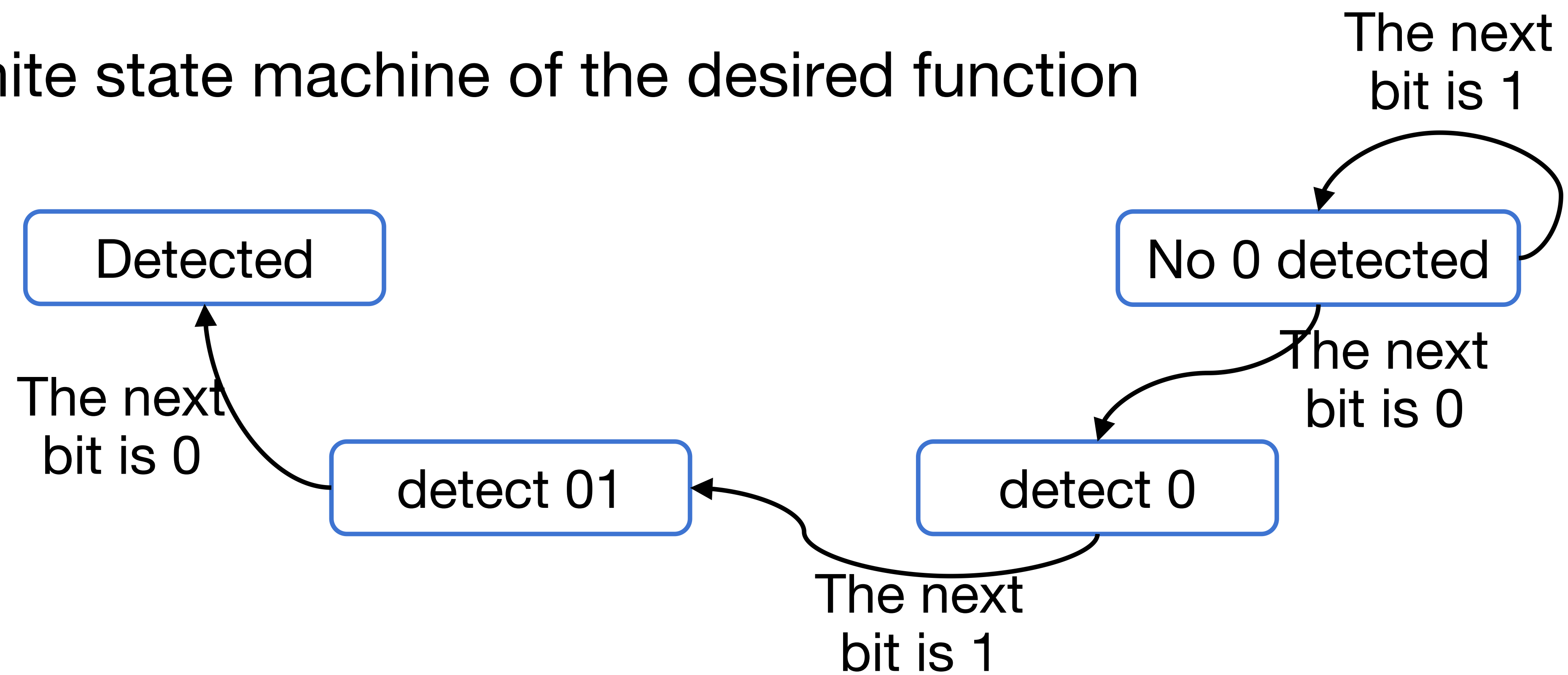


Warm-up

- A classic problem: sequence detection for “010” (non-overlapping)

```
Input:  0 1 0 0 1 0 1 0 1 1 0
Output: 0 0 0 1 0 0 1 0 0 0 0
```

- Step 1: Draw finite state machine of the desired function

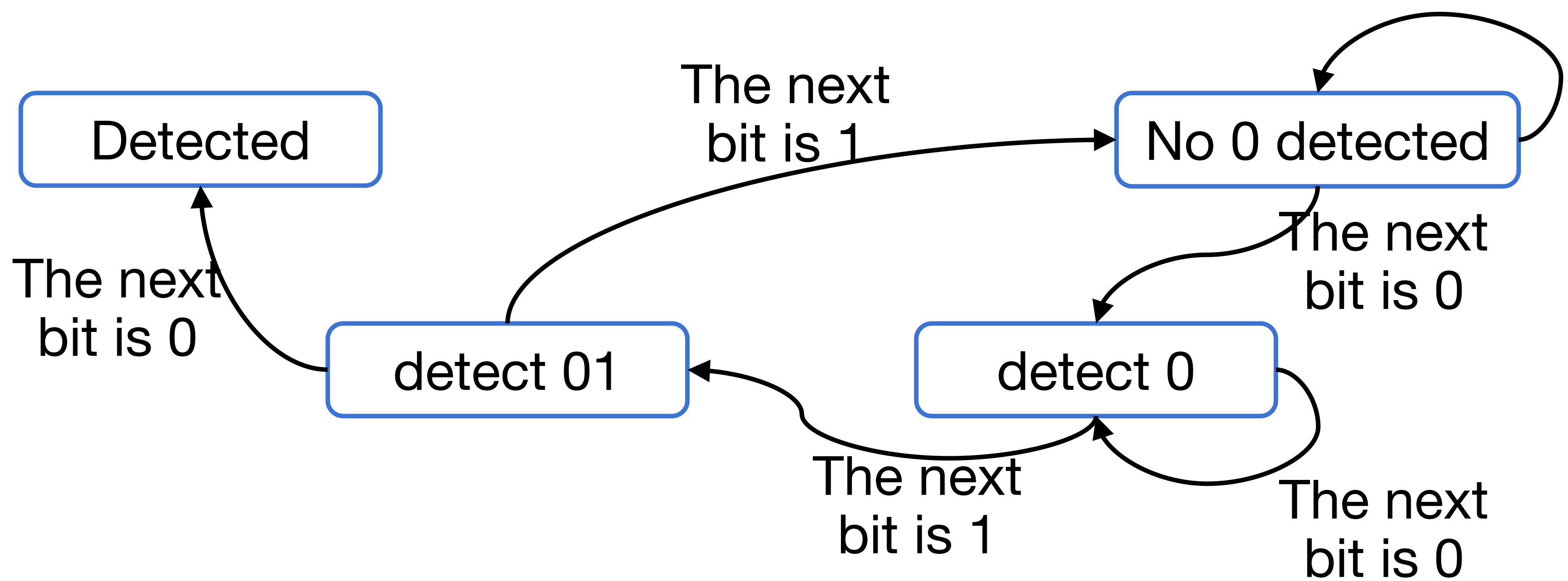


Warm-up

- A classic problem: sequence detection for “010” (non-overlapping)

```
Input:  0 1 0 0 1 0 1 0 1 1 0
Output: 0 0 0 1 0 0 1 0 0 0 0
```

- Step 1: Draw finite state machine of the desired function

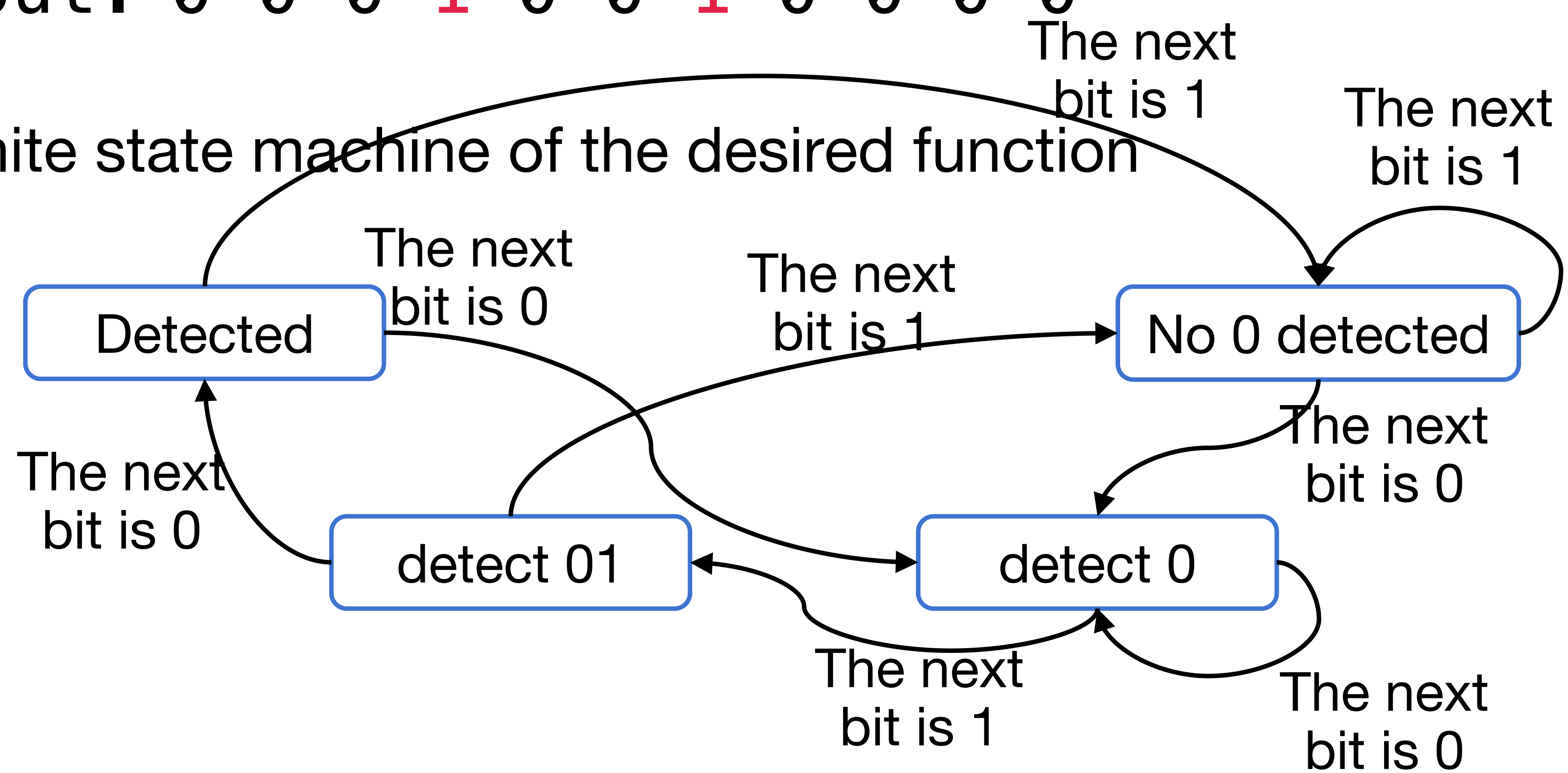


Warm-up

- A classic problem: sequence detection for “010” (non-overlapping)

```
Input:  0 1 0 0 1 0 1 0 1 1 0
Output: 0 0 0 1 0 0 1 0 0 0 0
```

- Step 1: Draw finite state machine of the desired function

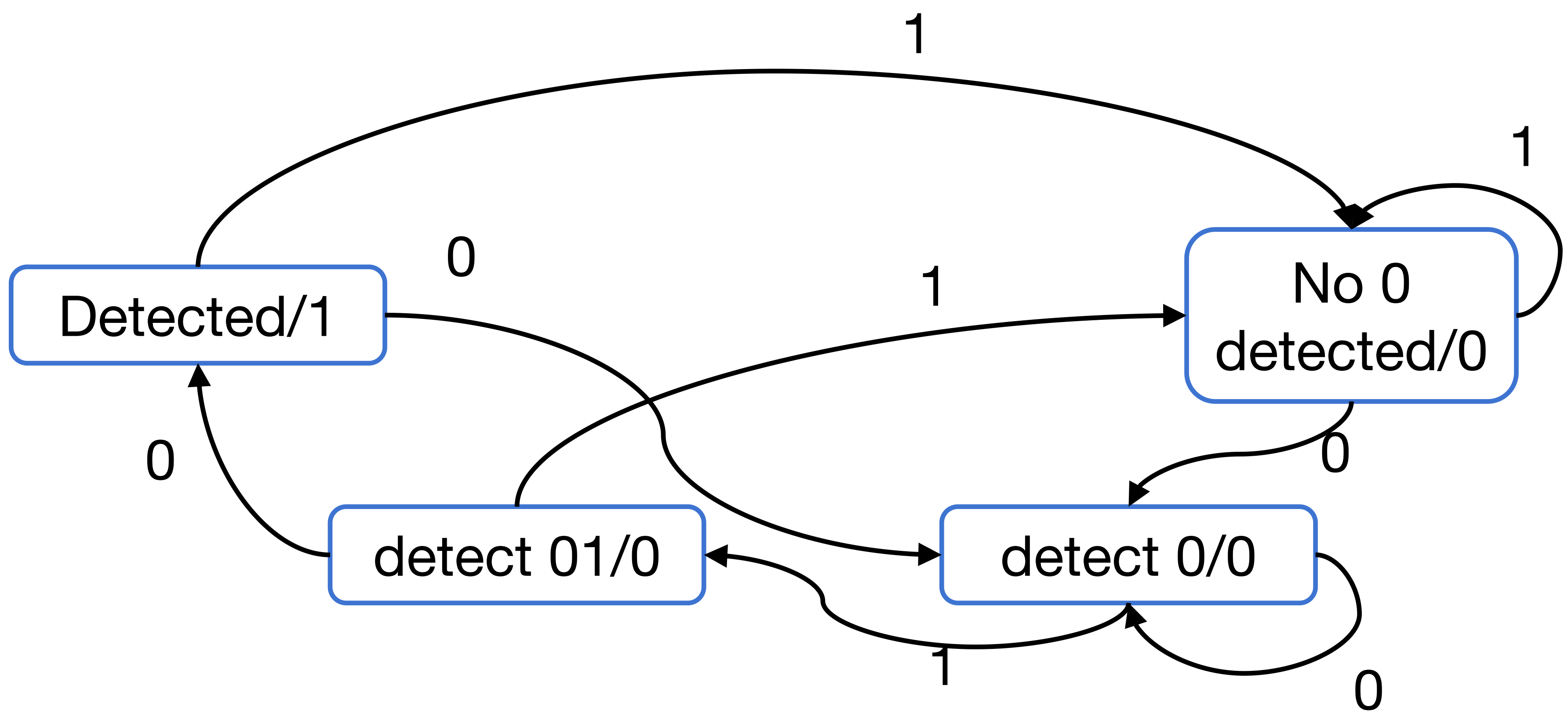


Warm-up

- A classic problem: sequence detection for “010” (non-overlapping)

```
Input:  0 1 0 0 1 0 1 0 1 1 0
Output: 0 0 0 1 0 0 1 0 0 0 0
```

- Step 1: Draw finite state machine of the desired function

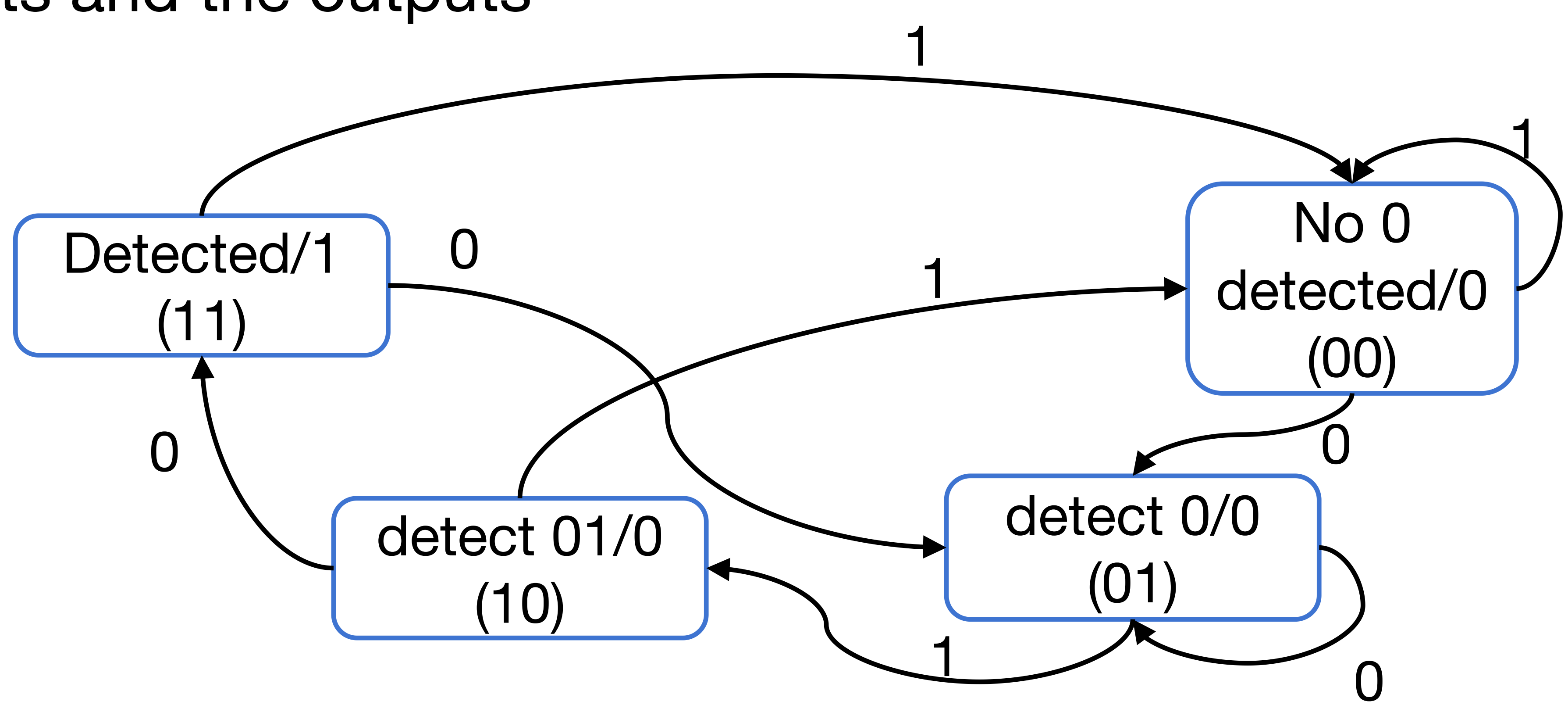


Warm-up

- A classic problem: sequence detection for "010" (non-overlapping)

```
Input:  0 1 0 0 1 0 1 0 1 1 0
Output: 0 0 0 1 0 0 1 0 0 0 0
```

