Laboratory Times

A Friday 9:00-11:00 room 003/4/5 Eolas (144 – Mainly CSSE)

B Friday 11:00-1:00 room 002 Eolas (48 - Other)

CS253 Architectures II

Lecture 1

CPU

Charles Markham

Course Topics

Architecture of a Small Microprocessor based Computer

Assembly language programming

Interrupts and IO

Machine Cycle

Representation of data

Memory

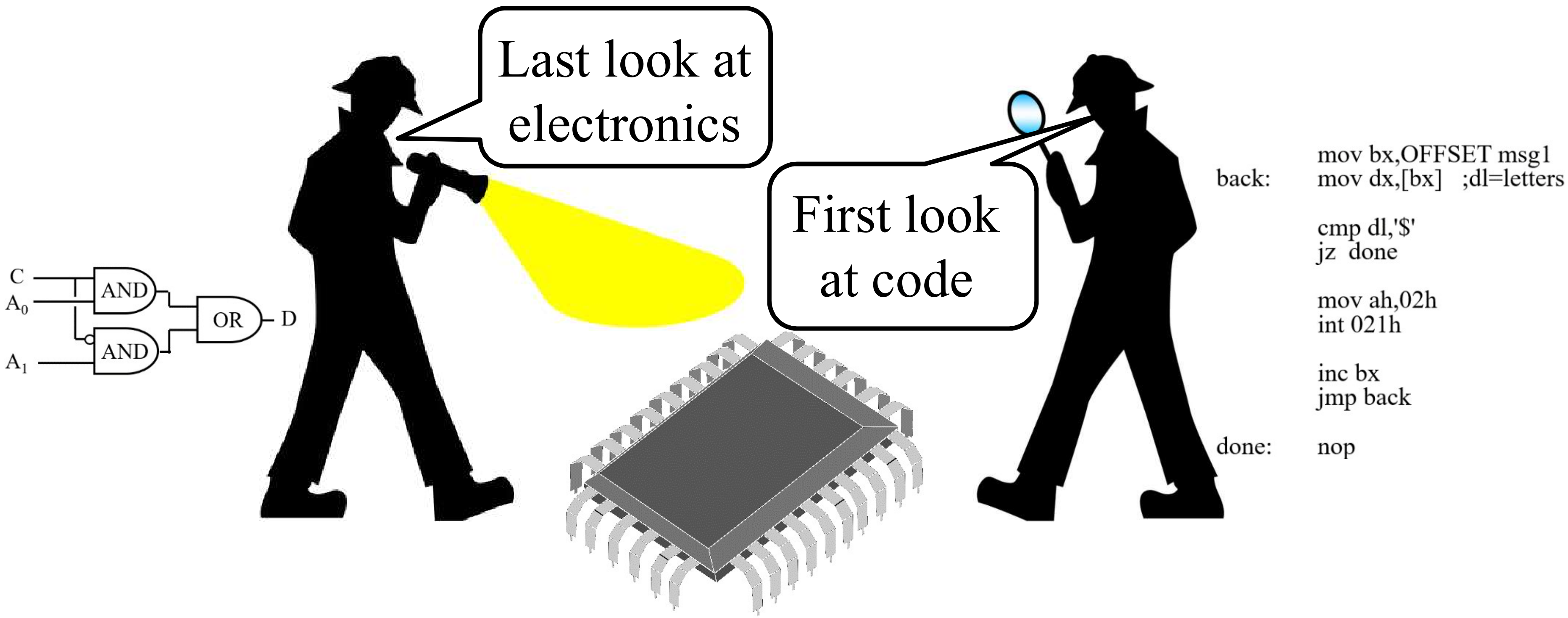
Buses

Modern Processors

CS253 Aims

Last look at the computer as a piece of hardware…

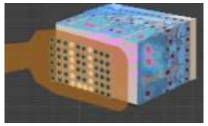
…and a first look at it as a device you can program.



Types of computer

There are many ways that we could choose to compute


IBM TrueNorth

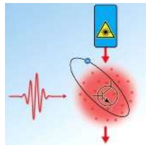


Retinal Prosthesis

Neurones

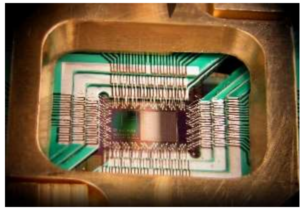
Spatial filtering

Holography



Light

Optical transistor



D-Wave 128 Qubits

n bits can be in only one of 2^n states

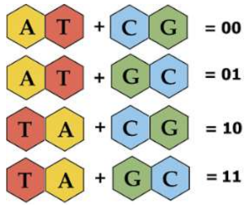
n Qubits contains superposition of 2^n states

