50.002 Computation Structures
Computation Models & Programmable Machines

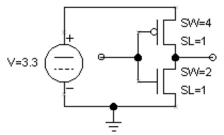
Oliver Weeger

2018 Term 3, Week 4, Session 1

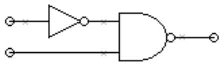
SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

INFORMATION SYSTEMS TECHNOLOGY AND DESIGN

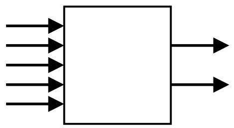
Where are we? The 50.002 roadmap



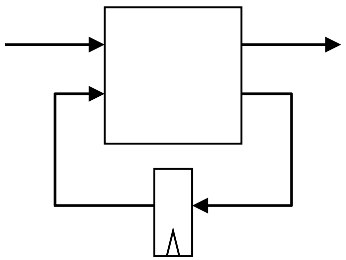
Digital abstraction, Fets & CMOS, Static discipline



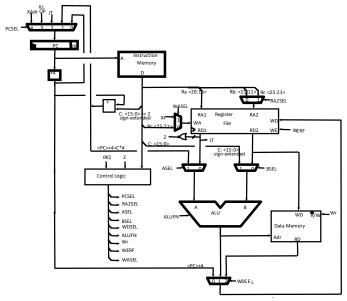
Logic gates & Boolean Algebra (AND, OR, NAND, NOR, etc.)



Combinational logic circuits: Truth tables, Multiplexers, ROMs



Sequential logic & Finite State Machines: Dynamic Discipline, Registers, State Transition Diagrams



CPU Architecture: *interpreter* for coded *programs*

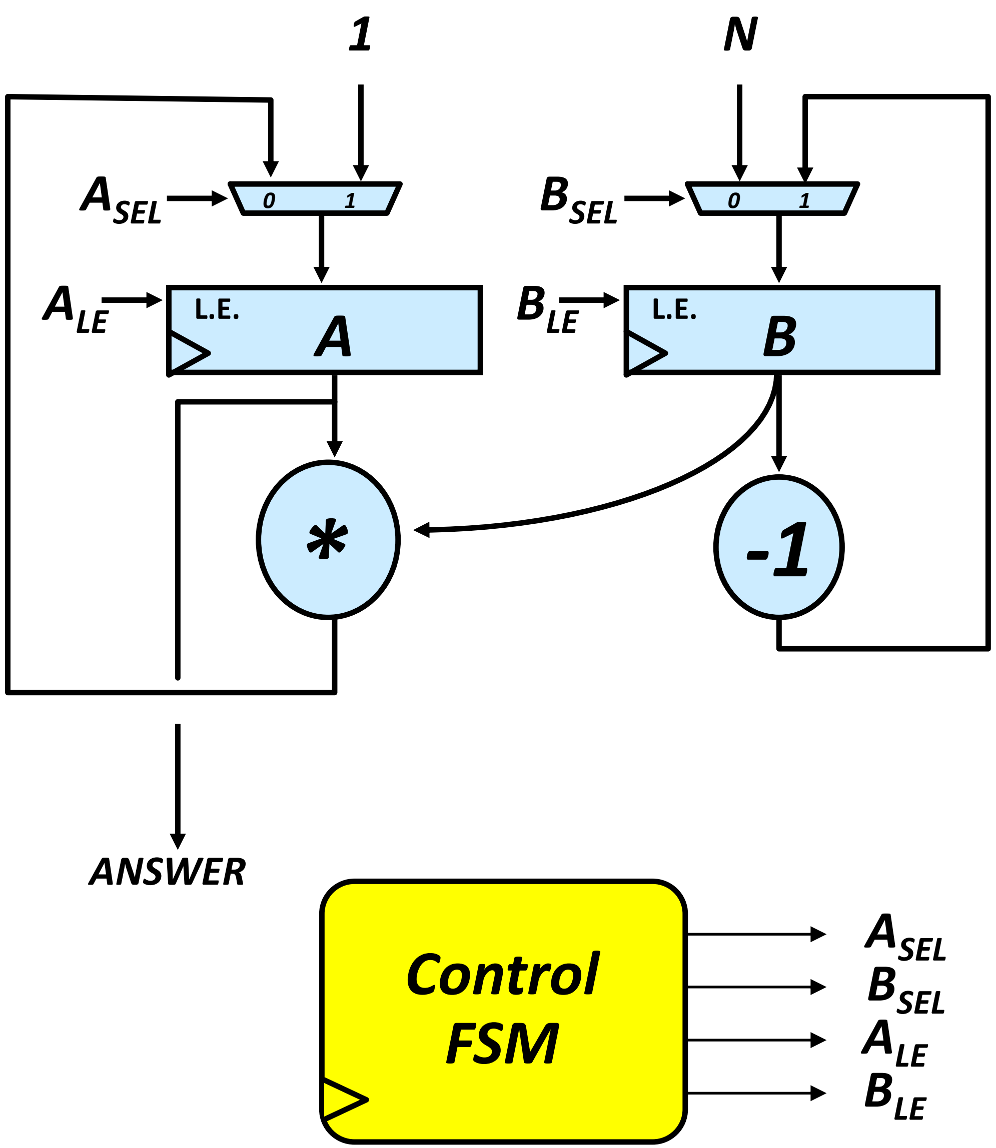
Use logic to compute mathematical functions, e.g. sums, %3
→ Limitations and models (computability & universality)