- Step 2: Define/assign binary numbers to represent the states, the inputs and the outputs

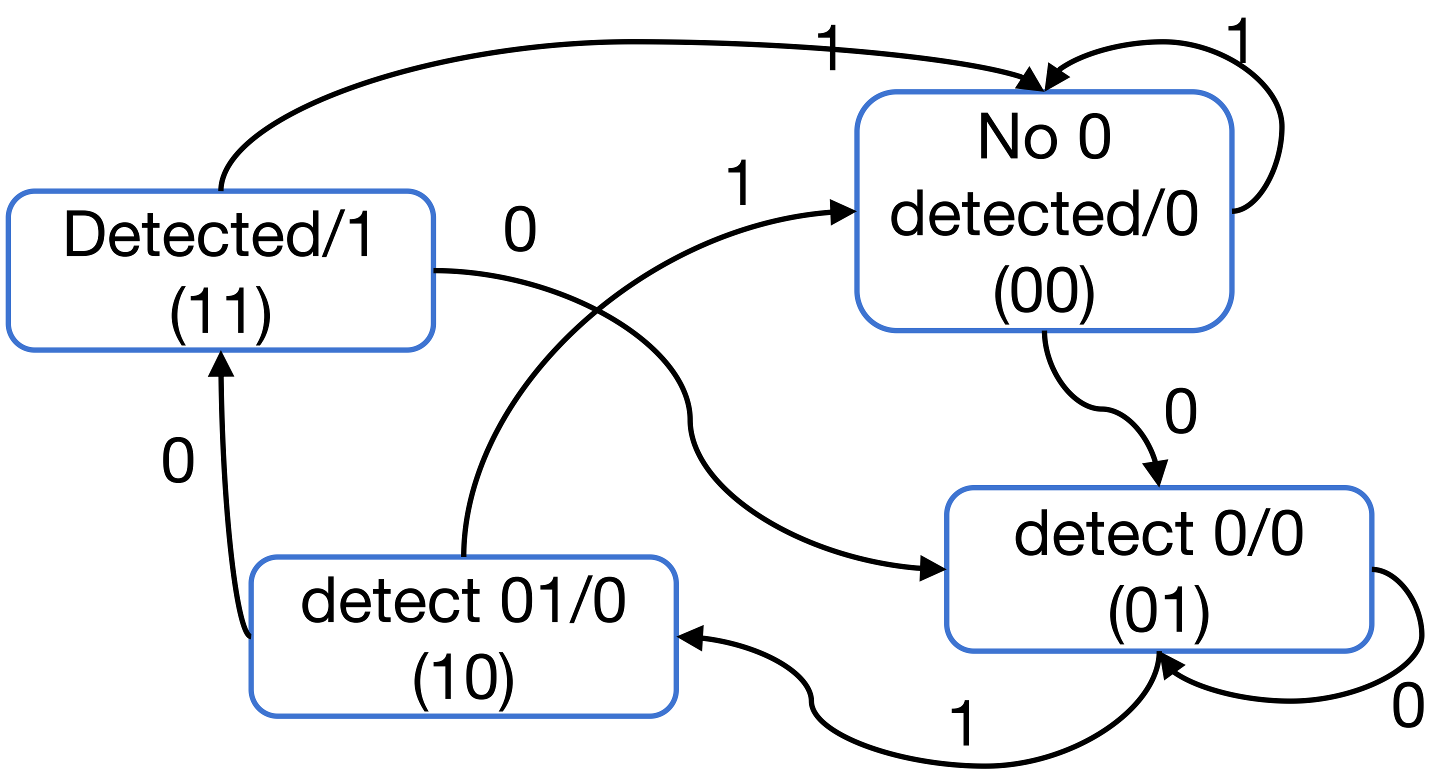


Warm-up

- A classic problem: sequence detection for “010” (non-overlapping)

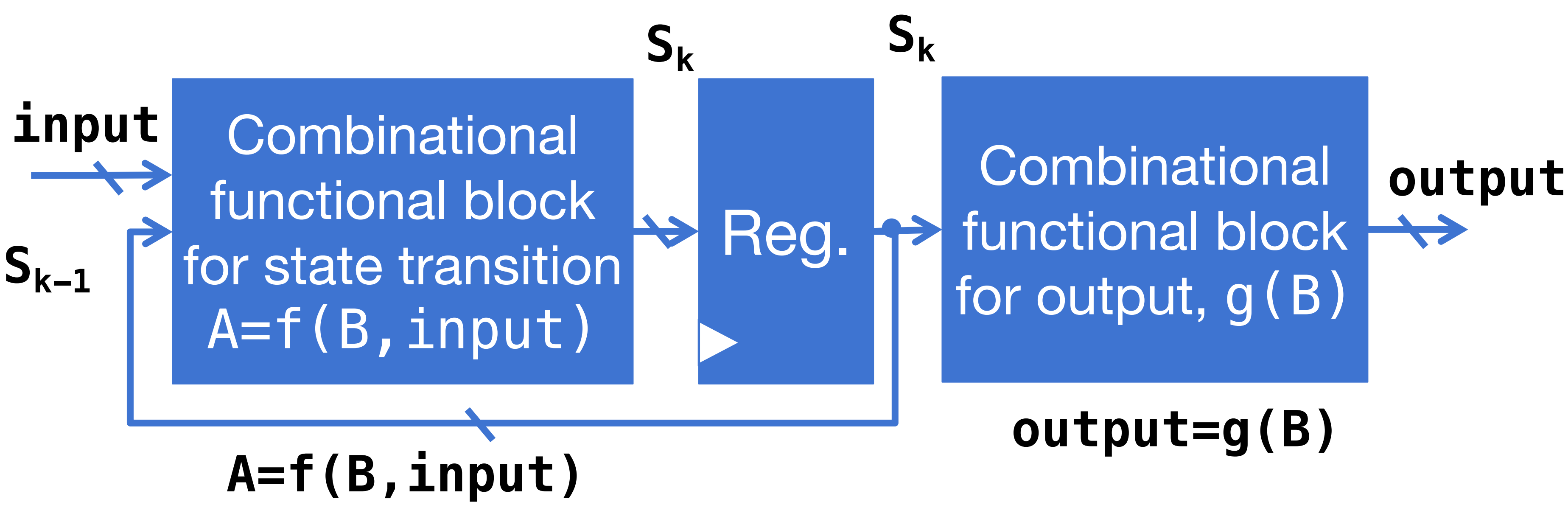
```
Input:  0 1 0 0 1 0 1 0 1 1 0
Output: 0 0 0 1 0 0 1 0 0 0 0
```

- Step 3: Write down the truth table (enumerate input/previous state (and current state) and their corresponding current state (and output))



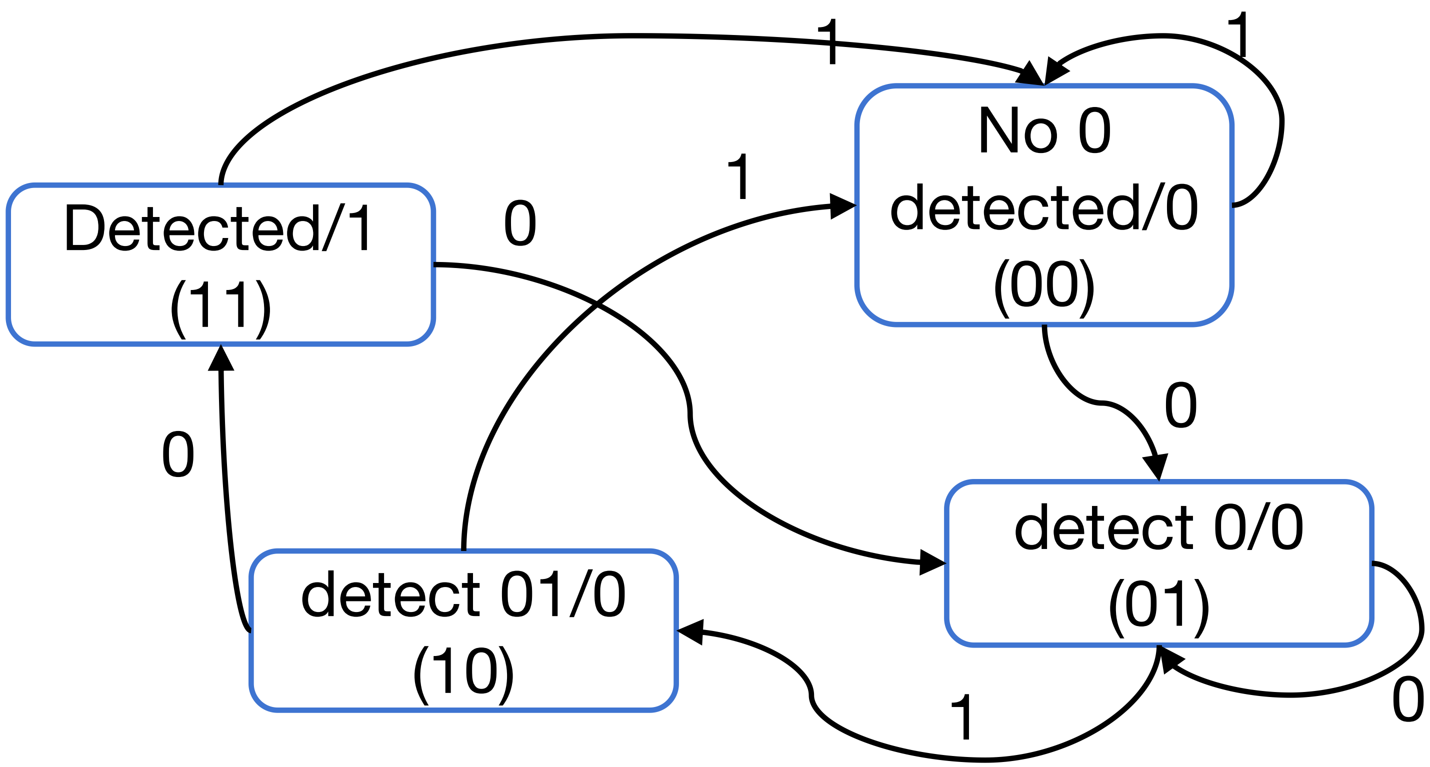
input	Previous state S[1] k-1	Previous state S[0] k-1	Current state S[1] k	Current state S[0] k	output
0	0	0	0	1	0
0	0	1	0	1	0
0	1	0	1	1	1
0	1	1	0	1	0
1	0	0	0	0	0
1	0	1	1	0	0
1	1	0	0	0	0
1	1	1	0	0	0

Warm-up



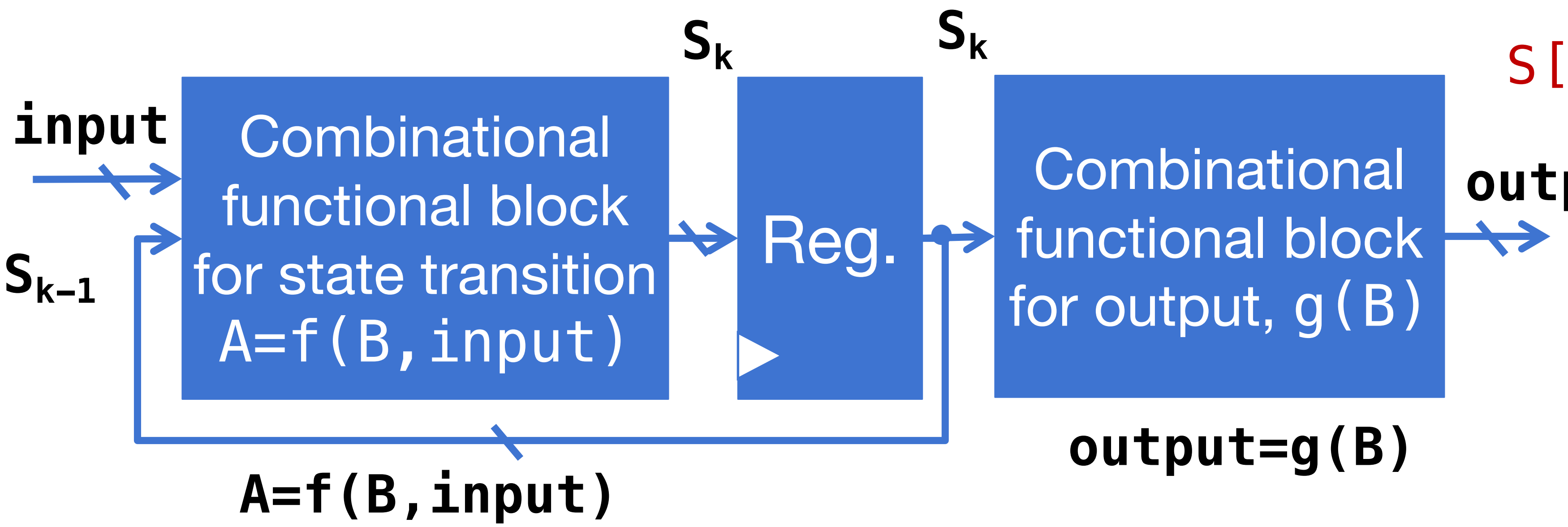
$output=S[1]_kS[0]_k$

- Step 4: Use template and decide the combinational block for state transition and output logic



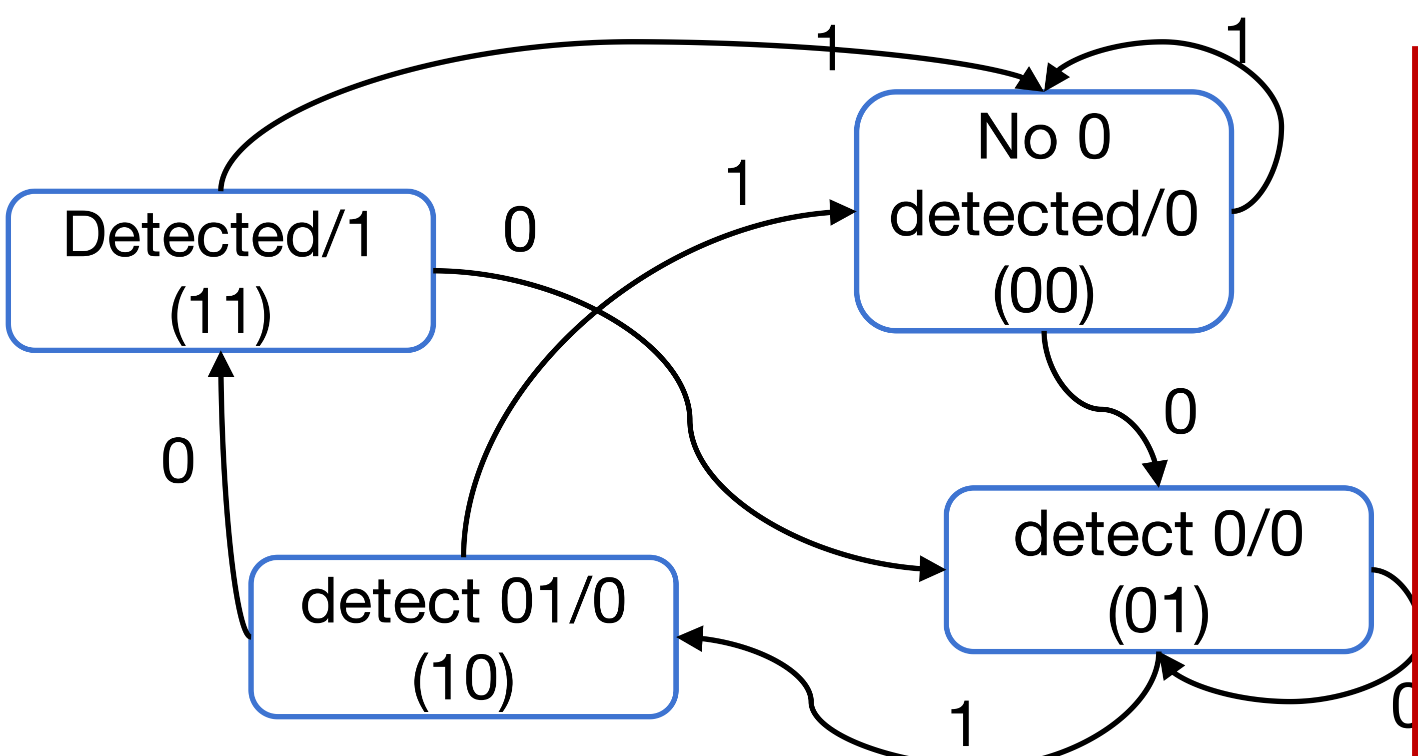
	Previous state		Current state		
input	$S[1]_{k-1}$	$S[0]_{k-1}$	$S[1]_k$	$S[0]_k$	output
0	0	0	0	1	0
0	0	1	0	1	0
0	1	0	1	1	1
0	1	1	0	1	0
1	0	0	0	0	0
1	0	1	1	0	0
1	1	0	0	0	0
1	1	1	0	0	0