Quantum

Difference Engine

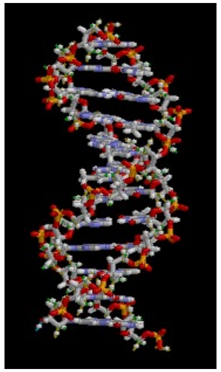
Pin Wheel

Mechanical

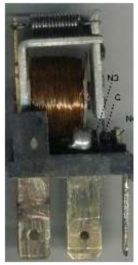


Coding of data

DNA



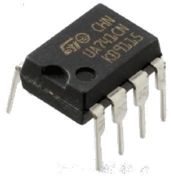
Structure



Relay

Valve

Transistor



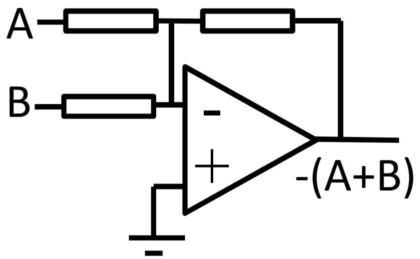
Electrical

IC

The DEC (Digital Electronic Computer) is by far the most popular

Types of computer

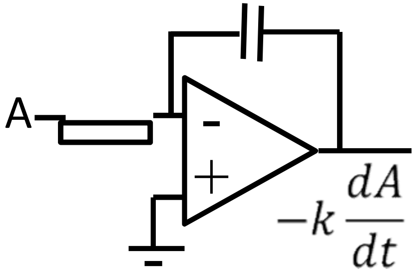
Analogue



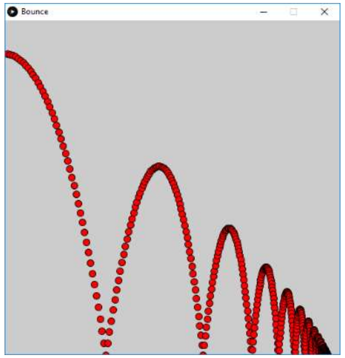
Analog adder circuit

$$-mg + kv + m\frac{d^2y}{dt^2} = 0$$

Differential Equation



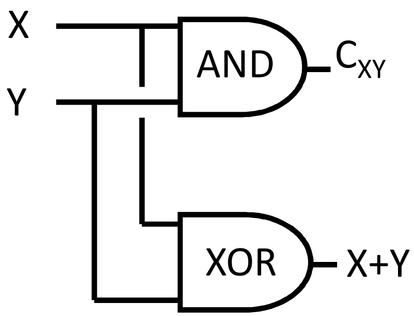
Differentiator



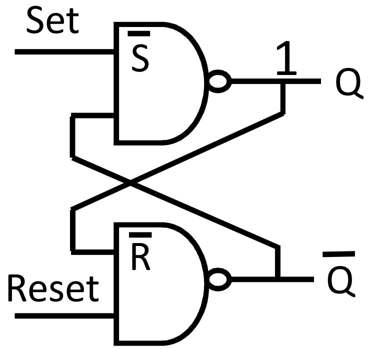
Response

An analogue signal is continuous (can have all values in a range). The signal is processed by electronics that can amplify, multiply, differentiate and integrate input signals to produce an output. Projectile motion can be simulated using a circuit that responds in the same way as the differential equation describing the motion .

Digital

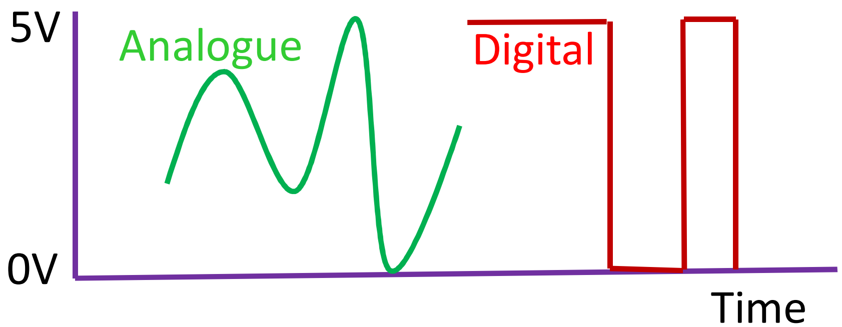


Adder



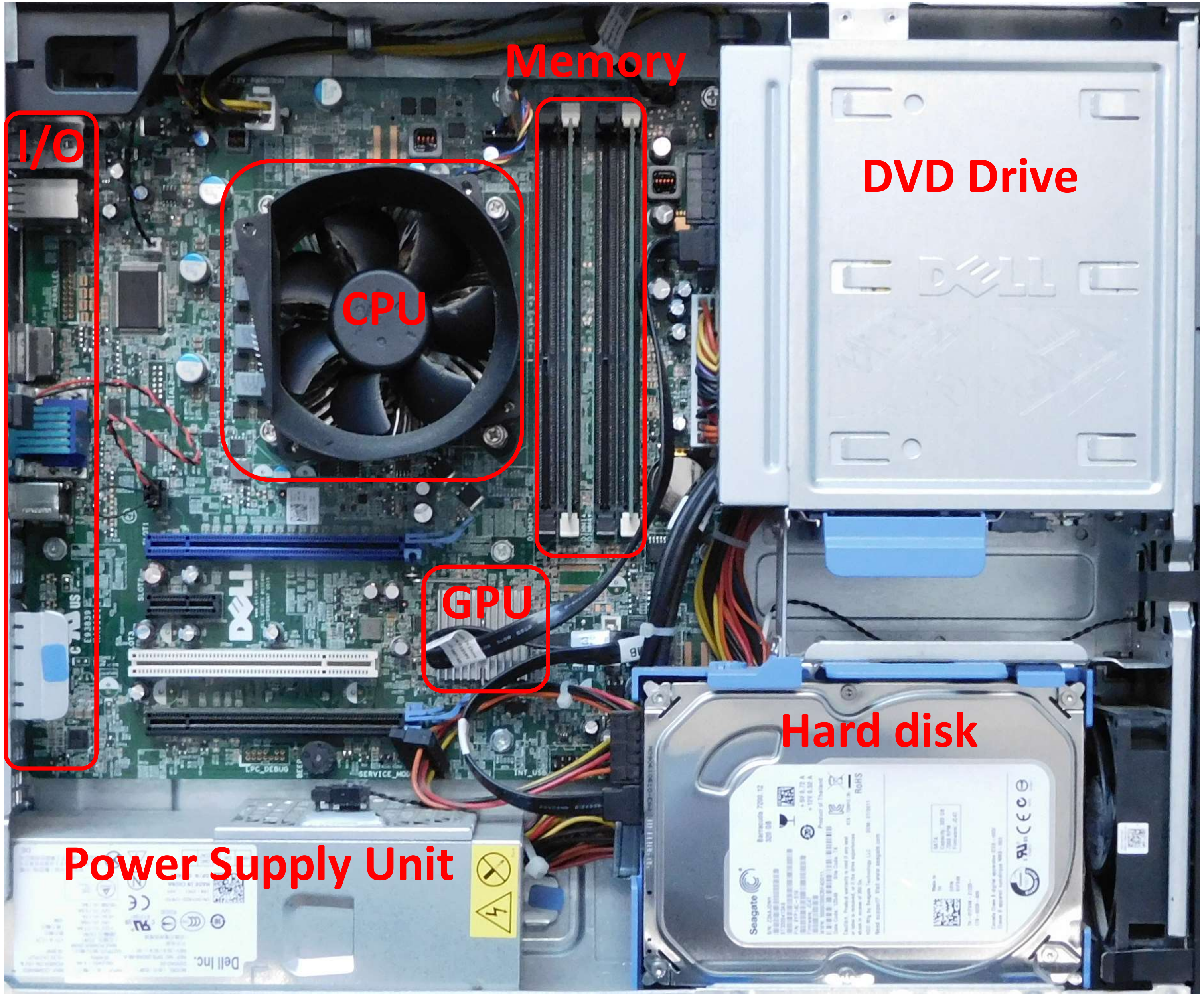
RS Flip flop (1 bit memory)