Programmability:

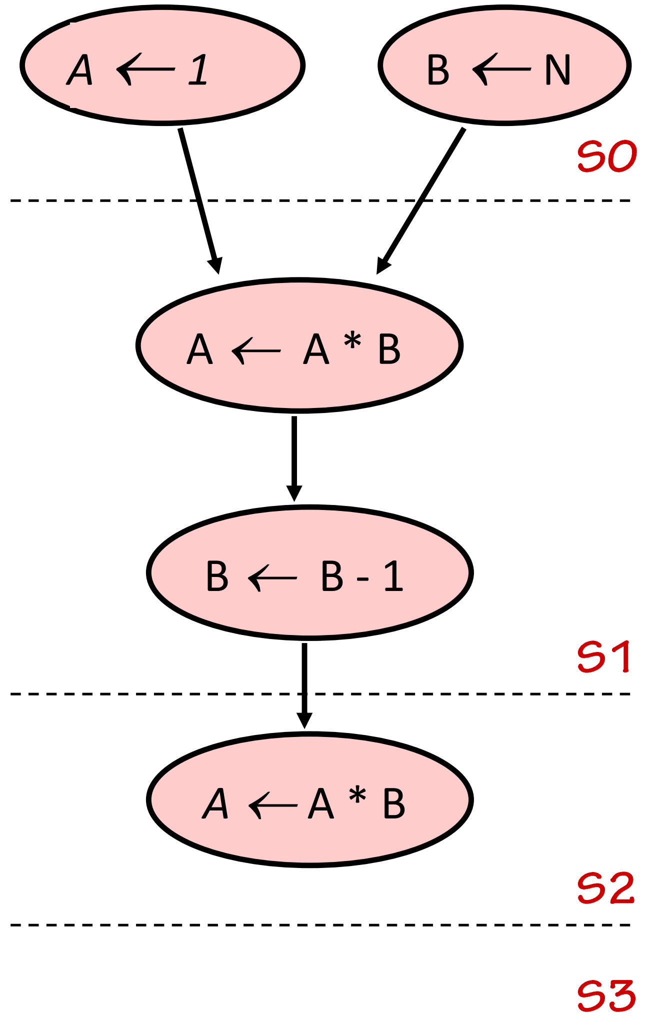
- General-Purpose Computer
- Instruction Set Architecture
- Interpretation, Programs, Languages, Translation
- Beta implementation

Why do we need programmability? Computing $N \cdot (N-1)$ ...

Data path and Control FSM:



Multi-step process:

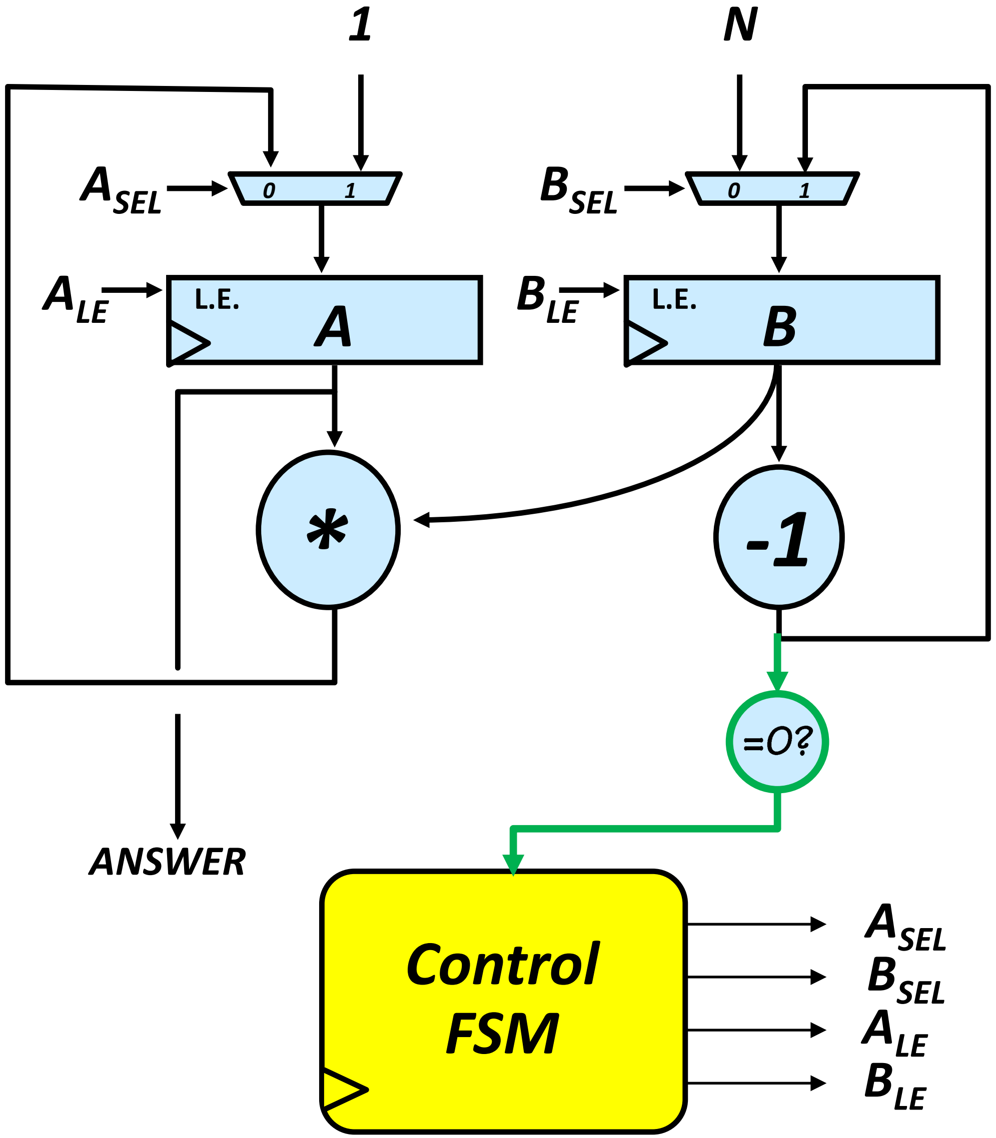


Control program table:

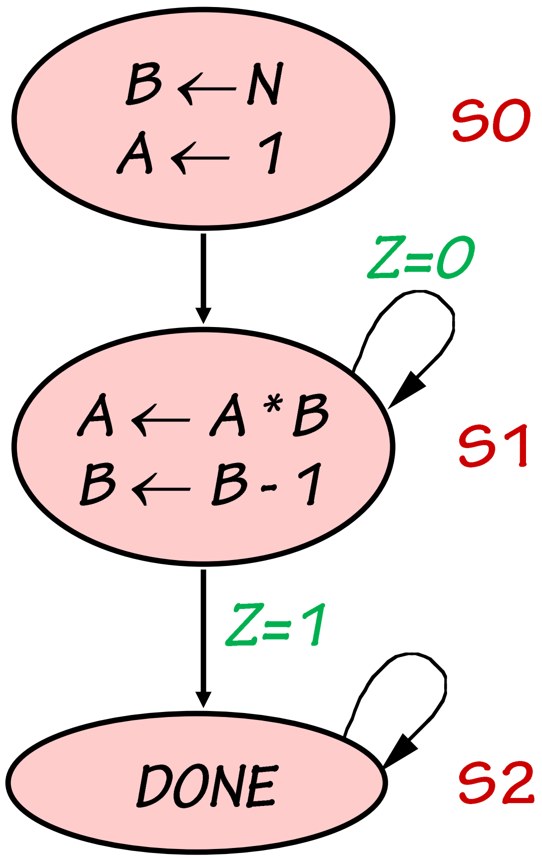
S_N	S_{N+1}	A_{sel}	A_{LE}	B_{sel}	B_{LE}
0	1	1	1	0	1
1	2	0	1	1	1
2	3	0	1	0	0
3	3	0	0	0	0

Why do we need programmability? Computing $N!$ ...