Warm-up



$S[1]_k = \overline{S[1]_{k-1}}\, S[0]_{k-1}\, input + S[1]_{k-1}\, \overline{S[0]_{k-1}}\, \overline{input}$

- Step 4: Use template and decide the combinational block for state transition and output logic



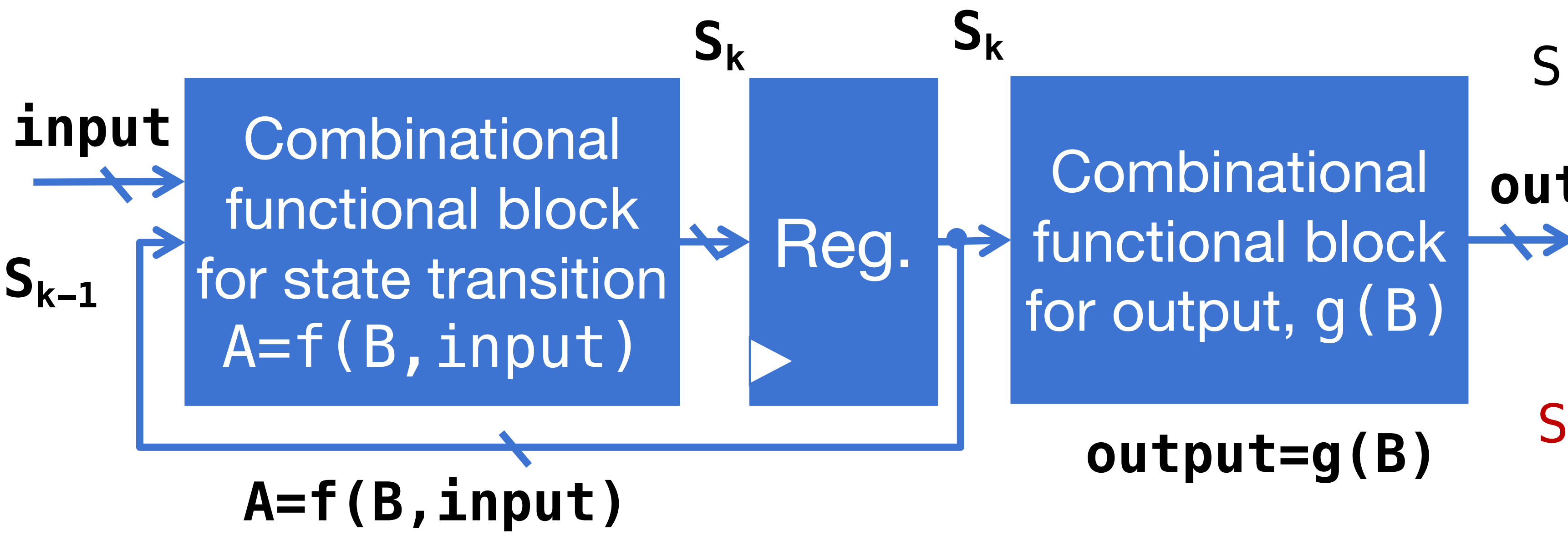
	Previous state		Current state		
input	S[1] k-1	S[0] k-1	S[1] k	S[0] k	output
0	0	0	0	1	0
0	0	1	0	1	0
0	1	0	1	1	1
0	1	1	0	1	0
1	0	0	0	0	0
1	0	1	1	0	0
1	1	0	0	0	0
1	1	1	0	0	0

Warm-up

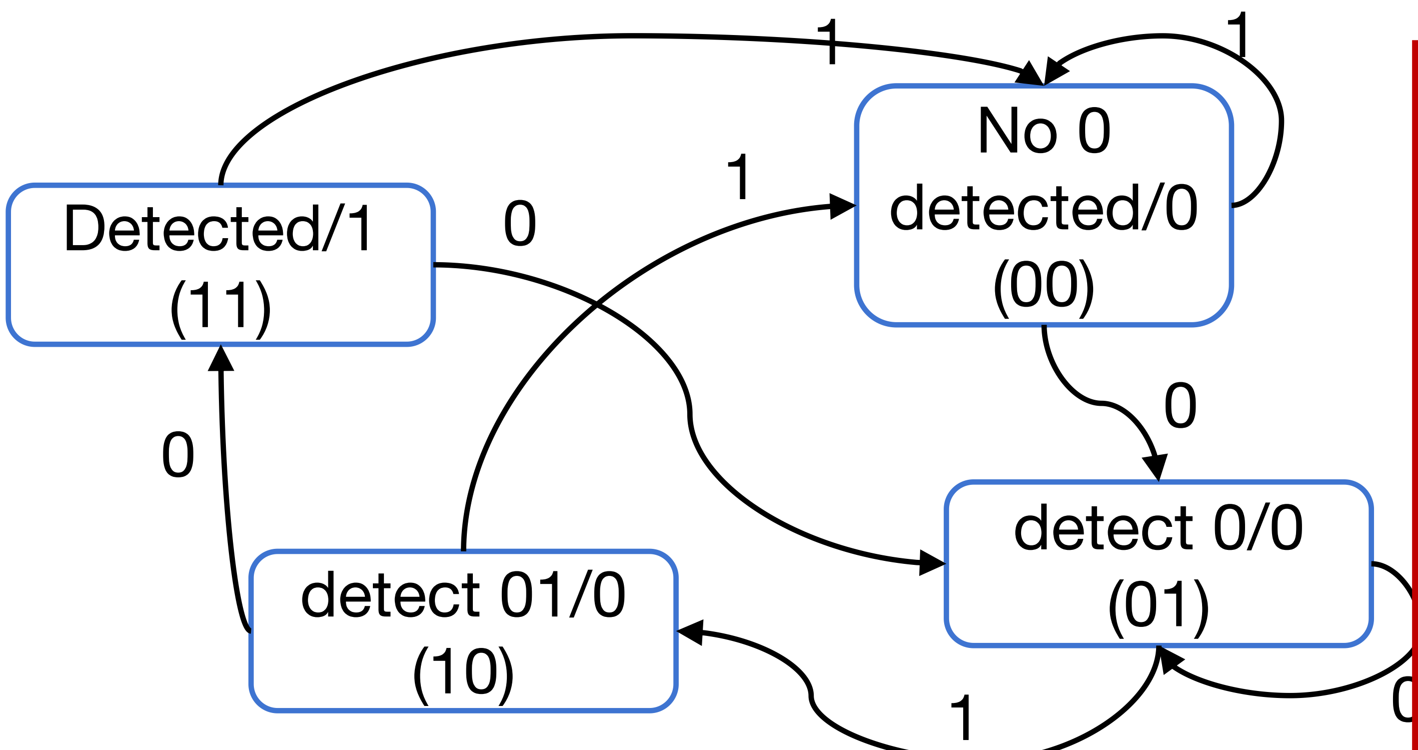
$output = S[1]_k S[0]_k$

$S[1]_k = \overline{S[1]_{k-1}} S[0]_{k-1} input + S[1]_{k-1} \overline{S[0]_{k-1}} \overline{input}$

$S[0]_k = \overline{input}$

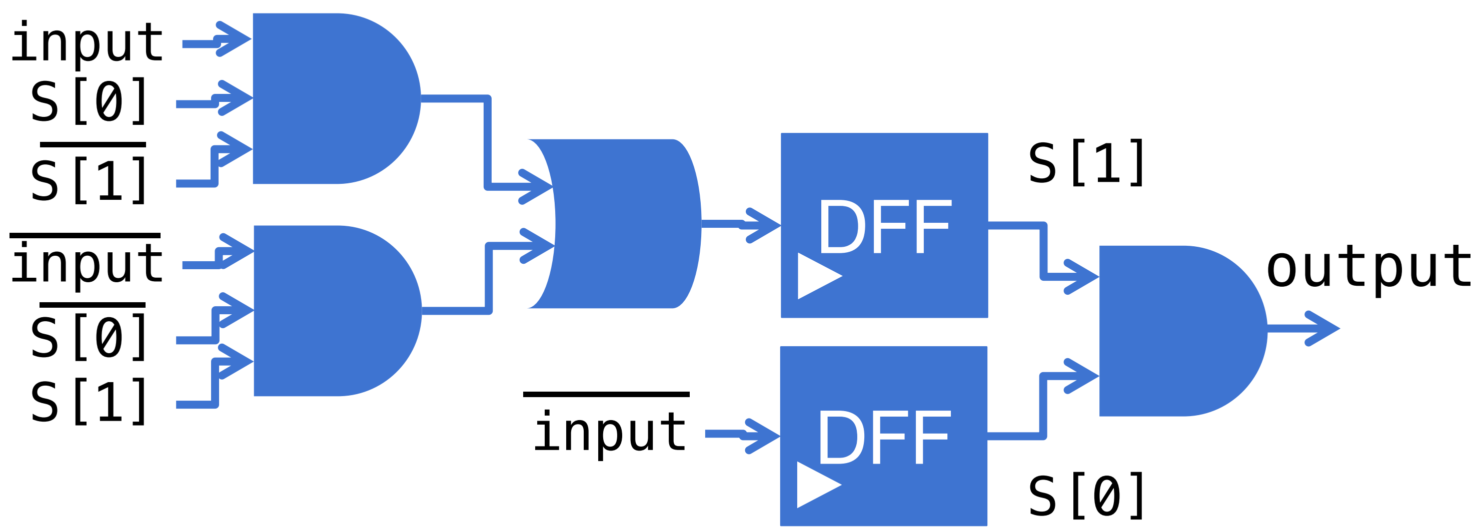


- Step 4: Use template and decide the combinational block for state transition and output logic



input	Previous state S[1] k−1	Previous state S[0] k−1	Current state S[1] k	Current state S[0] k	output
0	0	0	0	1	0
0	0	1	0	1	0
0	1	0	1	1	1
0	1	1	0	1	0
1	0	0	0	0	0
1	0	1	1	0	0
1	1	0	0	0	0
1	1	1	0	0	0

Warm-up

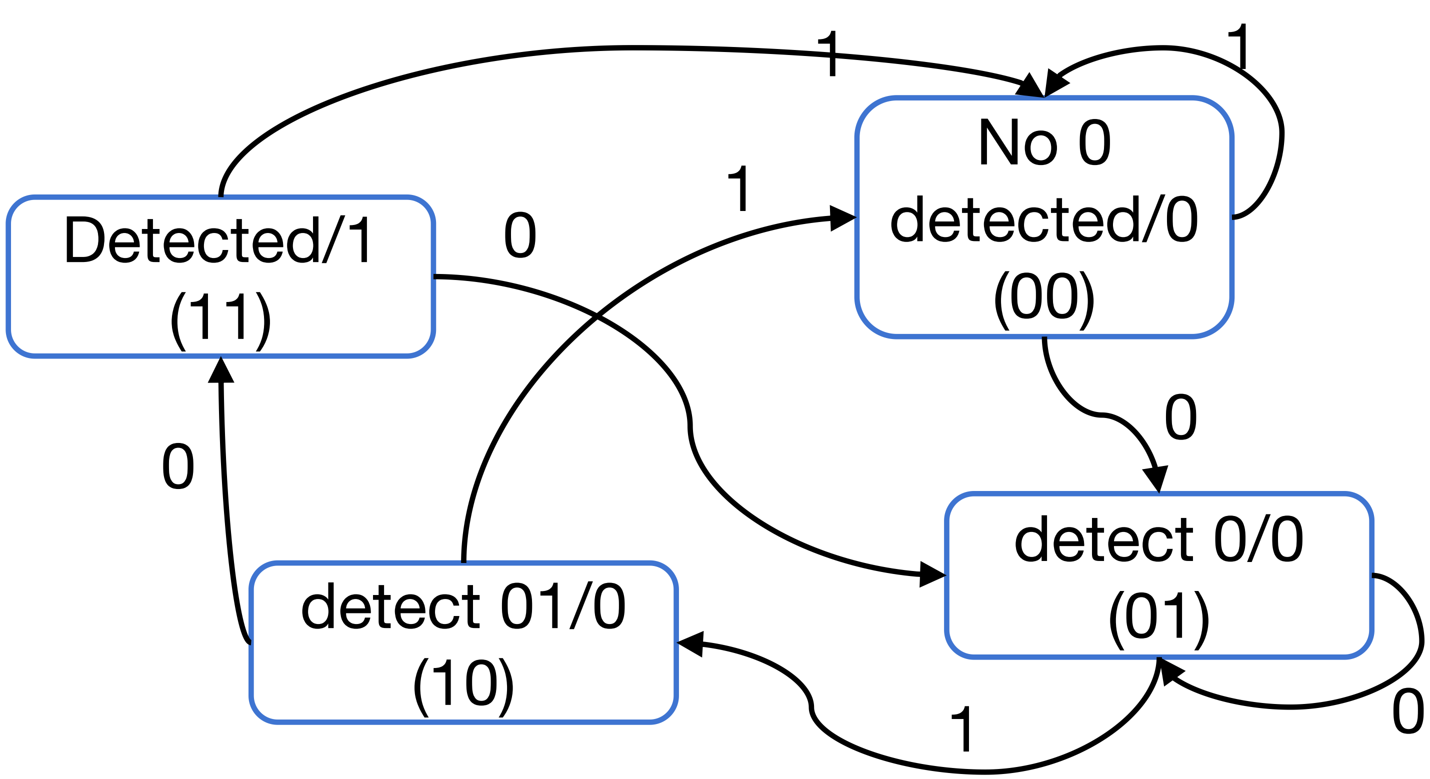


$$output=S[1]_k S[0]_k$$

$$S[1]_k=\overline{S[1]}_{k-1}S[0]_{k-1}input+S[1]_{k-1}\overline{S[0]}_{k-1}\overline{input}$$

$$S[0]_k=\overline{input}$$

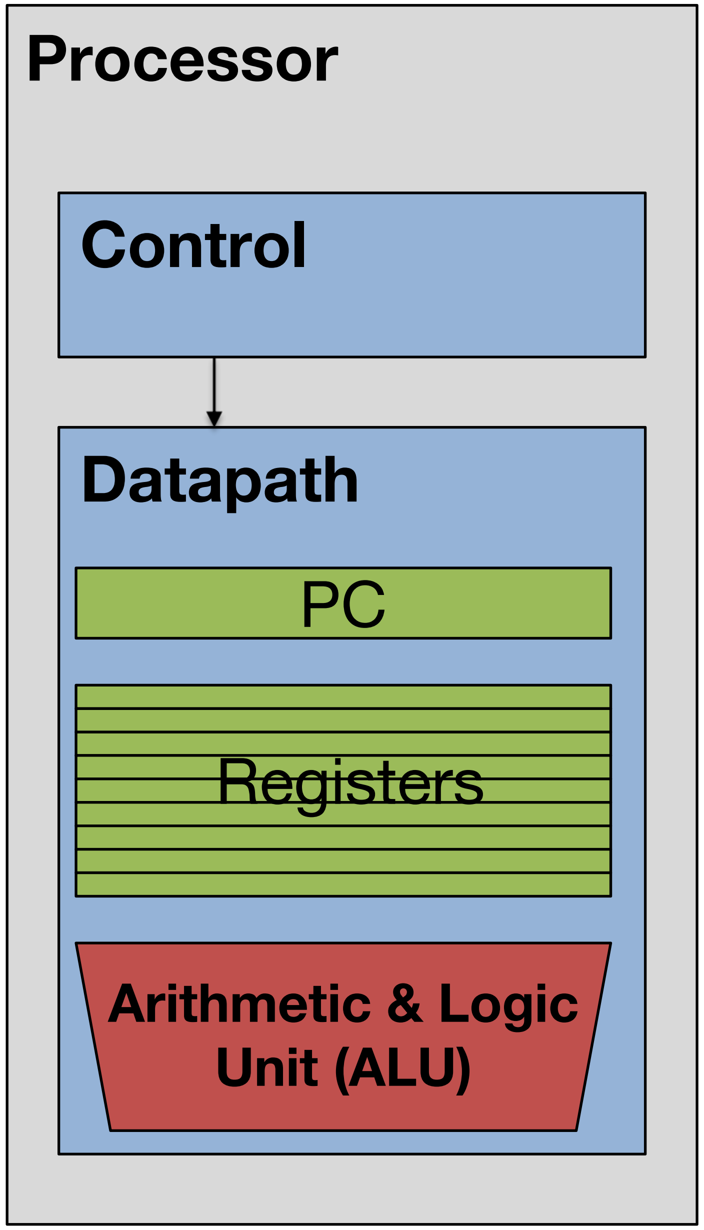
- Step 4: Use template and decide the combinational block for state transition and output logic



input		Previous state		Current state	output
	$S[1]_{k-1}$	$S[0]_{k-1}$	$S[1]_k$	$S[0]_k$	
0	0	0	0	1	0
0	0	1	0	1	0
0	1	0	1	1	1
0	1	1	0	1	0
1	0	0	0	0	0
1	0	1	1	0	0
1	1	0	0	0	0
1	1	1	0	0	0