A digital signal is discrete (can have one of two values in a range). Digital logic can use 0 volts to represent 0 in binary and say 5 volts to represent 1 in binary. Digital circuits to process the 1 or 0 information are much easier to build as they only need to switch between two values rather have an output that is continuous and accurate over a range.



Lift the lid on a PC

PC Mother Board

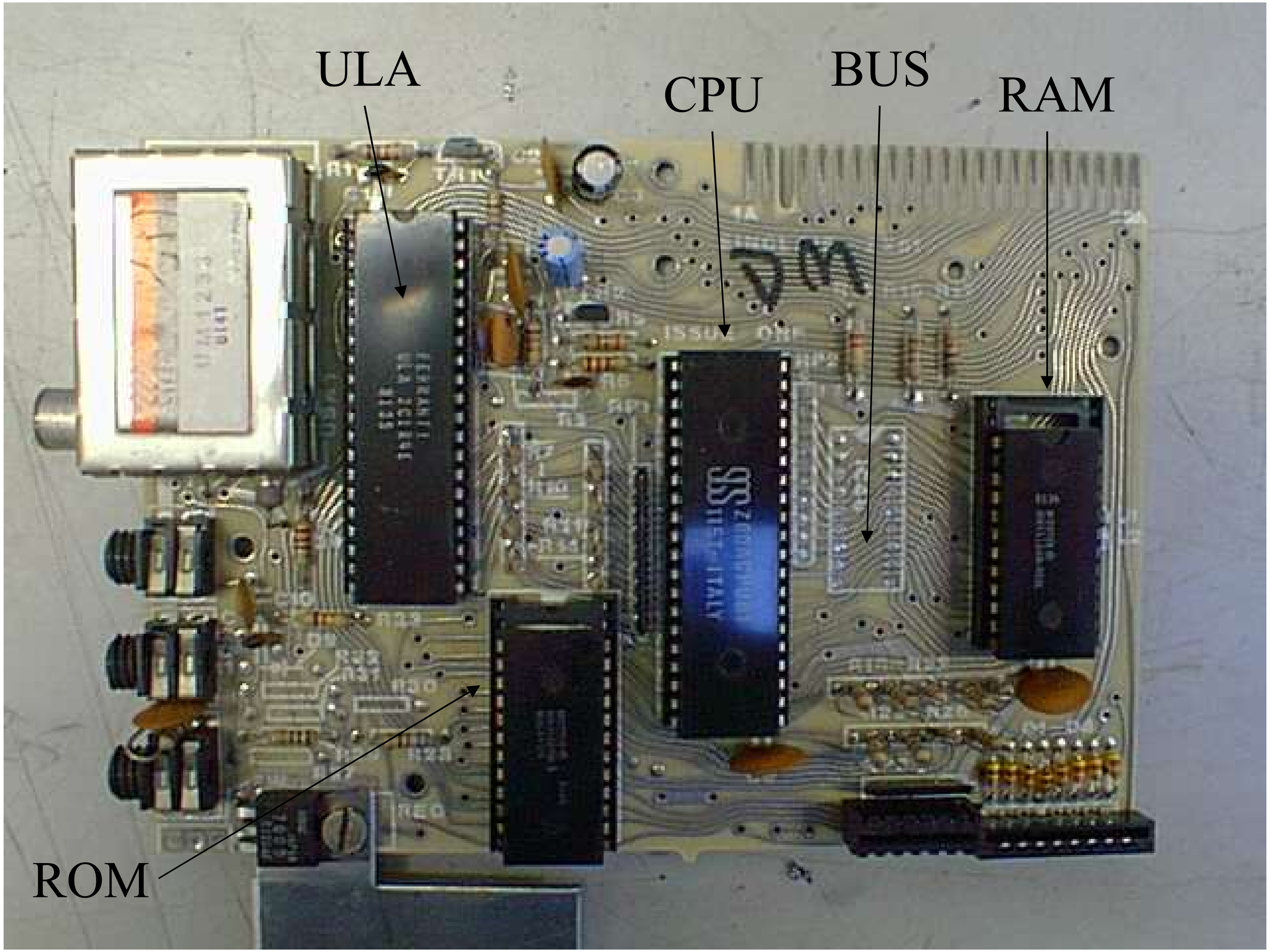


ZX81, Z80 Based Microcomputer

As straightforward as it gets, similar features to a PC.

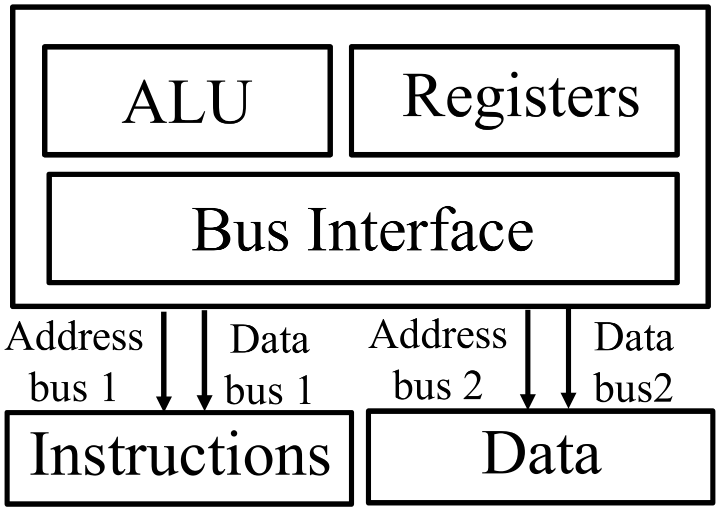
They are all DEC (Digital Electronic Computers)