Data path and Control FSM:



Multi-step process:

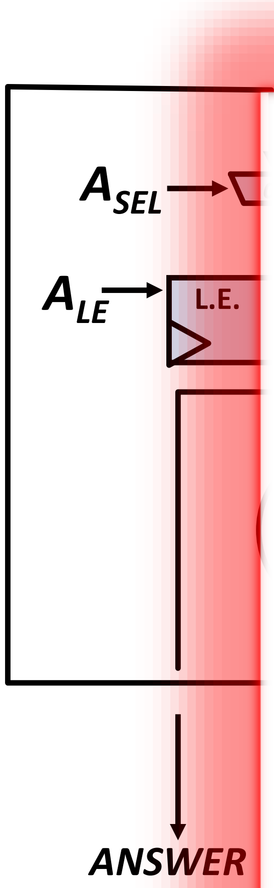


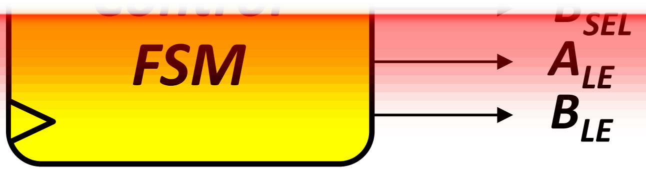
Control program table:

Z	S_N	S_{N+1}	A_{sel}	A_{LE}	B_{sel}	B_{LE}
-	0	1	1	1	0	1
0	1	1	0	1	1	1
1	1	2	0	1	1	1
-	2	2	0	0	0	0

Why do we need programmability? Computing $N!$ ...

Data path and Control FSM:





Multi-step process:

Control program table:

Programmable control system:

- Control processing at each step with an FSM
- Allow different control sequences to be loaded into the control FSM
- Re-use data path and reconfigure FSM to compute new functions

Short-comings:

- Not “general” yet, has only a tiny repertoire of operations
- “Program” is fixed, cannot generate and execute a new program
- Has only a limited amount of storage

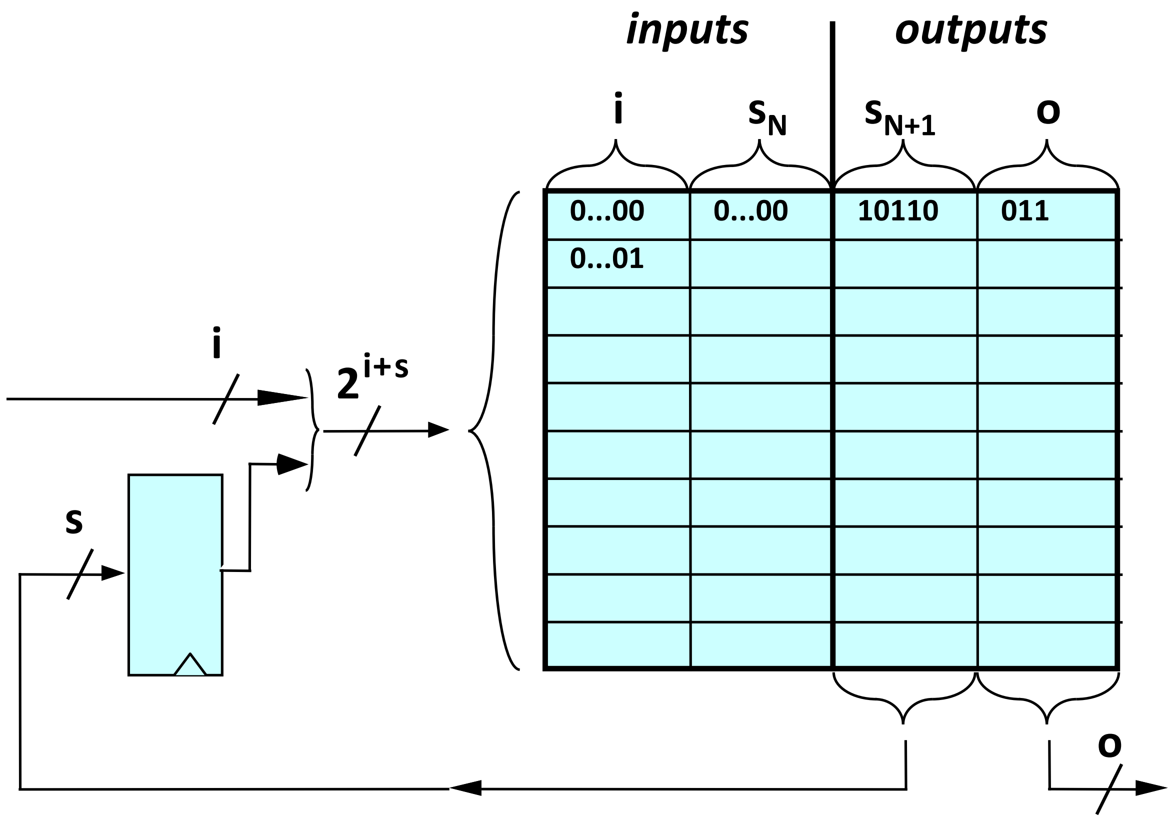
A_{LE}	B_{sel}	B_{LE}
1	0	1
1	1	1
1	1	1
0	0	0