Controller & Datapath

- A CPU that support RV32I can have so many states

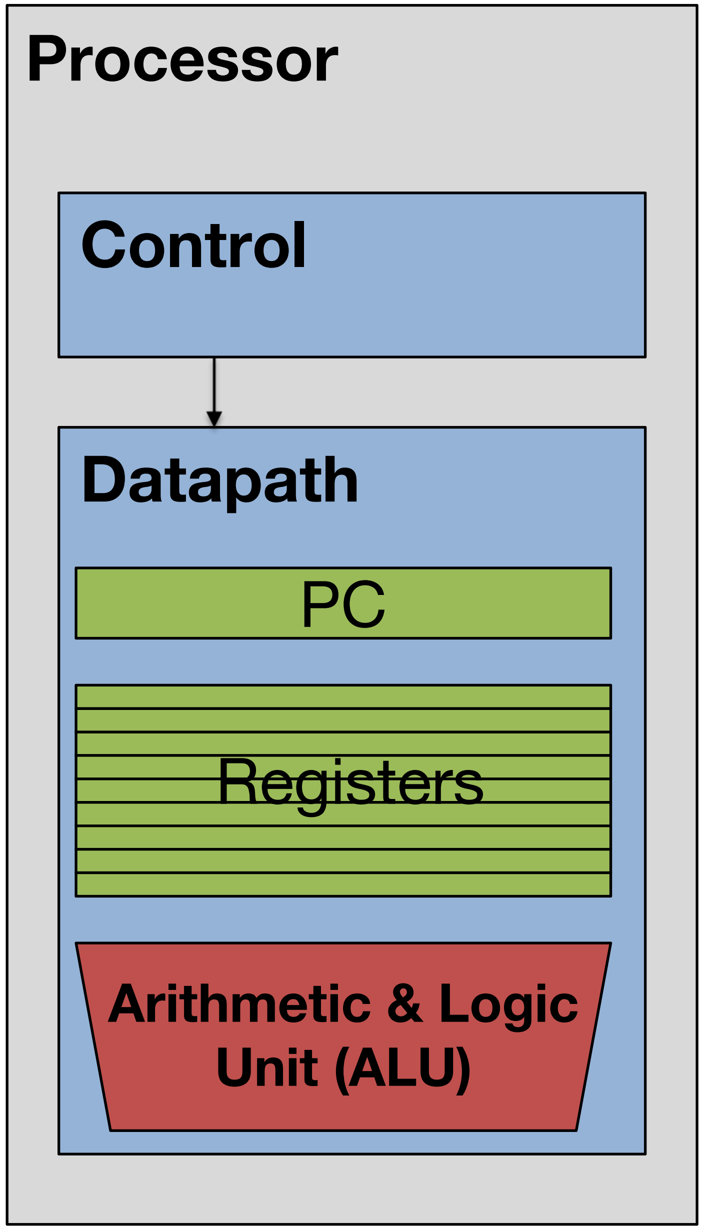


- Consider the 32 registers alone
  - `x0` always 0
  - Each bit in the other registers can be 0 or 1
- Not practical to enumerate all the state transitions
- Top-down design: build small modules and then connect them as needed
- Most digital systems can be divided into datapth and controller
  - Datapath contains data processing and storage
  - Controller controls data access (still can be modeled as FSM)
- Recall the execution of an instruction

- Our Goal: Implement a RISC-V processor as a synchronous digital system (SDS).
- Each RV32I instruction can be done within 1 clock cycle (single-cycle CPU).

Controller & Datapath

- A CPU that support RV32I can have so many states

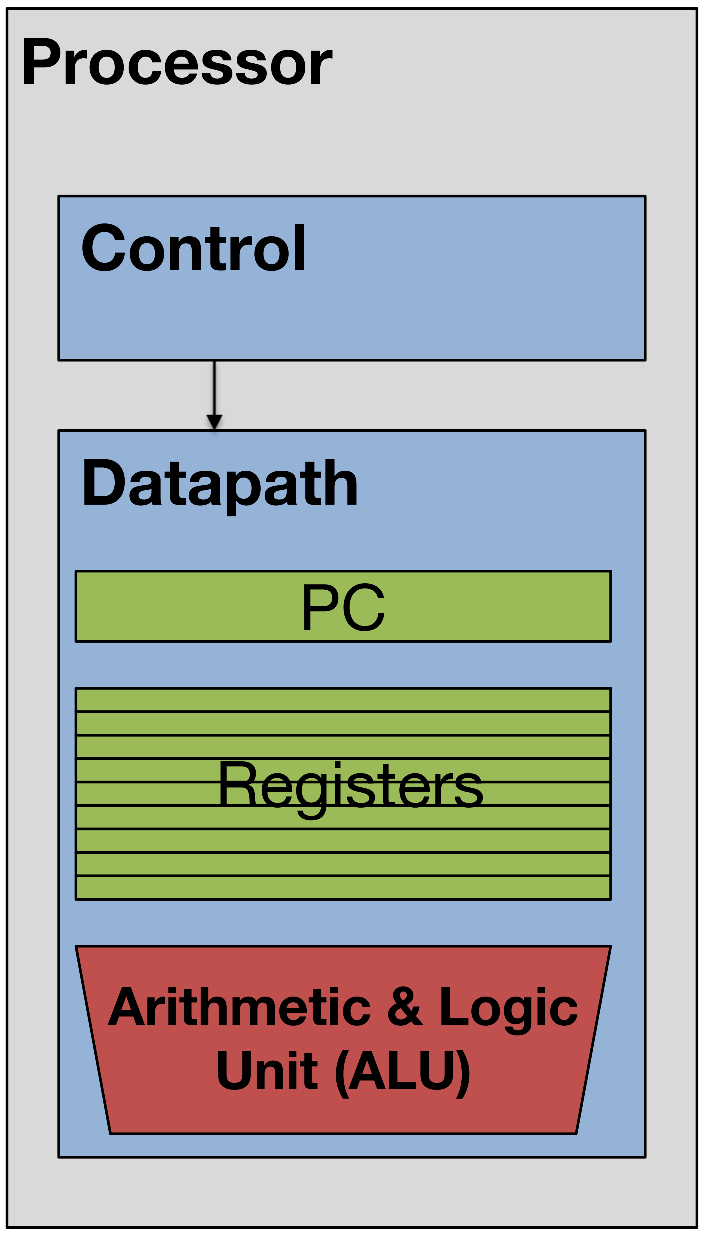


- Datapath
  - Start with basic building blocks
  - Add building blocks to the digital system with added supported instructions
- Controller
  - Can be considered as an FSM

- Our Goal: Implement a RISC-V processor as a synchronous digital system (SDS).
- Each RV32I instruction can be done within 1 clock cycle (single-cycle CPU).

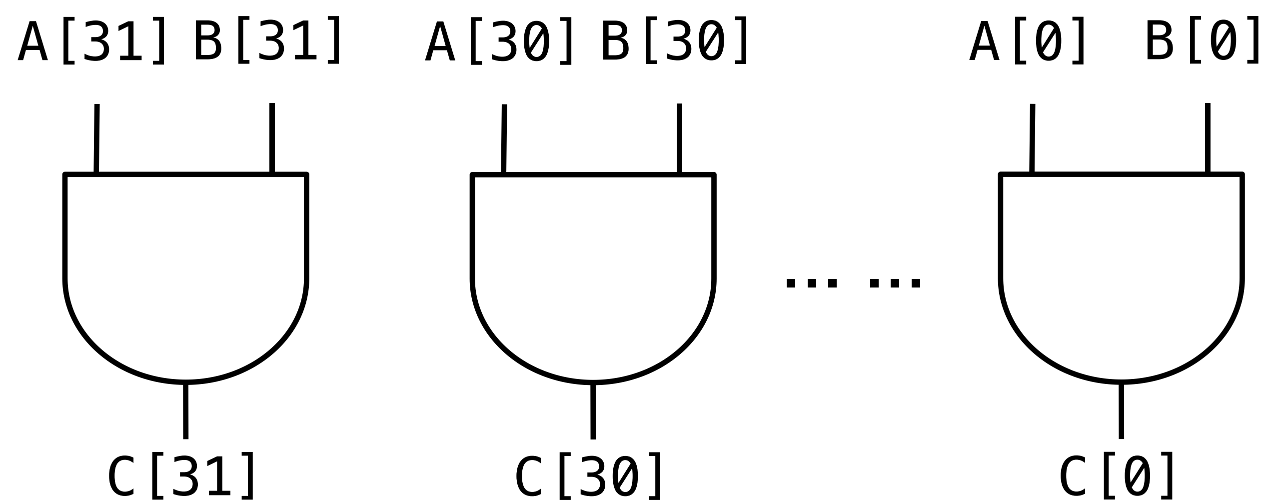
Useful building blocks

- An ALU should be able to execute all the arithmetic and logic operations



ADD	ADDI
SUB	SLTI
SLL	SLTIU
SLT	XORI
SLTU	ORI
XOR	ANDI
SRL	
SRA	
OR	
AND	

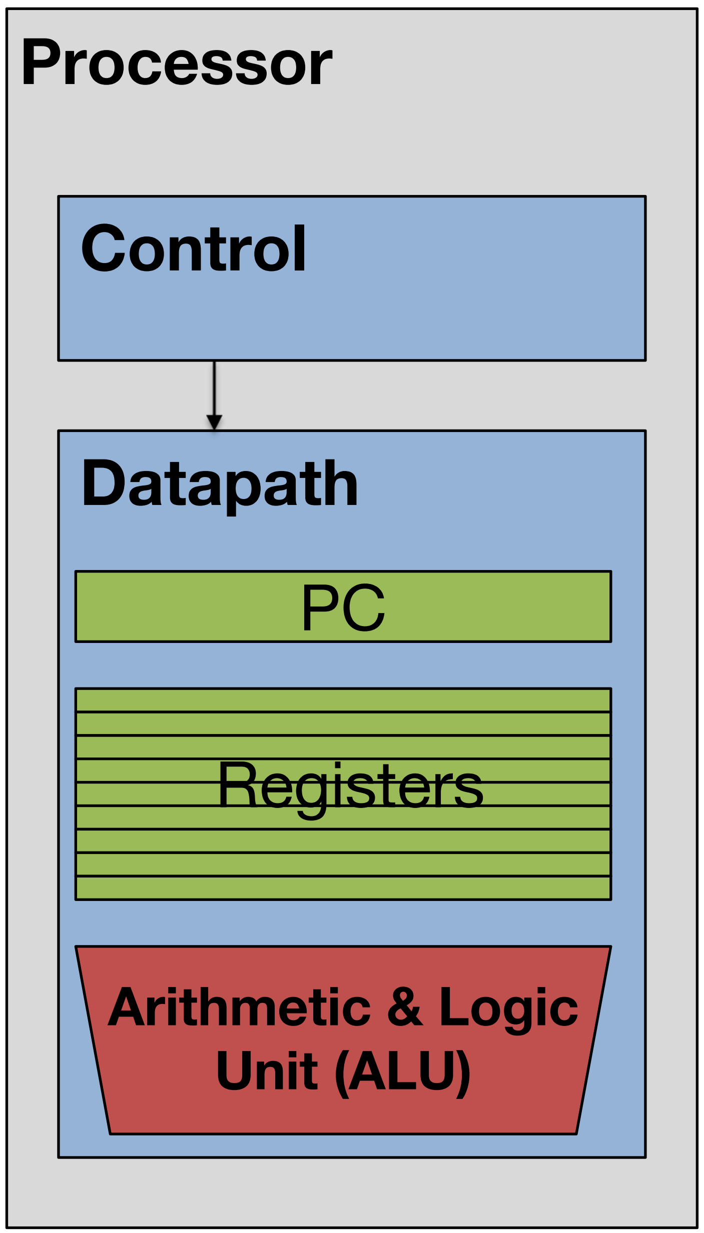
- AND as an example
  - 2 32-bit inputs A and B
  - 1 32-bit output C



- Our Goal: Implement a RISC-V processor as a synchronous digital system.
- Each RV32I instruction can be done within 1 clock cycle.

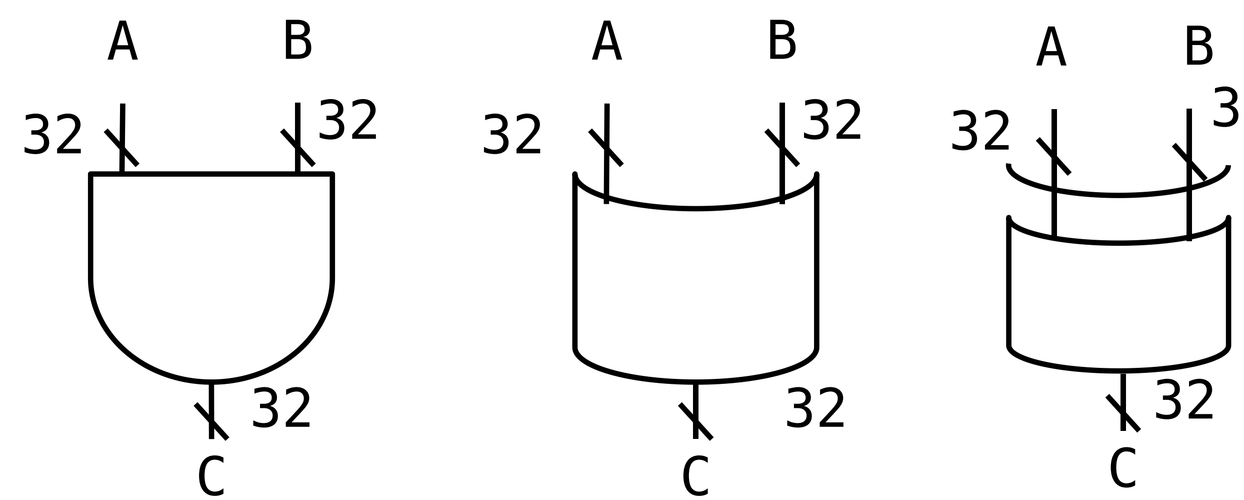
Useful building blocks

- An ALU should be able to execute all the arithmetic and logic operations



ADD	ADDI
SUB	SLTI
SLL	SLTIU
SLT	XORI
SLTU	ORI
XOR	ANDI
SRL	
SRA	
OR	
AND	

- AND as an example
  - 2 32-bit inputs A and B
  - 1 32-bit output C



A simplified AND gate array symbol

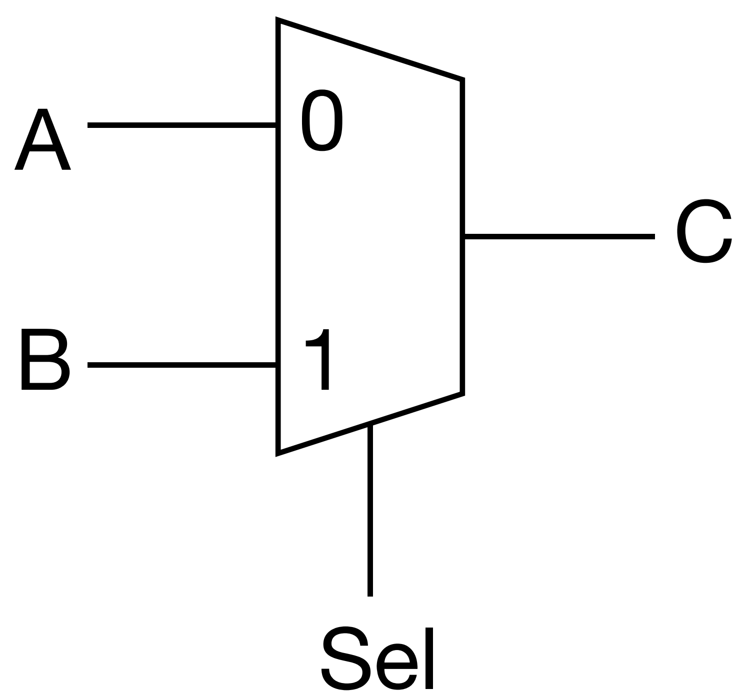
OR

XOR

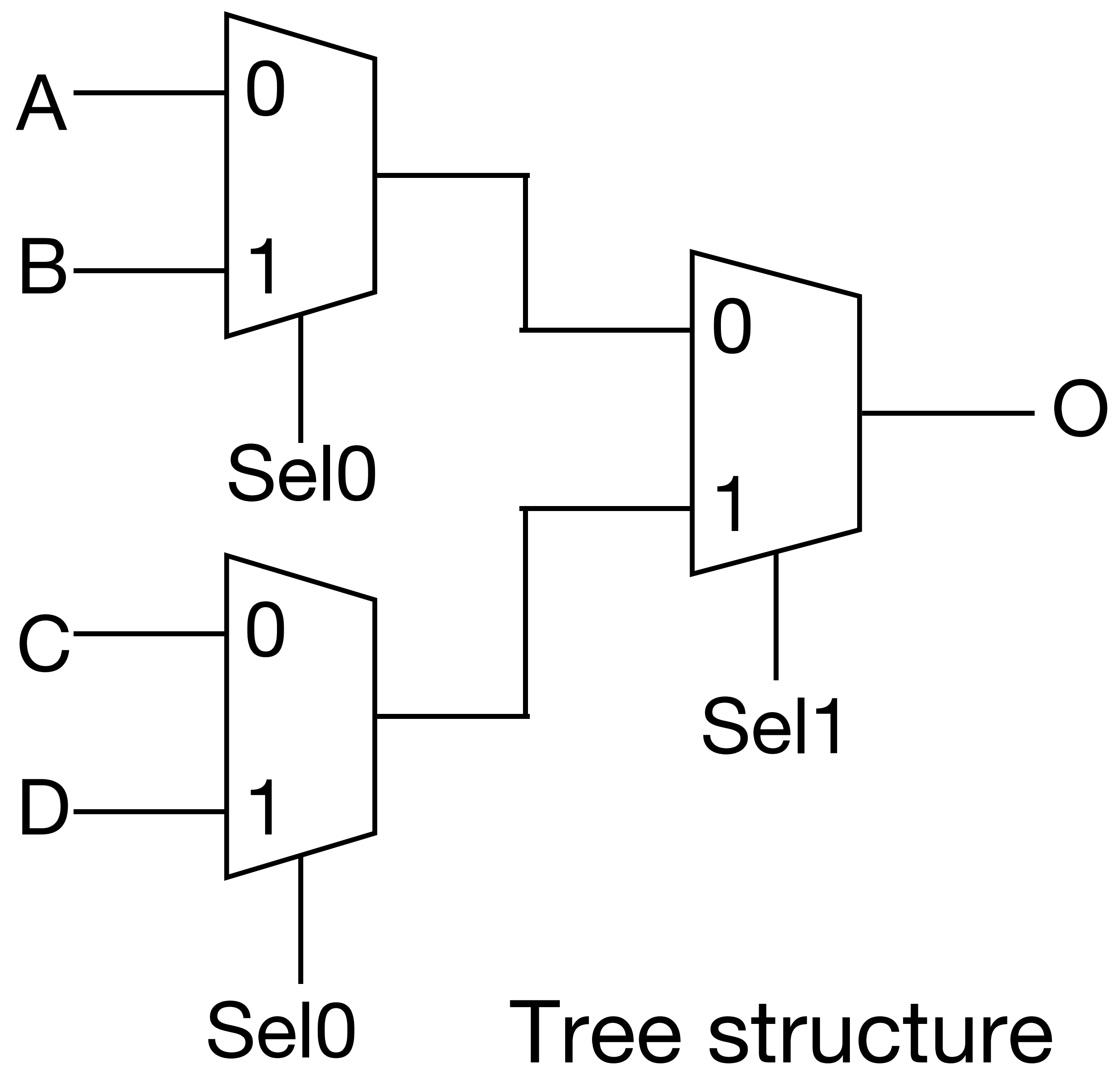
- Our Goal: Implement a RISC-V processor as a synchronous digital system.
- Each RV32I instruction can be done within 1 clock cycle.