Z80A Mother Board



Types of computer

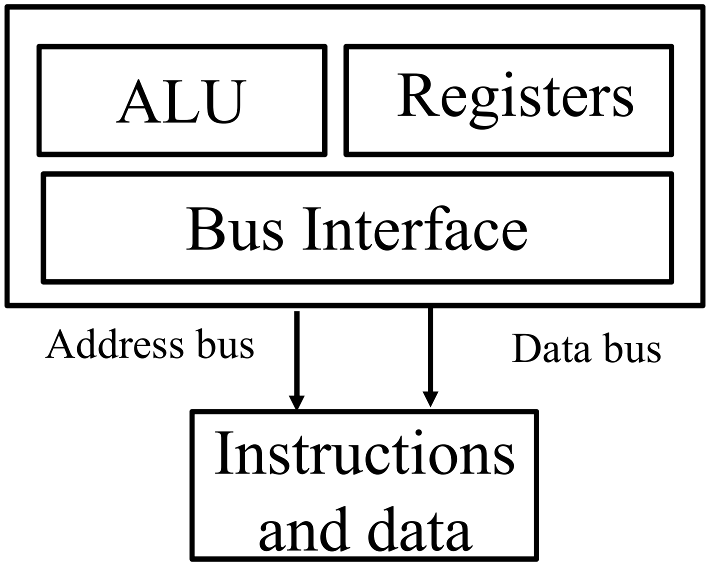
Harvard




```
Arizona PIC Microcontroller
Instruction  Data   Assembly language
5A           01    movlw B'00000001' ; w=1
4F           03    movwf PORTB   ; Port B=w
```

Memory to store instructions (operators) is separate to the memory used for data (operands).

Von Neumann

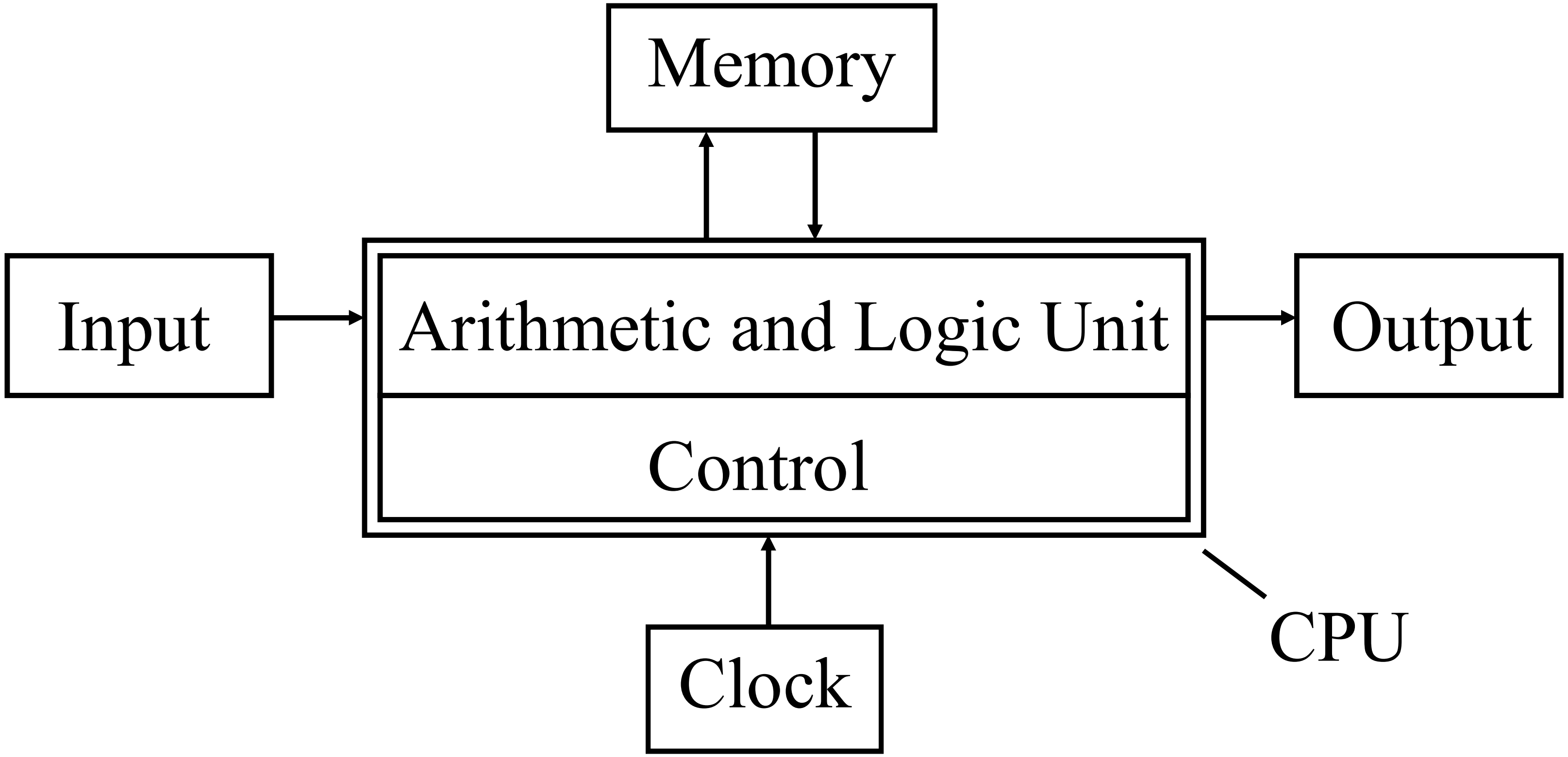



```
x86 Assembly language
Instruction and Data           Assembly language
B0 00                             mov al,0
B8 02 38                          mov ax,568
```

There is only one memory used for both data and instructions. Looking at bytes in memory it would difficult to tell with certainty which stores code and which stores data.