Finite State Machines: Enumeration

FSM with i inputs, o outputs, s states:

→ ROM/truth table has 2^{i+s} rows (words) with $o + s$ columns (bits) each

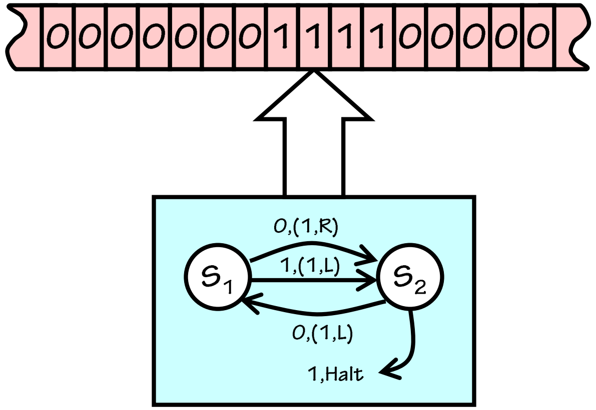
→ $\boxed{\text{Potentially } 2^{(o+s)2^{i+s}} \text{ different FSMs}}$



i	s	o	FSM#	Truth Table	
1	1	1	1	00000000	
1	1	1	2	00000001	2^8 FSMs
			…	…	
1	1	1	256	11111111	
2	2	2	257	000000…000000	
2	2	2	258	000000…000001	2^{64} FSMs
			…	…	
3	3	3		000000…000000	
			…	…	
4	4	4		000000…000000	
			…	…	

Finite State Machines → Turing Machines

- Limitation of FSM: cannot solve problems that require arbitrarily many states
- Turing Machine:
  - FSM combined with doubly-infinite tape
  - Can read & write at tape in every step
- Can solve problem with infinitely many states, e.g. parentheses checking for any string



<u>Problem</u> (work on it for 5 min with your neighbour):

- Is there an FSM that can determine whether so far an odd number of 1s and an even number of 0s have been entered?
- Can you draw a state-transition diagram for such an FSM with 4 states?