Useful Combinational Circuits

- Multiplexer (2-to-1)



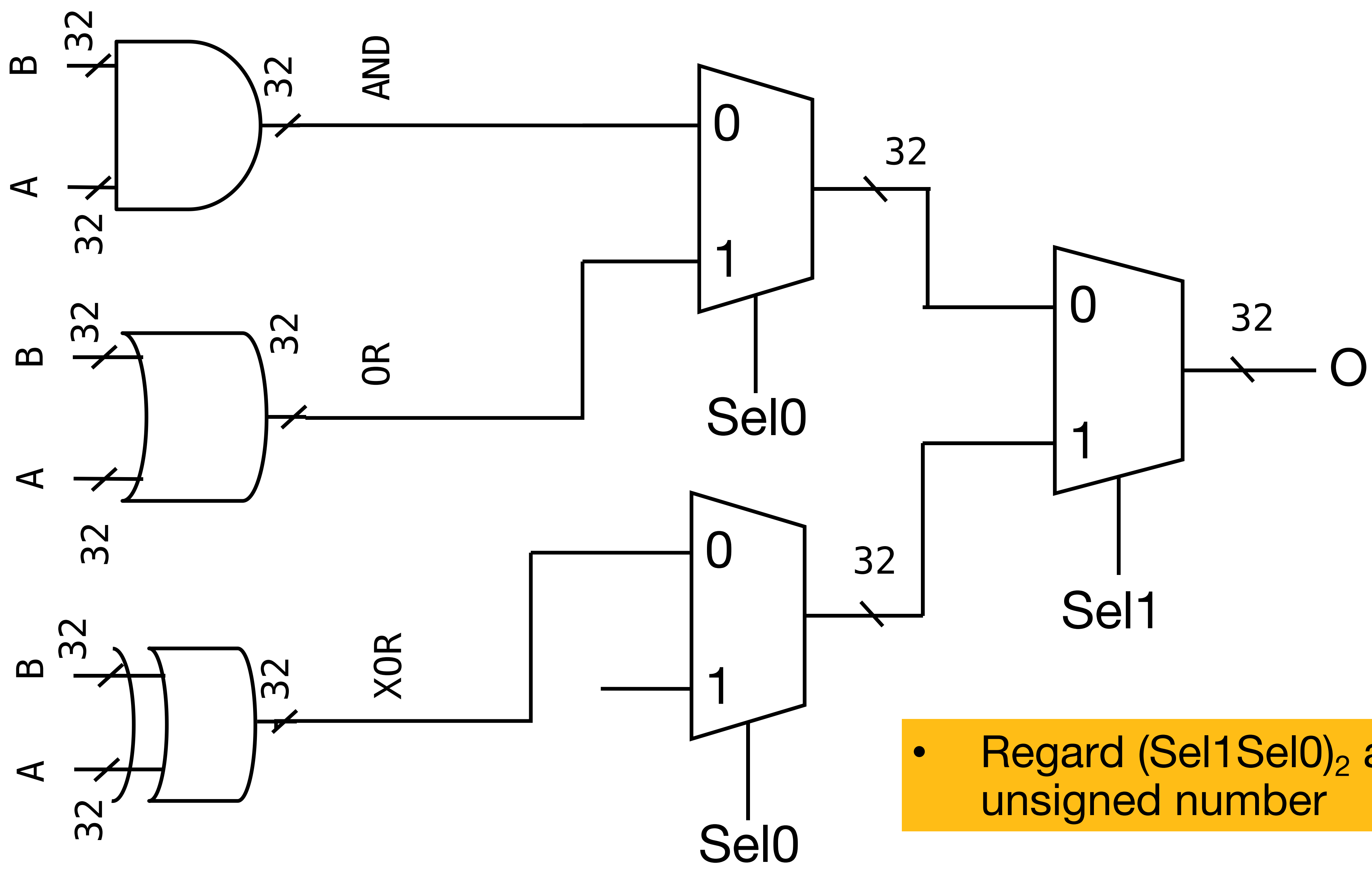
- Multiplexer (2^n-to-1)



Tree structure

Control through selection

- 32-bit Multiplexer and logic gates to support some logic instructions

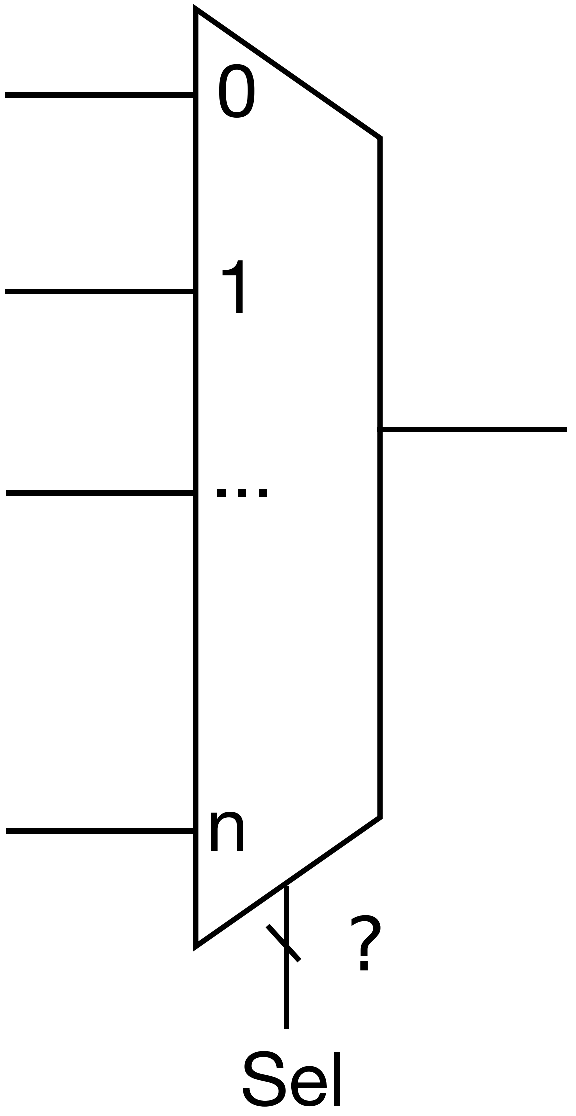


- Regard $(Sel1Sel0)_2$ as an unsigned number

- More layers of multiplexer to select from more inputs

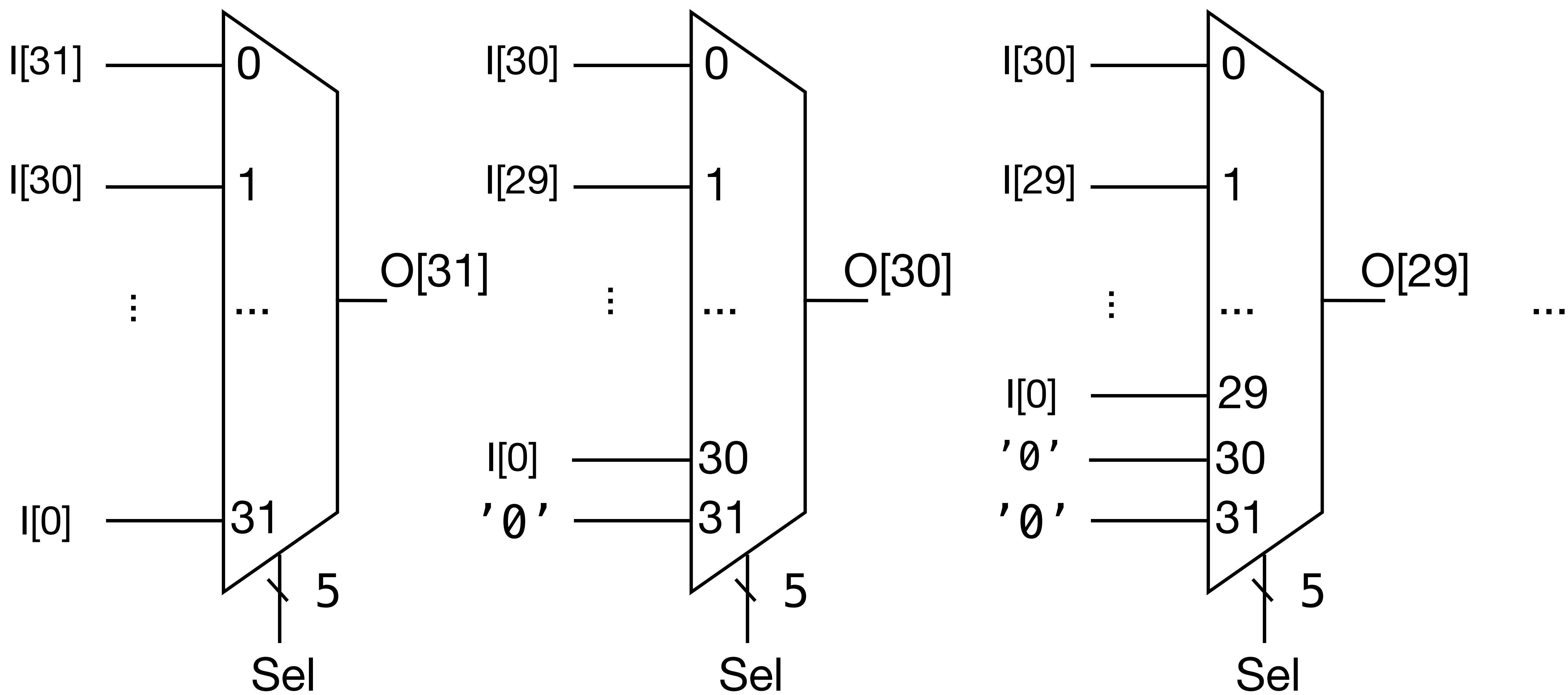
Multiplexer

- N-to-1 multiplexer symbol



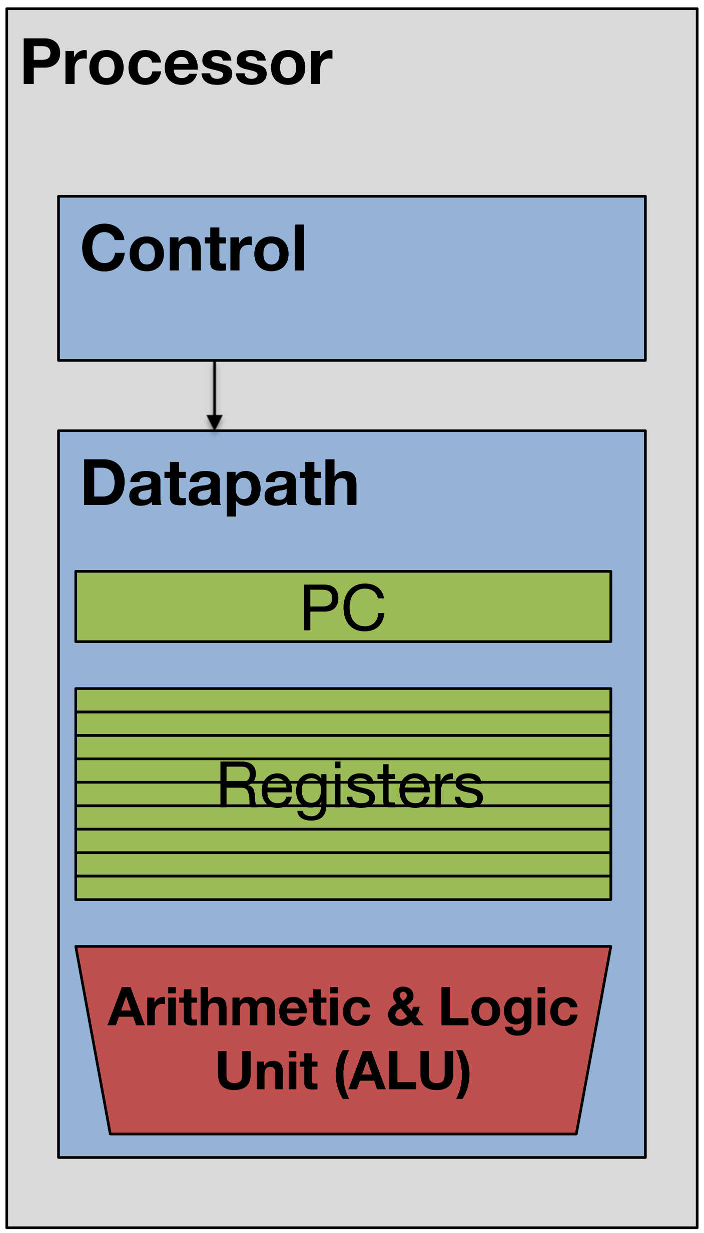
Multiplexers used for shifter

- Left shift a single bit -> left shift multiple single bits
- Other shifter designs such as barrel shifter