Even though the x86 processor is von Neumann, internally it can split data and instruction pipelines to speed things up (essentially Harvard in nature)

Organisation of a von Neumann digital electronic computer



A Compiled Assembly Language Program

Memory location

Operator

Operand

```
0017  90          Cnt: nop
0018  B0 00            mov al,0
001A  B8 0238          mov ax,568
001D  8A D6            mov dl,dh
001F  8B C3            mov ax,bx
0021  B8 0017 R        mov ax,Cnt
0024  8B 07            mov ax,[bx]
0026  8B 00            mov ax,[bx][si]
```

Machine Code

Assembly Language

Amateur radio – my introduction assembly language

Nr.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
Buchstabenreihe	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	<	≡	Bu	Zi	ZWR	
Zeichenreihe	-	?	:	✠	3				8	🔔	(	)	.	,	9	0	1	4	'	5	7	=	2	/	6	+						
Anlaufschritt																																
5er Schrittgruppe 1	●	●		●	●	●				●	●						●		●		●		●	●	●	●			●	●		
2	●		●				●		●	●	●	●				●	●	●			●	●	●					●	●	●		
3			●			●		●	●		●		●	●		●	●		●		●	●		●	●				●		●	
4		●	●	●		●	●			●	●		●	●	●			●				●		●			●		●	●		
5		●					●	●				●	●		●	●	●			●		●	●	●	●	●			●	●		
Sperrschritt 1½ fach	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●

□ Pausenschritt

● Stromschritt

< Wagenrücklauf

≡ Zeilenvorschub

✠ Wer da?

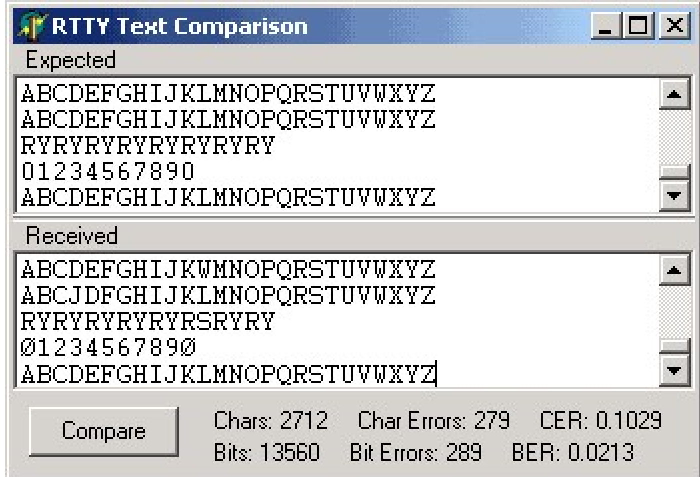
Frei für den internen Betrieb eines jeden Landes, aber im zwischenstaatl. Verkehr nicht zugelassen.