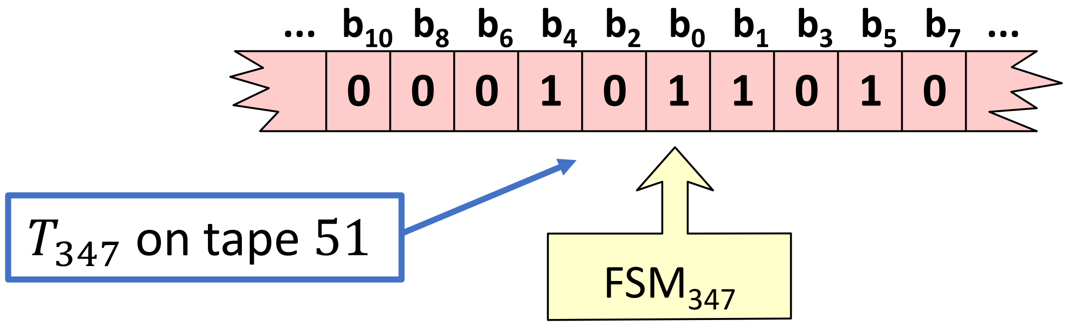
Turing machine for parentheses checking

- Check if a string of arbitrary length contains a well-formed set of parentheses, e.g.
  “(() ()) ()” ✓ “(()” ✗ “())” ✗
- Counting of “(“ and “)” leads to infinitely many states → not solvable with FSM
- Representation of string on infinite tape as: Ø (( ) ( )) ( ) Ø
- Turing machine truth table
- Interpretation of states:
  - S0: Search for “)” or “Ø” to the right
  - S1: Search for “(“ to the left
  - S2: Search for “Ø” to the left
  - S3: Halt
- Can be easily extended to “proper” strings with other characters or nested types of brackets

Curr. state	Read	→	Write	Move	Next state
S0	(	→	(	R	S0
S0	)	→	x	L	S1
S0	x	→	x	R	S0
S0	Ø	→	Ø	L	S2
S1	(	→	x	R	S0
S1	x	→	x	L	S1
S1	Ø	→	0	H	S3
S2	(	→	0	H	S3
S2	x	→	x	L	S2
S2	Ø	→	1	H	S3

Turning machines …

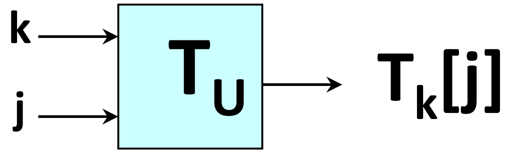
- Can be given canonical names for bounded tape configurations



- Can be used to compute integer functions:
  $$y = T_k[x]$$
  (k: FSM index, x: input tape configuration,
  y: output tape configuration)
  But not all integer functions can be computed using Turing machines
- <u>Computable functions:</u>
  $f(x)$ computable $\Leftrightarrow \exists k\!: \forall x\!:\; f(x) = T_k[x] = f_k(x)$
- Church-Turing Hypothesis: any computable function is computable by a TM
- Uncomputable functions (e.g. Halt function) cannot be computed by a TM

→ Special-purpose Turing machines for multiplication, factorization, sorting, etc.

Universal Functions & Universality

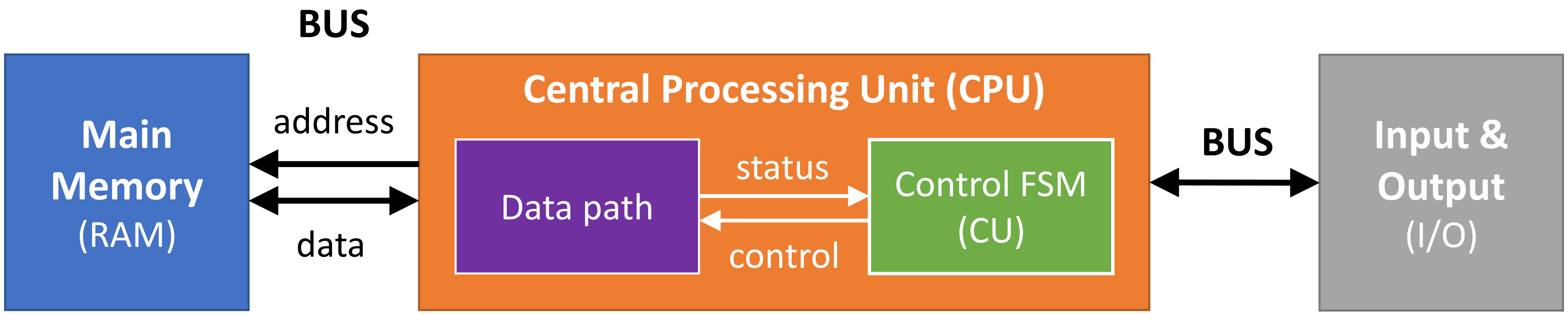


- The universal function: $U(k, j) = T_k(j)$
- U is computable by a Turing machine
- Important idea: <u>Interpretation</u>
  - k encodes a "program" – a description of some arbitrary TM
  - j encodes the input data to be used
  - T_U interprets the program, emulating its processing of the data

  → Manipulate coded representations of computing machines, rather than the machines themselves
- Universal Turing Machine is the paradigm for modern general-purpose computers!
- … now back to reality!