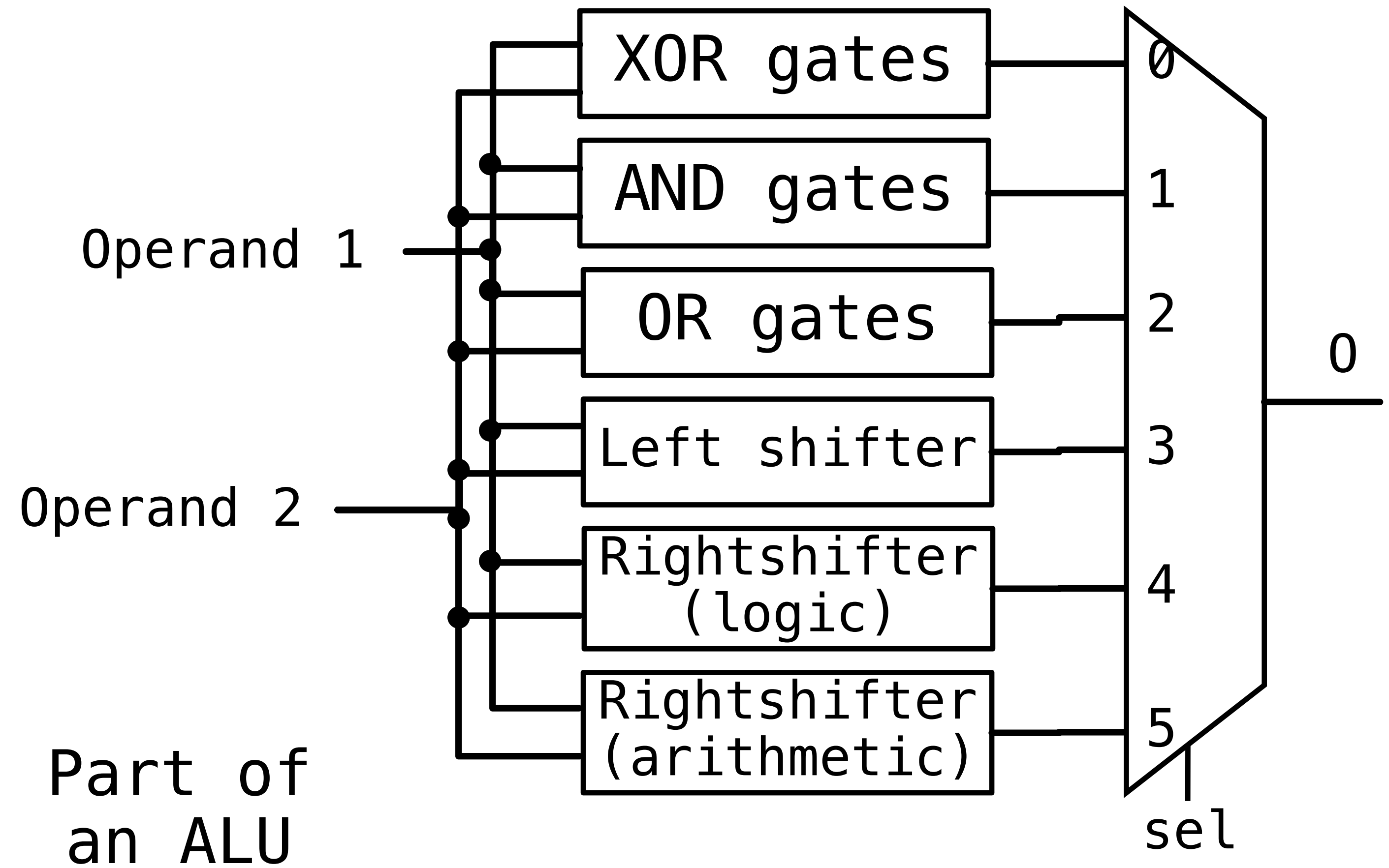
Useful building blocks

- An ALU should be able to execute all the arithmetic and logic operations



ADD
SUB
SLL
SLT
SLTU
XOR
SRL
SRA
OR
AND

ADDI
SLTI
SLTIU
XORI
ORI
ANDI

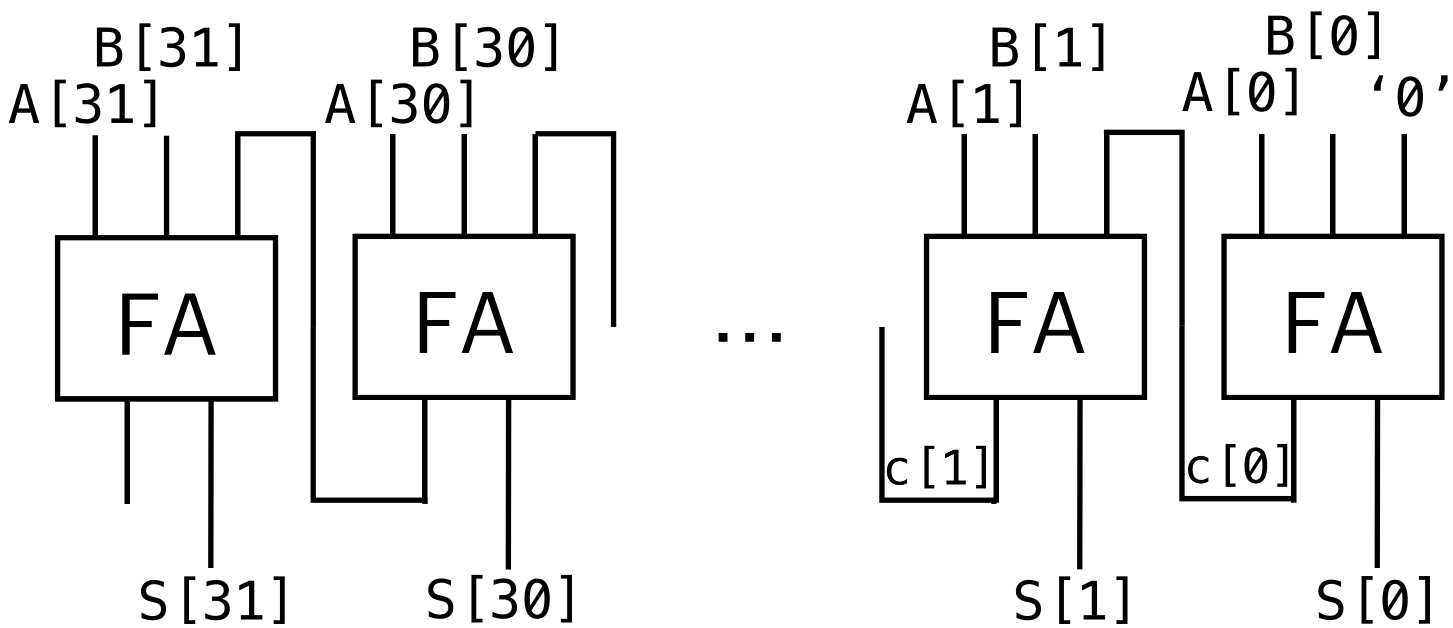


Note that all the signals expect the selection signals are 32-bit.

- Our Goal: Implement a RISC-V processor as a synchronous digital system.
- Each RV32I instruction can be done within 1 clock cycle.

Adder & subtractor

- An adder design

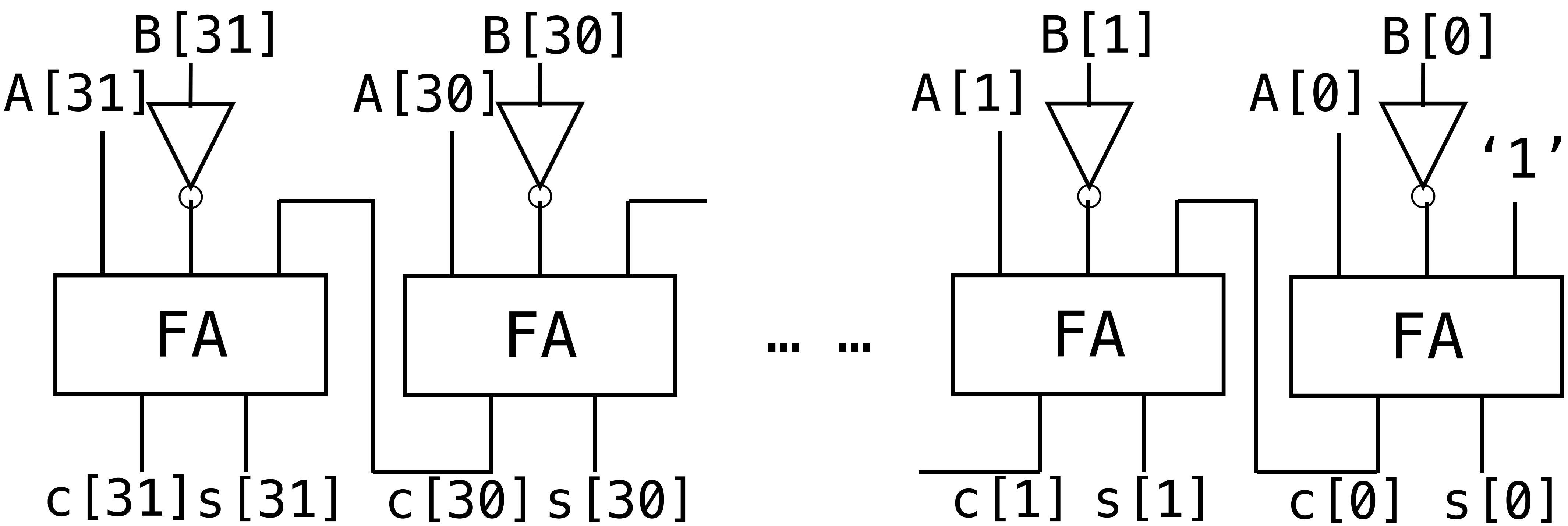


A 32-bit adder

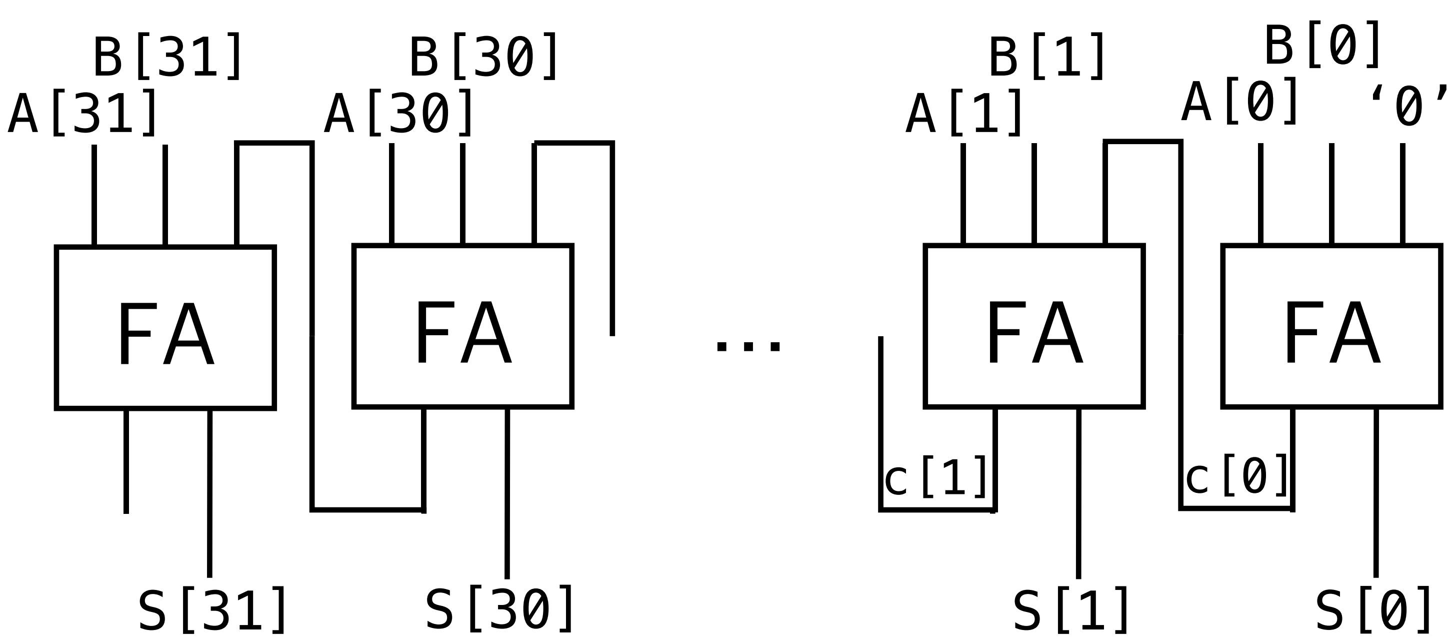
- A smart subtractor design
  - Recall that subtracting a number is equivalent to adding its negative version