🔔 Klingel

Zi Ziffernumschaltung

Bu Buchstabenumschaltung

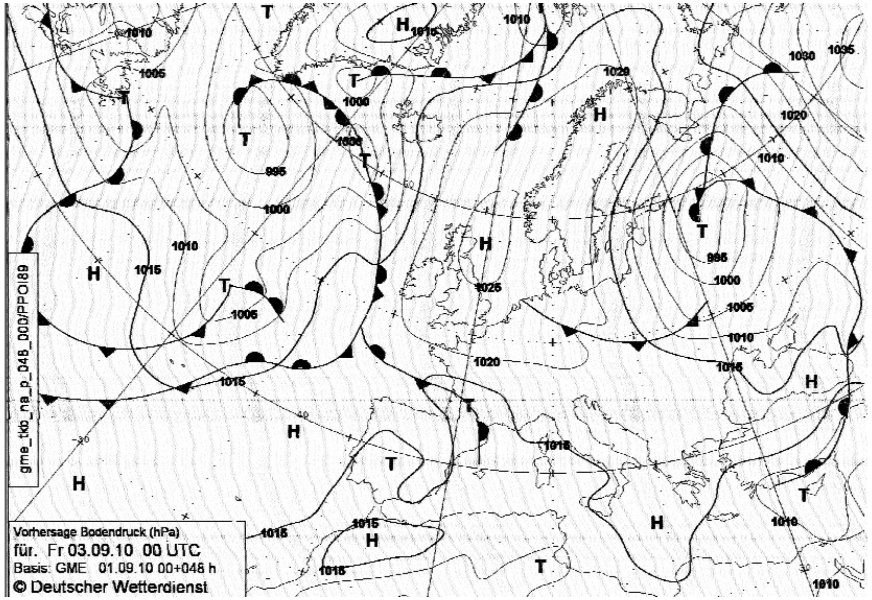
ZWR Zwischenraum

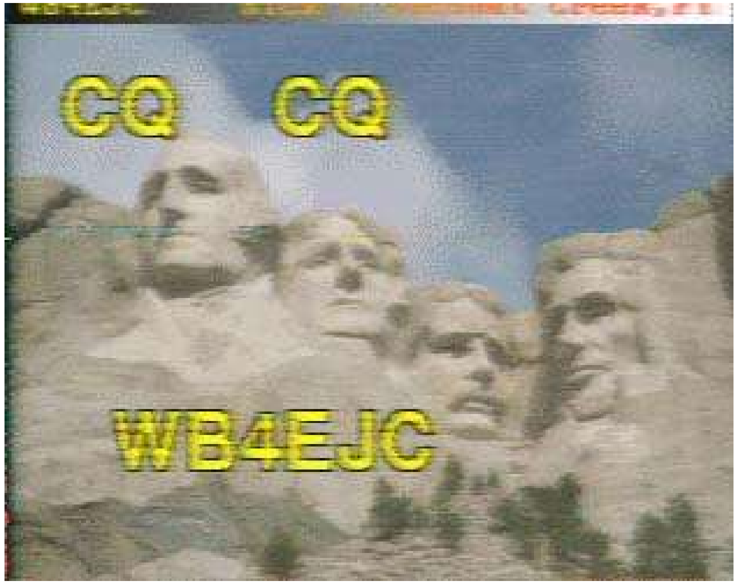


RTTY Radio-teletype

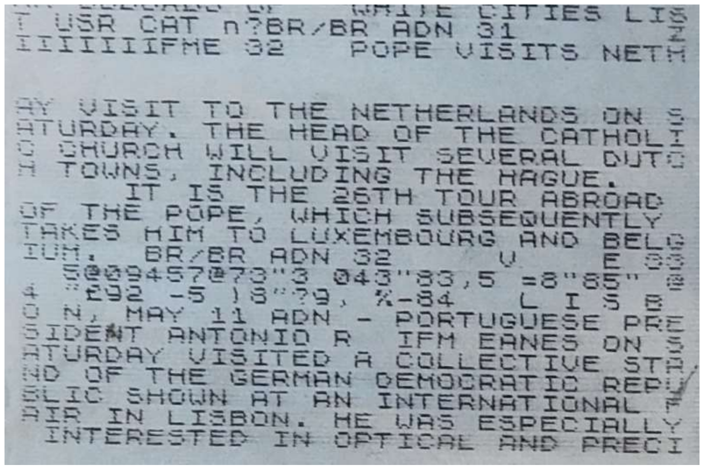
Amateur radio

Shortwave





Wefax – Weather fax

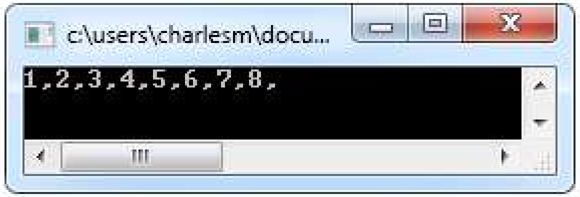


RTTY– Radio Teletype

SSTV– Slow scan TV

A more recent use of assembly language – access to mmx instructions

```
union mmx_word{
              unsigned char byte[8];
              unsigned __int64 value;
            };

mmx_word NUM1={0,1,2,3,4,5,6,7};
mmx_word NUM2={1,1,1,1,1,1,1,1};

 __asm
 {
   movq  mm0,NUM1
   movq  mm1,NUM2

   paddb mm0,mm1  // Add 8 bytes simultaneously

   movq  NUM1,mm0
 }

 for(int i=0;i<8;i++) printf("%d,",(unsigned int)NUM1.byte[i]);
```



Output

A more recent use of assembly language – reverse engineering

```
#include "stdafx.h"
using namespace System;
int main(array<System::String ^> ^args)
{
  unsigned int x=123;
  unsigned int y=456;
  unsigned int z;
  z=x+y;
}
```

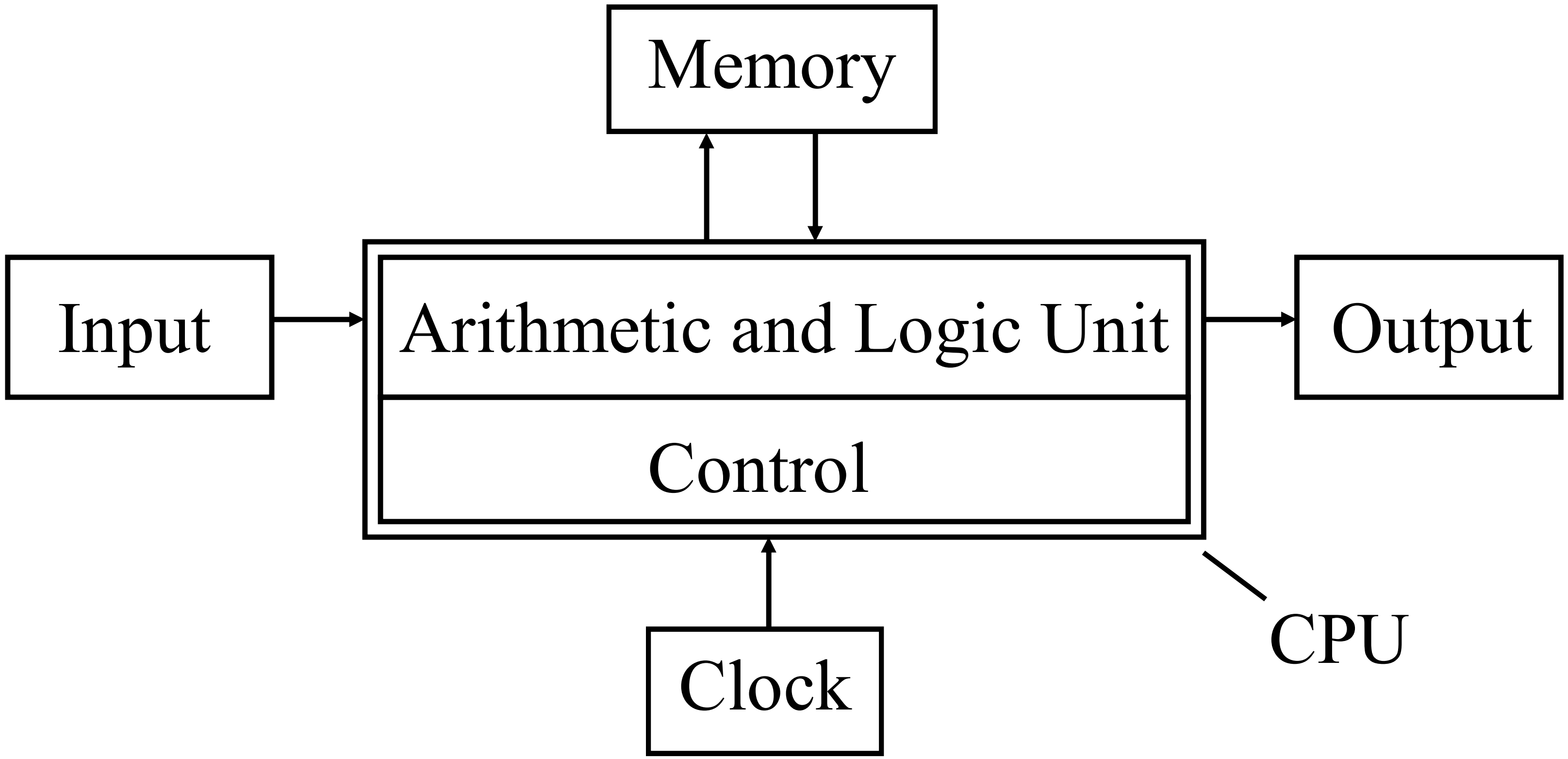
Compile and run

```
; 9    :         unsigned int x=123;

  0000b 16                ldc.i.0 0       ; i32 0x0
  0000c 0a                stloc.0         ; $T8928
  0000d 1f 7b             ldc.i4.s 123    ; u32 0x7b
  0000f 0c                stloc.2         ; _x$

; 10   :         unsigned int y=456;

  00010 20 c8 01 00 00    ldc.i4 456      ; u32 0x1c8
  00015 0b                stloc.1         ; _y$

; 11   :         unsigned int z;
; 12   :         z=x*y;

  00016 08                ldloc.2         ; _x$
  00017 07                ldloc.1         ; _y$
  00018 5a                mul
  00019 0d                stloc.3         ; _z$
```