The von Neumann model – A general-purpose computer

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN



E.g. 16 GB DDR3 64 bit RAM

ROM (read-only memory) &
RAM (randon-access memory),
Array of **words** of k **bits**:

- Early computers:
  8 bits = 1 byte
- Then & "beta":
  32 bits = 4 bytes
- Now:
  64 bits = 8 bytes

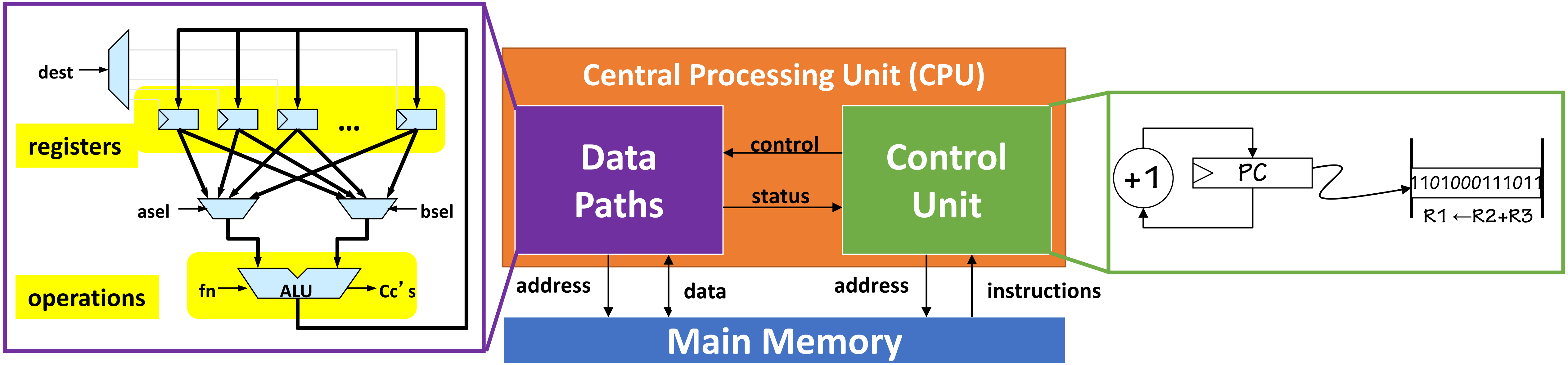
E.g. Intel Core i7 64 bit

- Several **registers** (8, 32, 64, …) in **control unit** (CU – a FSM)
- Data paths (logic) performing specified set of operations (instructions)
  → **Instruction set**
  → **Arithmetic logic unit** (ALU)

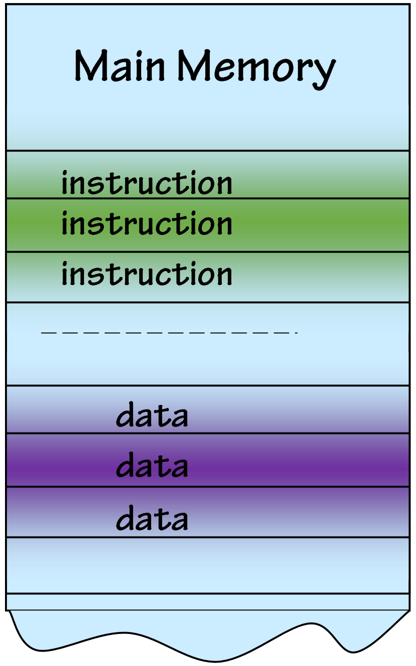
E.g. keyboard, mouse, monitor

Communication with the outside world

Anatomy of von Neumann computer



- **Program Counter (PC)**: Address of next instruction to be executed
- **Instructions** coded as binary data
- Logic to **interpret** (**translate**) **instructions into control signals** for data path
- Logic to **feed data from memory** into data path (registers)
- **ALU** to perform operations on data in registers
- PC advances



Pseudo code of FSM:

Reset PC ← 0

Repeat - CU reads word of instruction from mem[PC]

- Data paths and ALU interpret and execute instruction
- PC ← PC+1

Summary

- Basis for modern computer science:
  - Formal models such as Turing machine
  - Concepts of computability, universality and programmability
  - Algorithms are represented as data that needs to be interpreted
  - Hardware & software (programs, compilers, interpreters)
- Von Neumann model and general-purpose computer:
  - CPU + Memory + I/O
  - CPU: PC, CU, ALU, registers
  - Memory: contains instructions and data
  - Universal and programmable