A smart subtractor design

$$\mathbf{A - B = A + (-B) = A + \bar{B} + 1 \quad (mod\ 2^{N-1})}$$

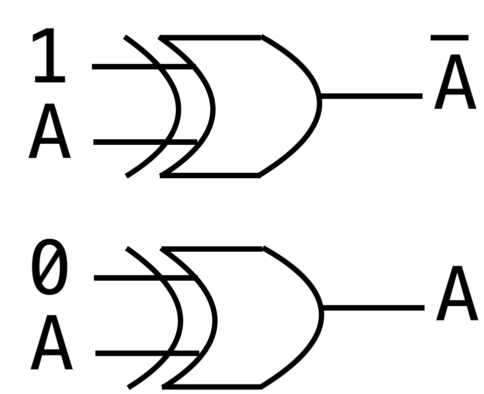


A 32-bit subtractor



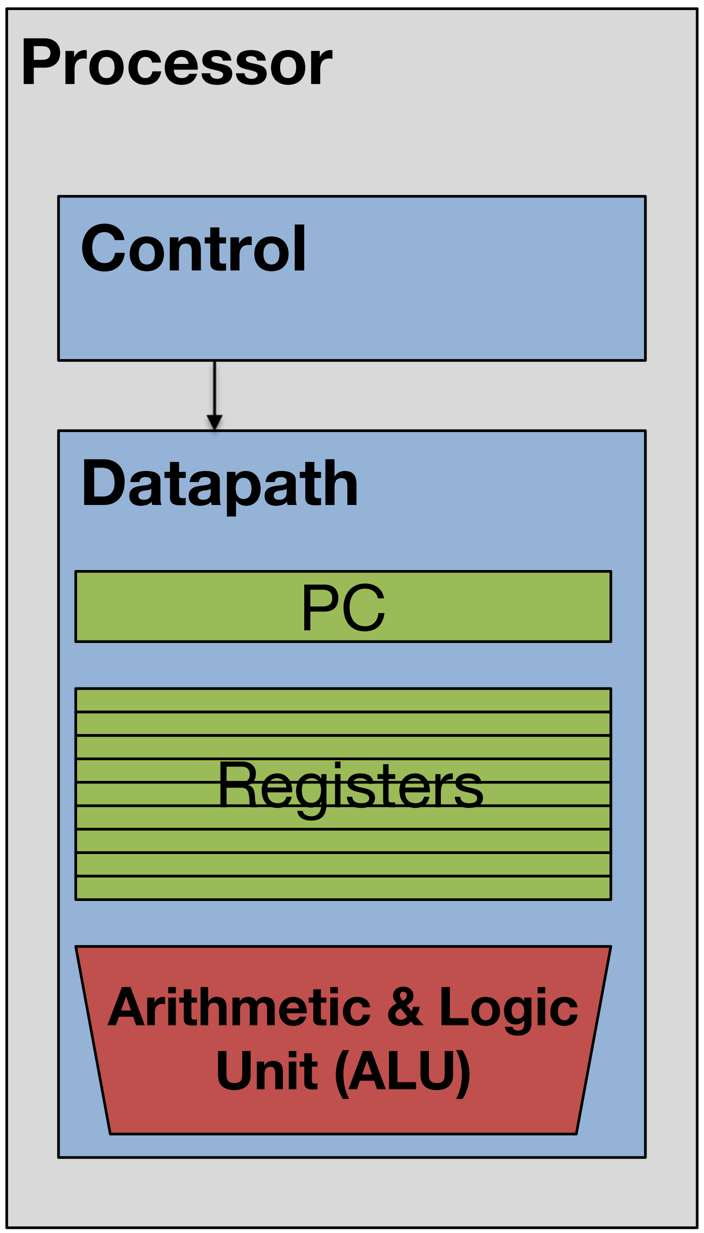
A 32-bit adder

- Recall XOR gate



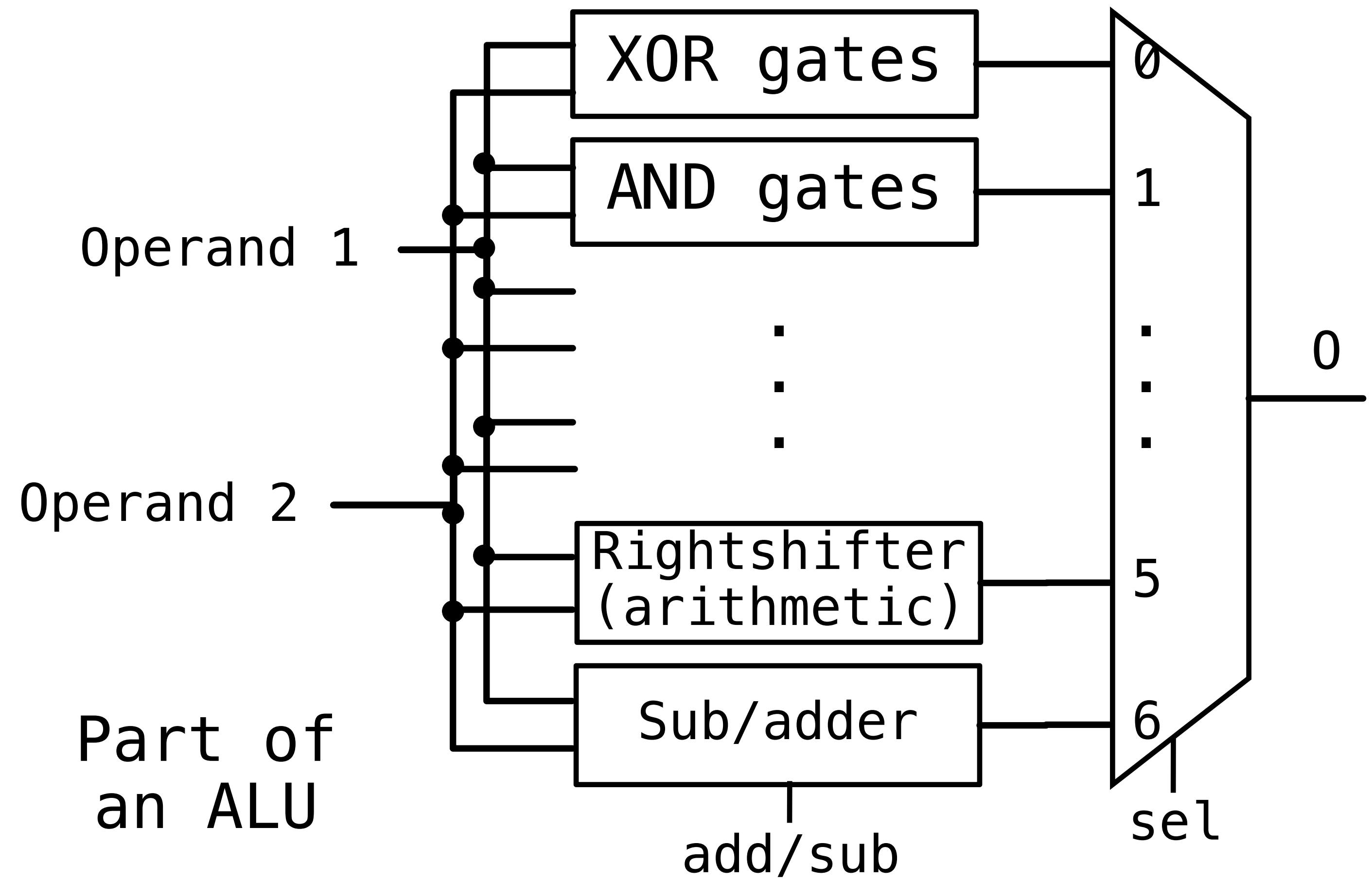
Useful building blocks

- An ALU should be able to execute all the arithmetic and logic operations



ADD
SUB
SLL
SLT
SLTU
XOR
SRL
SRA
OR
AND

ADDI
SLTI
SLTIU
XORI
ORI
ANDI

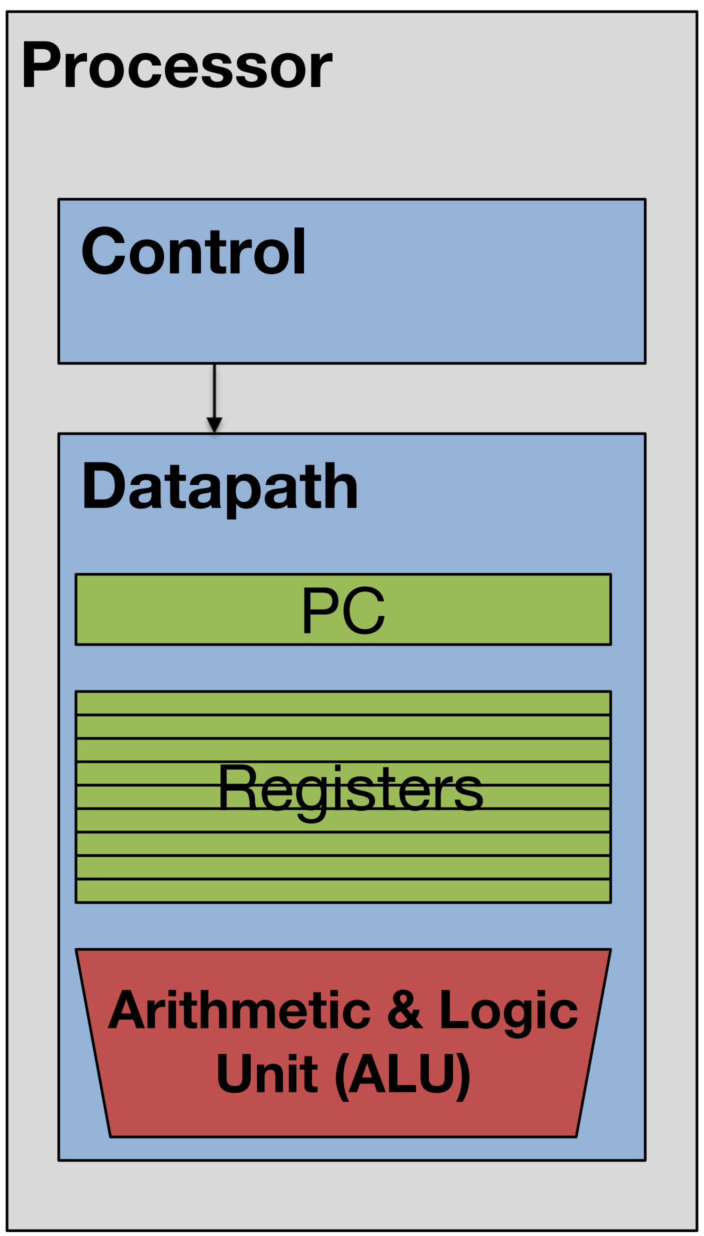


Note that all the signals expect the selection signals are 32-bit.

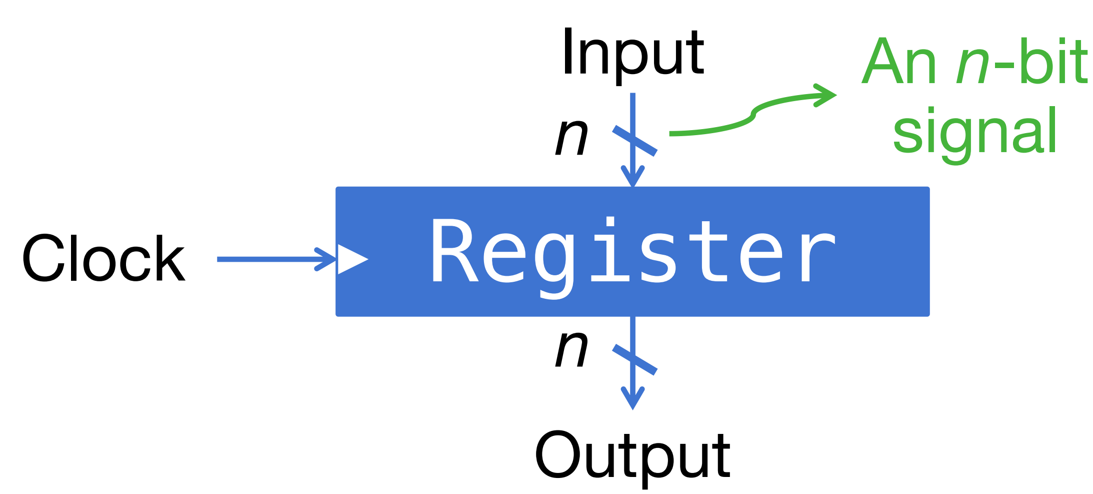
- ALU design that supports R-/I-arithmetic and logic operations completed

Useful building blocks-Register file

- The register file is the component that contains all the general purpose registers of the microprocessor
- A register file should provide data given the register numbers
- A register file should be able to change the stored value

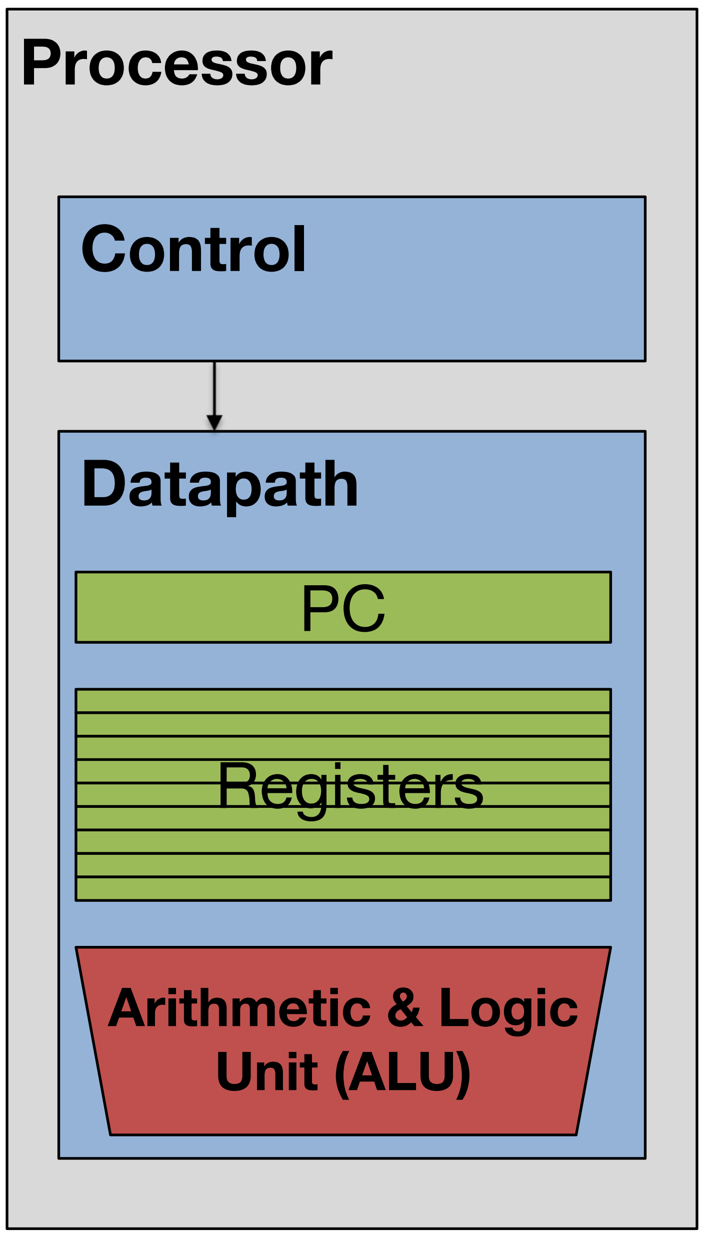


- Recall we have registers that store values

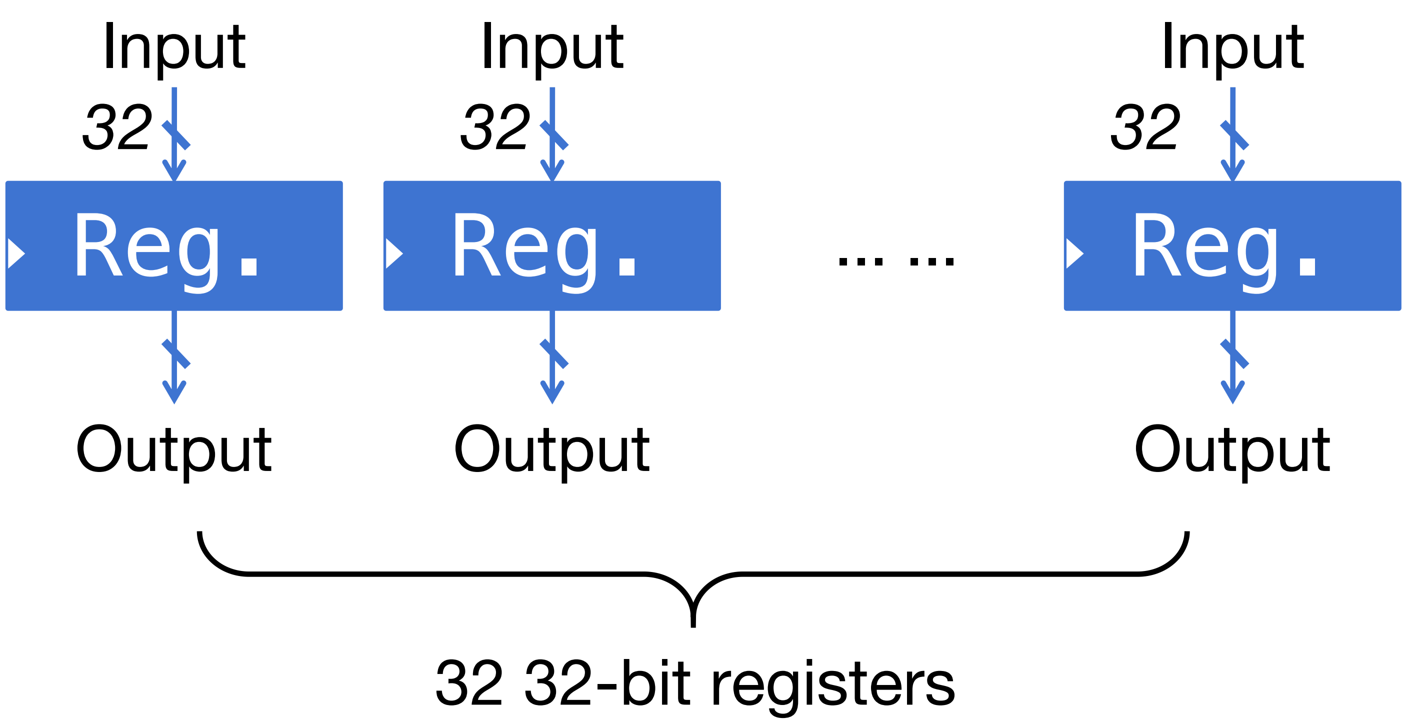


Useful building blocks-Register file

- The register file is the component that contains all the general purpose registers of the microprocessor
- A register file should provide data given the register numbers
- A register file should be able to change the stored value



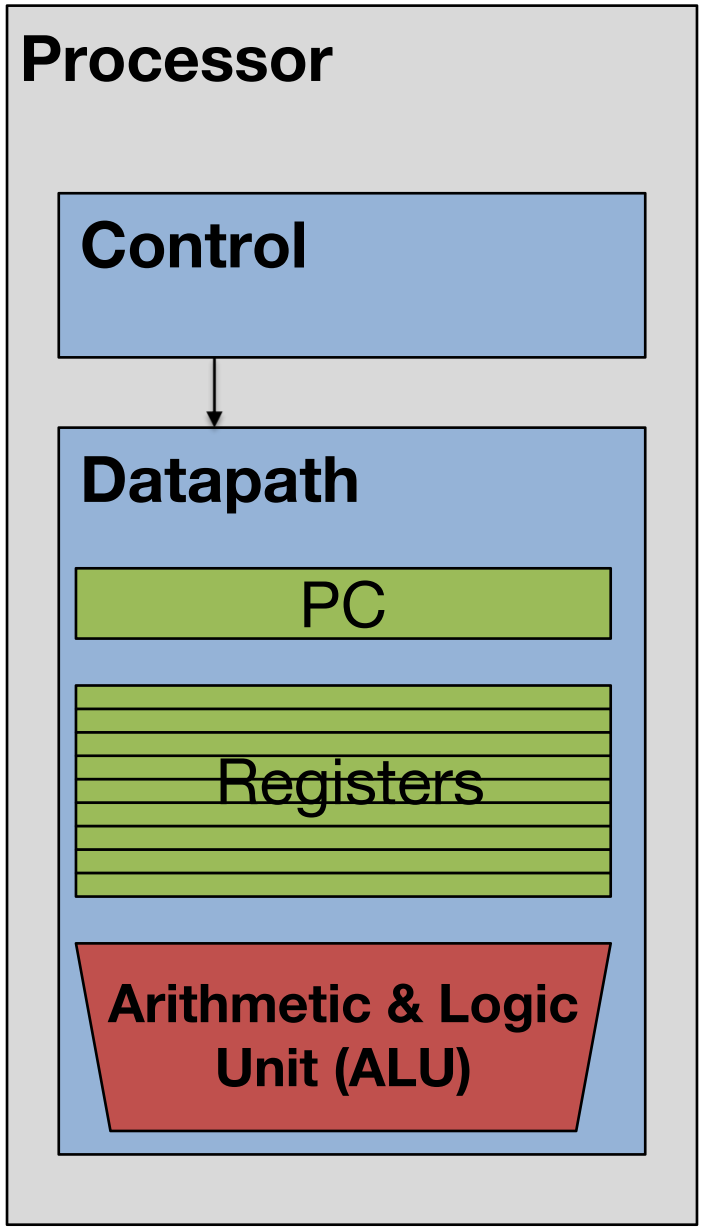
- Recall we have registers that store values



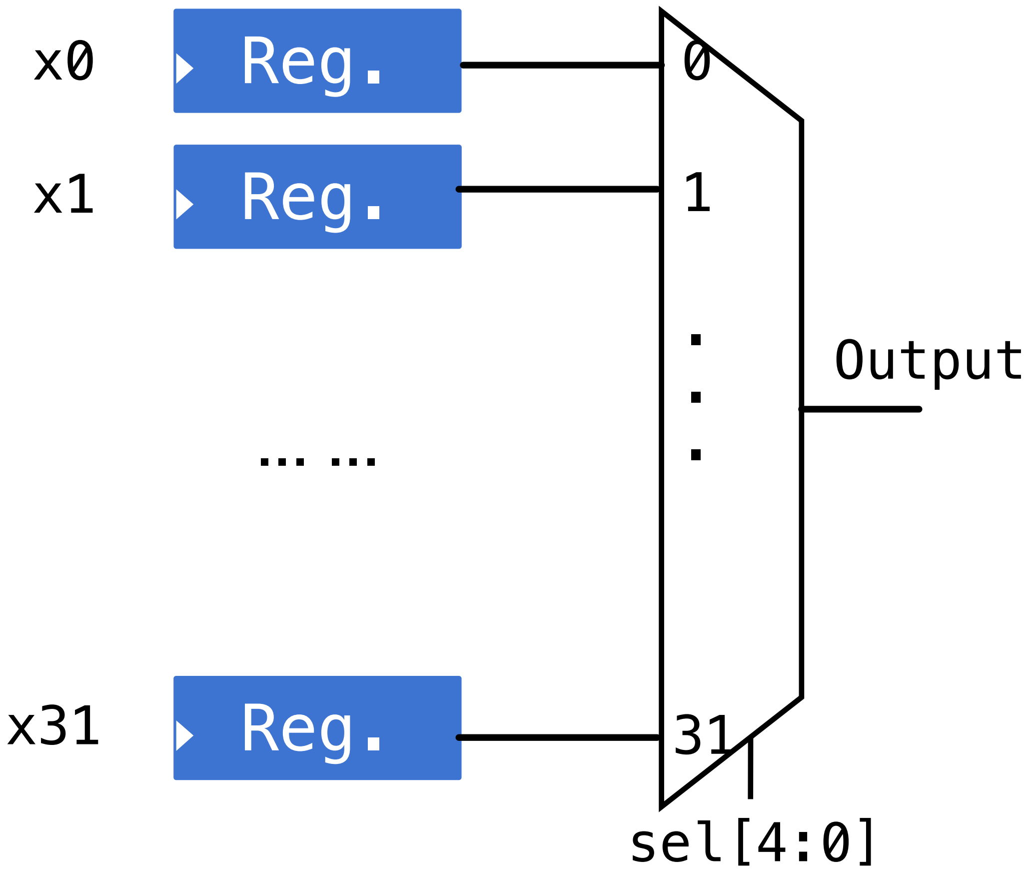
- How to select one to output? Multiplexer

Useful building blocks-Register file

- The register file is the component that contains all the general purpose registers of the microprocessor
- A register file should provide data given the register numbers
- A register file should be able to change the stored value