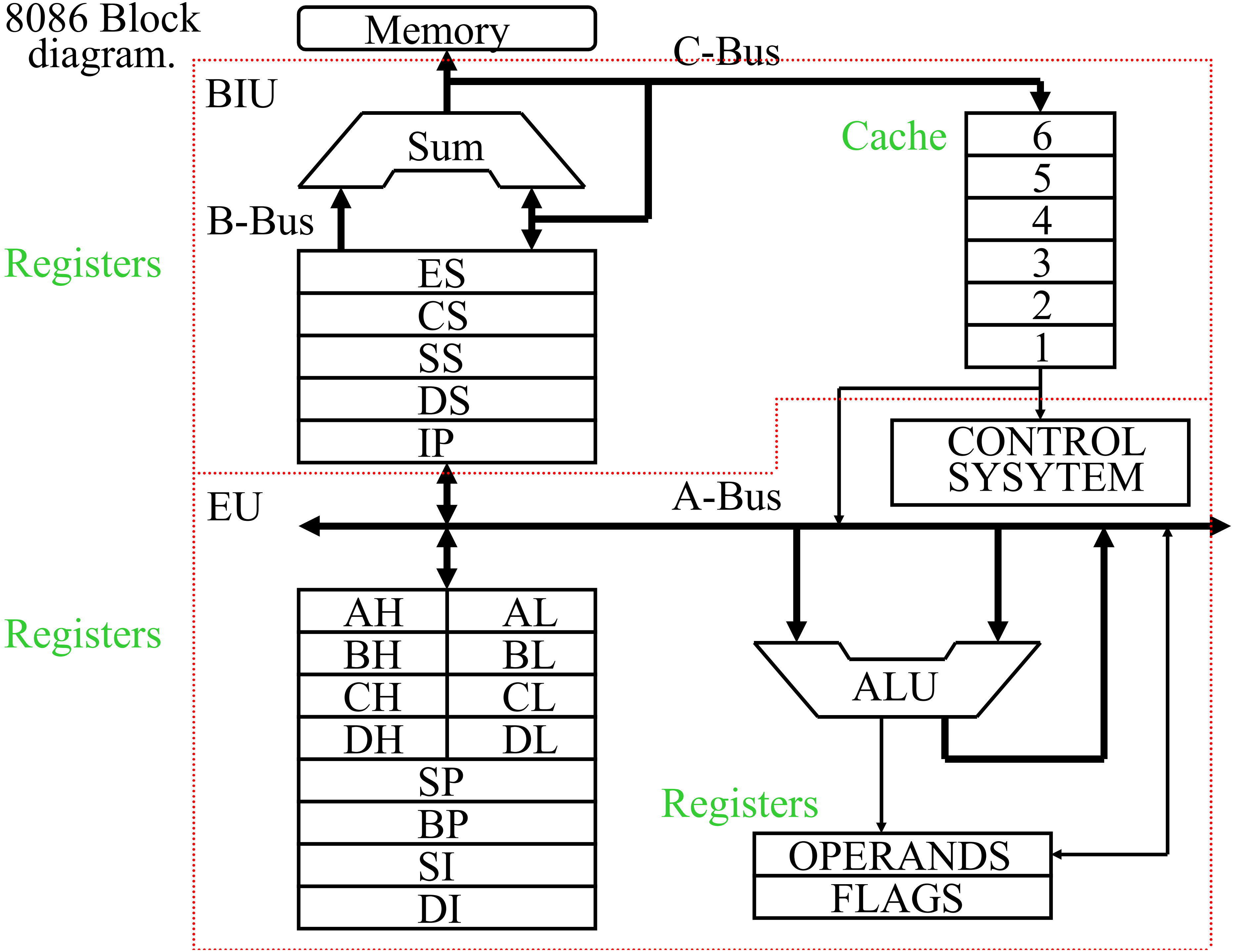
Note: This is not strictly assembly language it is byte code running on a virtual machine called the common library runtime CLR.

https://en.wikipedia.org/wiki/Assembly_language

Organisation of a digital computer



8086 Block diagram.



The CPU has two main function blocks

BIU: Bus Interface Unit

The BIU sends outs addresses fetches instructions from memory and reads and writes to ports and to memory.

EU: Execution Unit

The EU instructs the BIU where to fetch instructions, it decodes instructions and executes instructions.

The EU contains both the ALU and Control Circuitry.

The A-Bus, B-Bus and C-Bus are high speed data paths contained within the Microprocessor itself.

Control Circuits (EU)

The control unit fetches instructions from the queue. The queue is a first in first out store of 6 bytes.

The store is kept full by the BIU. This means that main memory is not accessed for each byte of each instruction. The technique is known as *pipleling*.