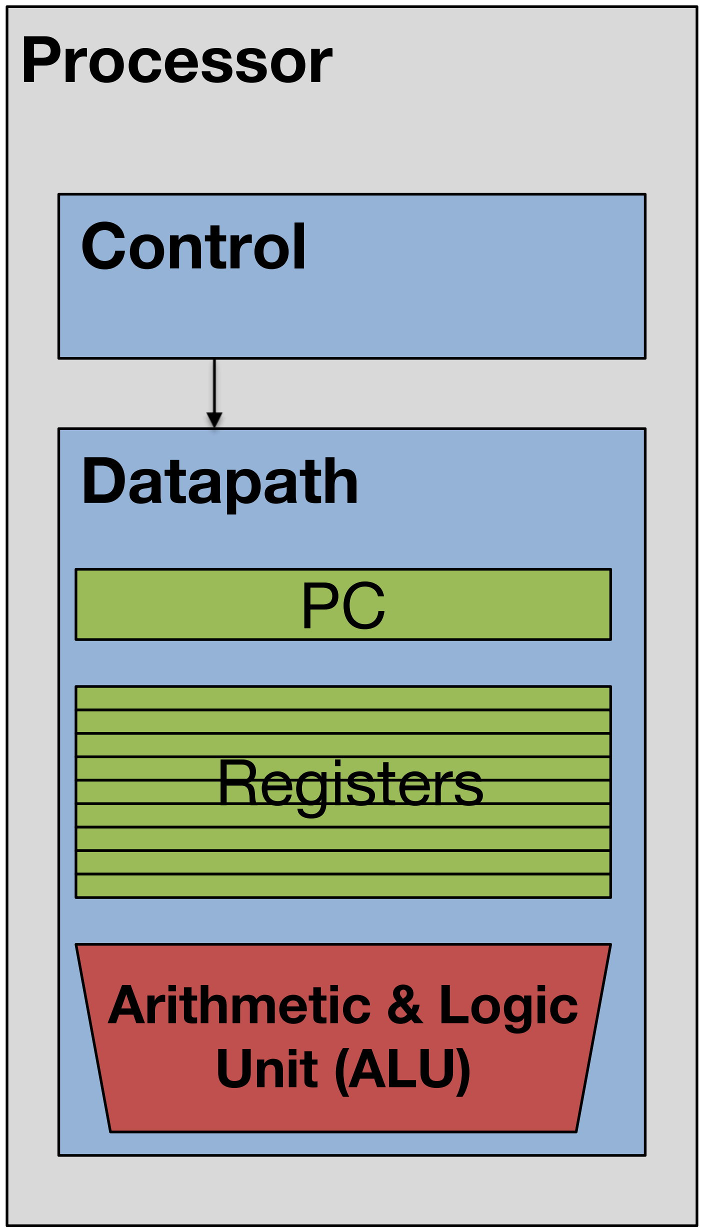
- Recall we have registers that store values



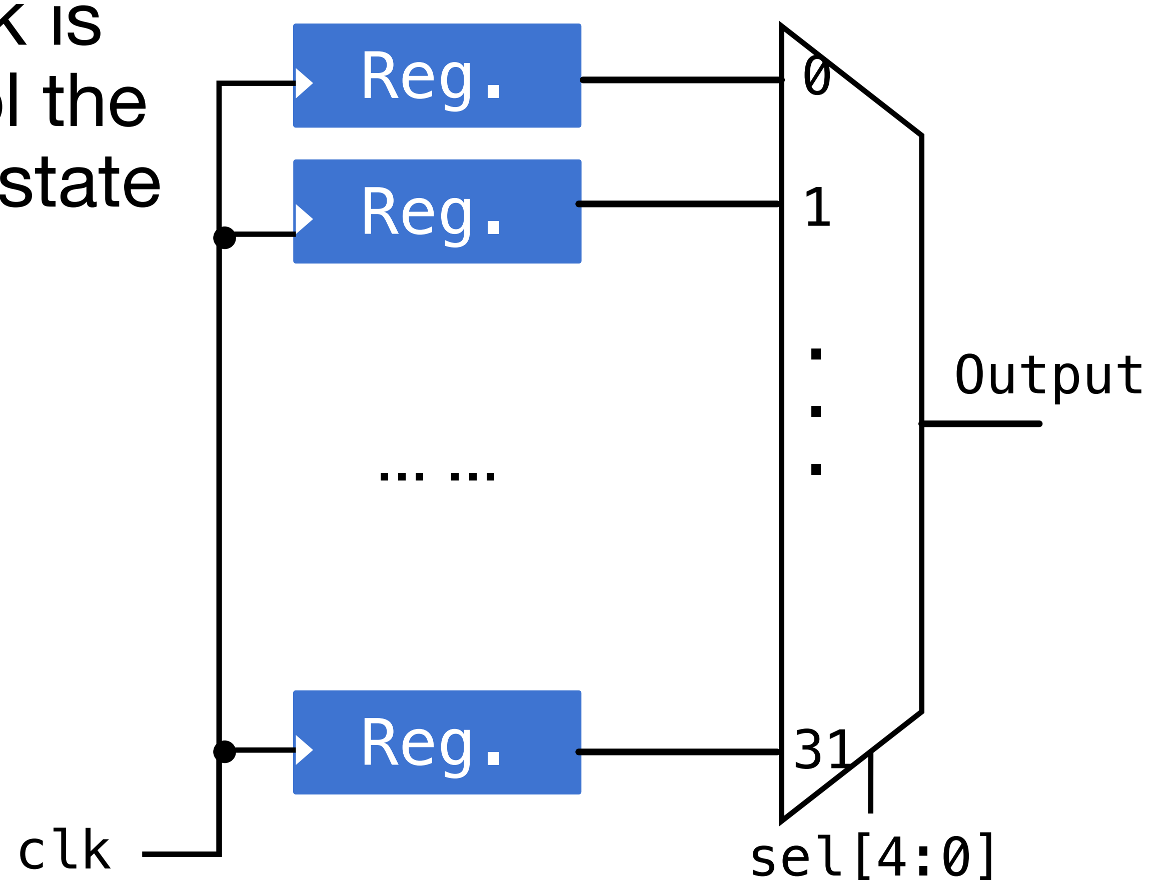
Multiplexer

Useful building blocks-Register file

- The register file is the component that contains all the general purpose registers of the microprocessor
- A register file should provide data given the register numbers
- A register file should be able to change the stored value

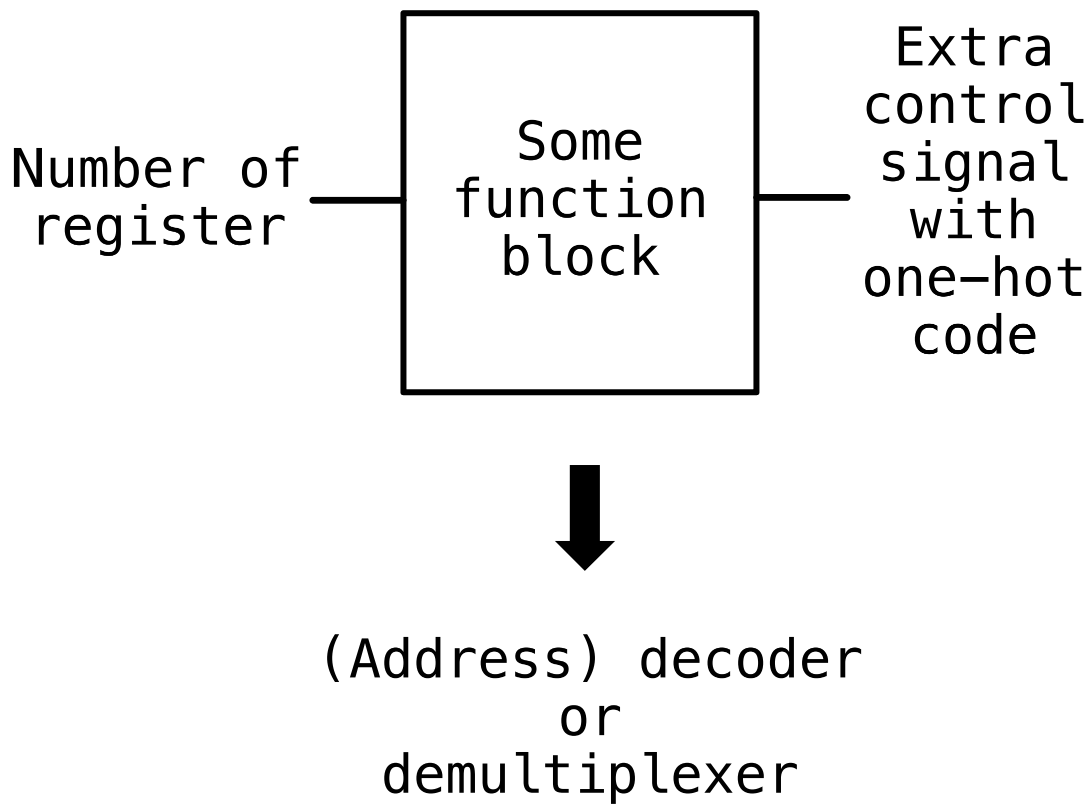


- How do we change values of a specific reg.?
- Recall that `clk` is used to control the change of the state

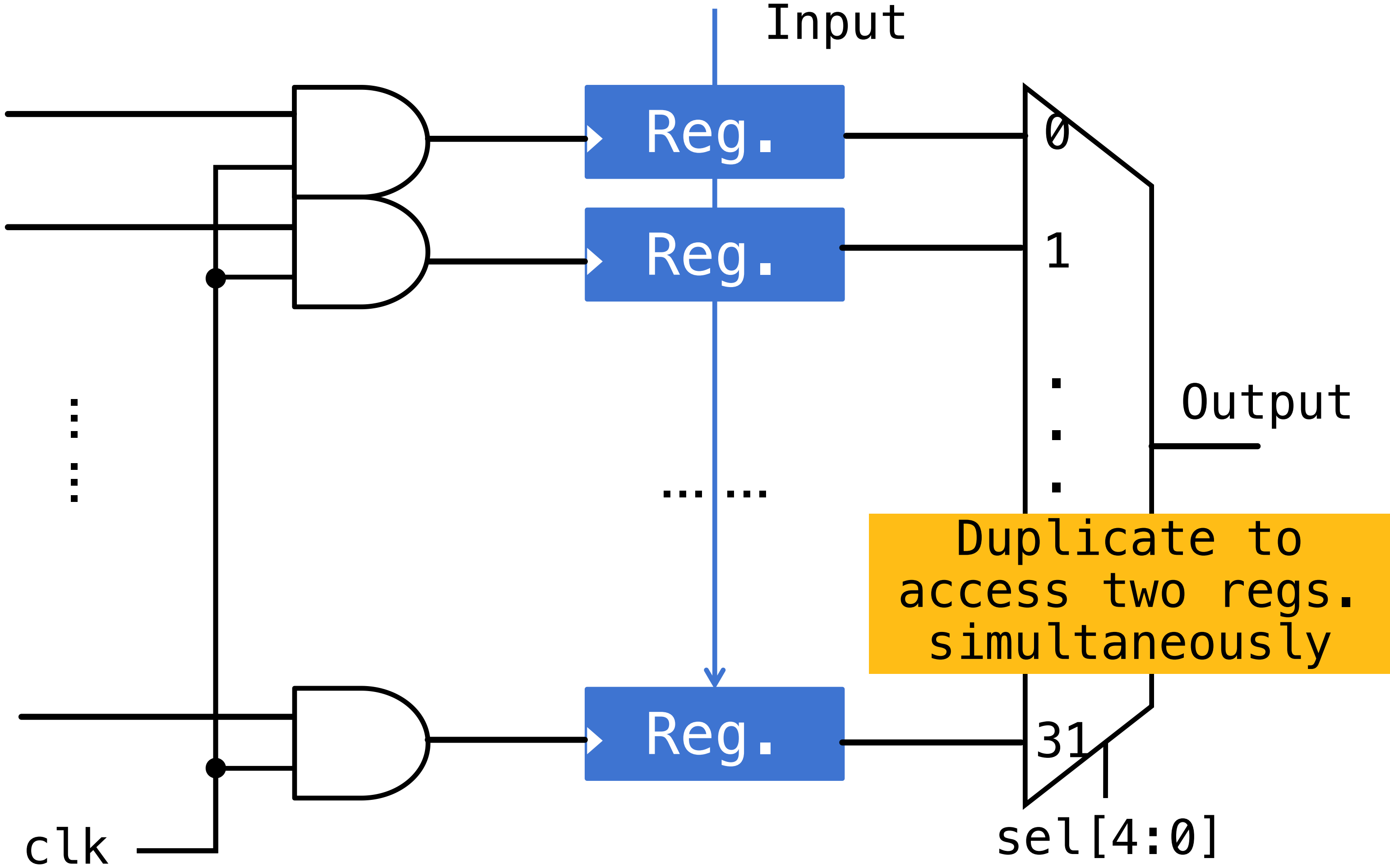


Useful building blocks-Register file

- The register file is the component that contains all the general purpose registers of the microprocessor
- A register file should provide data given the register numbers
- A register file should be able to change the stored value



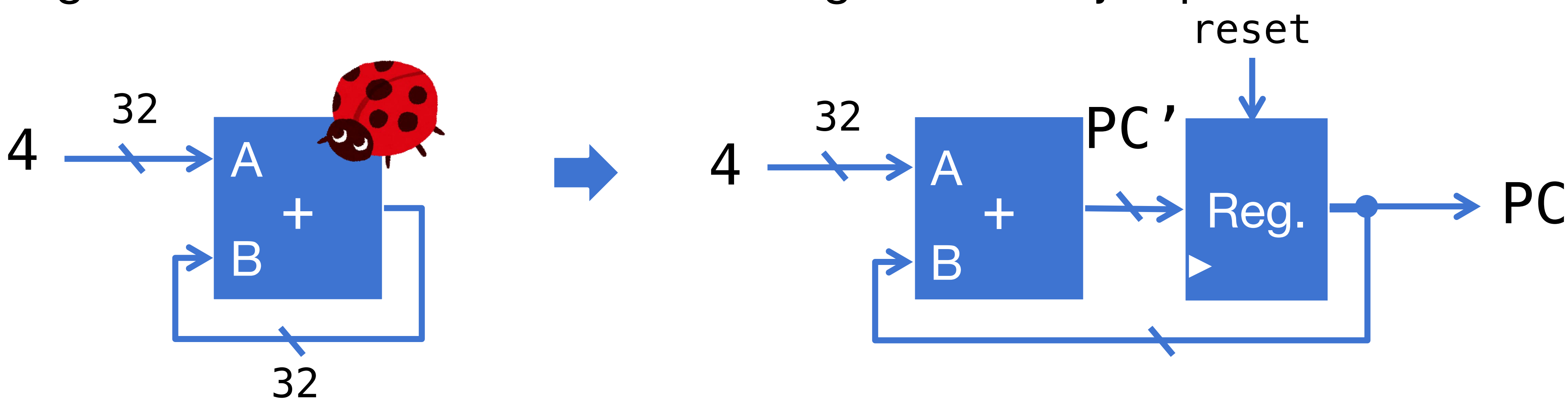
- How do we change values of a specific reg.?



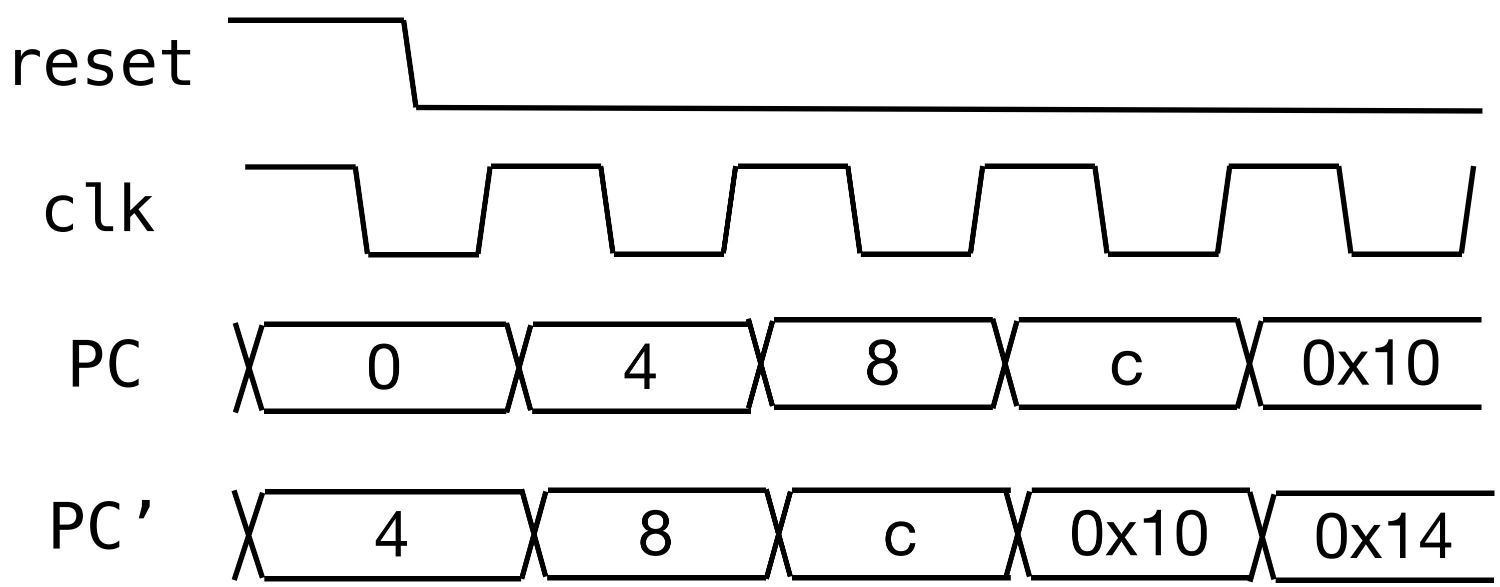
- Reg. file design completed

We have covered PC register previously

- Synchronous digital circuit can have feedback, e.g., iterative accumulator
  - e.g. PC = PC + 4 without considering branch or jump



- Timing diagram



Useful building blocks-Memory

- Memory similar to register file except that the basic cell design is different
- Requires refresh for DRAM