Some instructions such as conditional jumps and call to subroutines can not be pipelined (more later).

The ALU (EU)

The ALU in the 8086 can ADD, Subtract, AND, OR, XOR, increment, decrement, complement and shift 16-bit binary numbers.

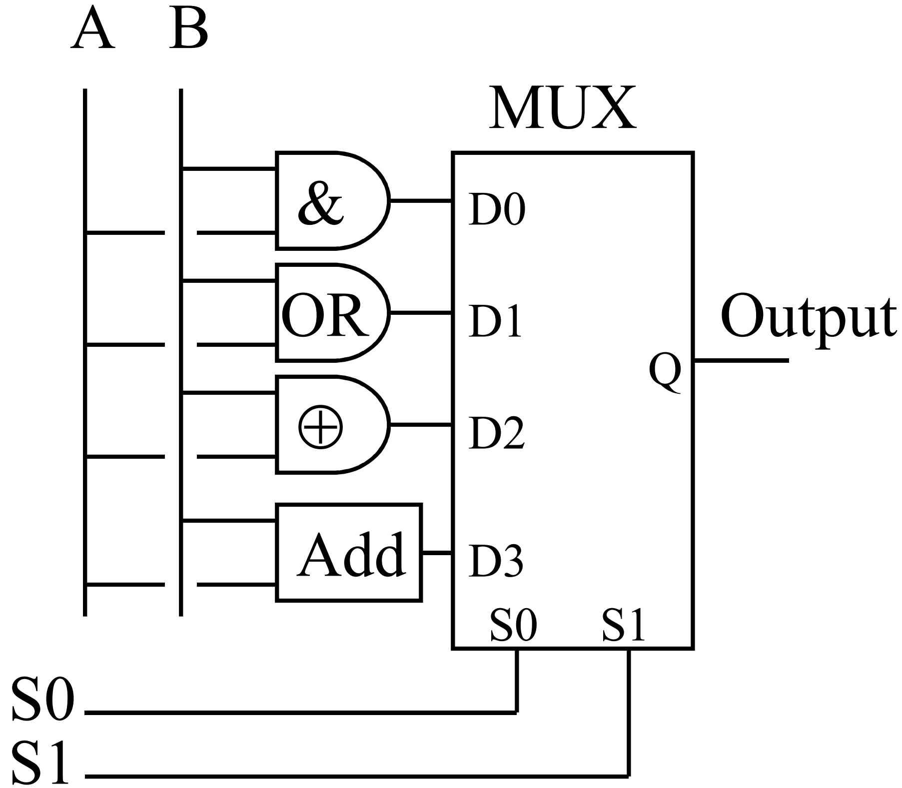
As an exercise in understanding the ALU we will attempt to construct a 4-bit ALU using the Xilinx kit later in the year.

Add:	Ouptut=A+B	Pass:	Output=A
Subtract:	Output=A-B	Complement:	Output=$\overline{A}$
Exor:	Output=A⊕B	Set:	Output=1
AND:	Output=A.B	Shift Left	Output=A*2
OR:	Output=A+B	Shift Right	Output=A/2

ALU

Inputs A and B are operated on by all the functions available. The multiplexer connects the ALU output to the desired function.

S0	S1	Output
0	0	A&B
0	1	A OR B
1	0	A⊕B
1	1	A+B



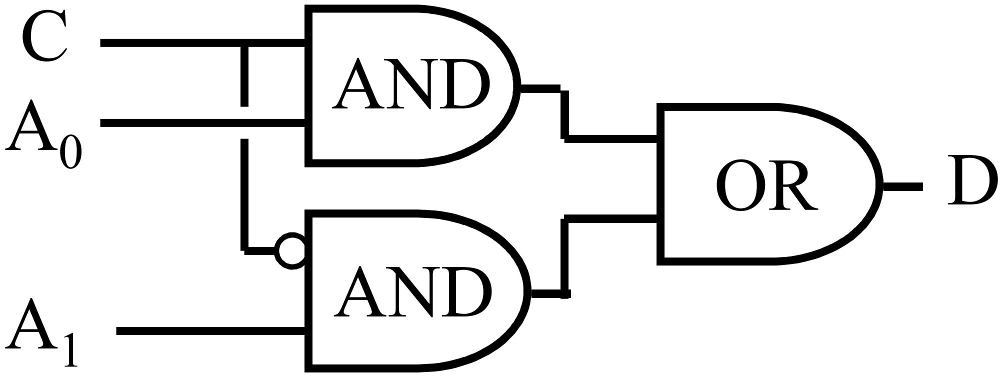
MUX: Multiplexer

ALU

Bit operations such as AND, OR, Complement, EXOR do not require knowledge of the state of other bits in the input words.

Bit operations such as Shift Left, Shift Right, ADD and Subtract need to know about the state of other bits. This increases the amount of wiring necessary but the result can still be achieved with a separate multiplexer for each bit.

Multiplexers



AND

A	B	A.B
0	0	0
0	1	0
1	0	0
1	1	1

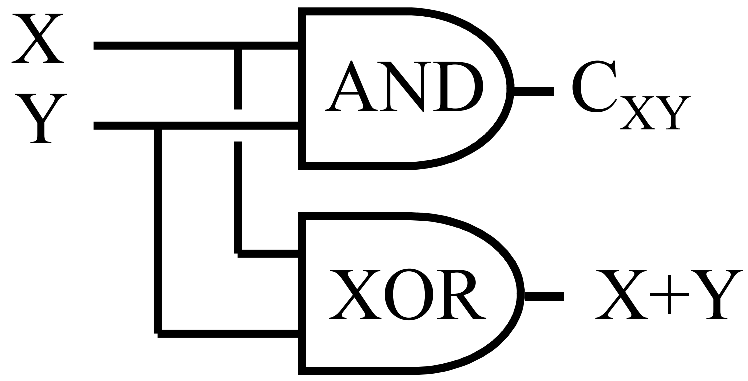
OR

A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

C	D
0	A_1
1	A_0

The Control line C selects which input is routed to the output. How would you design a 4 input multiplexer?

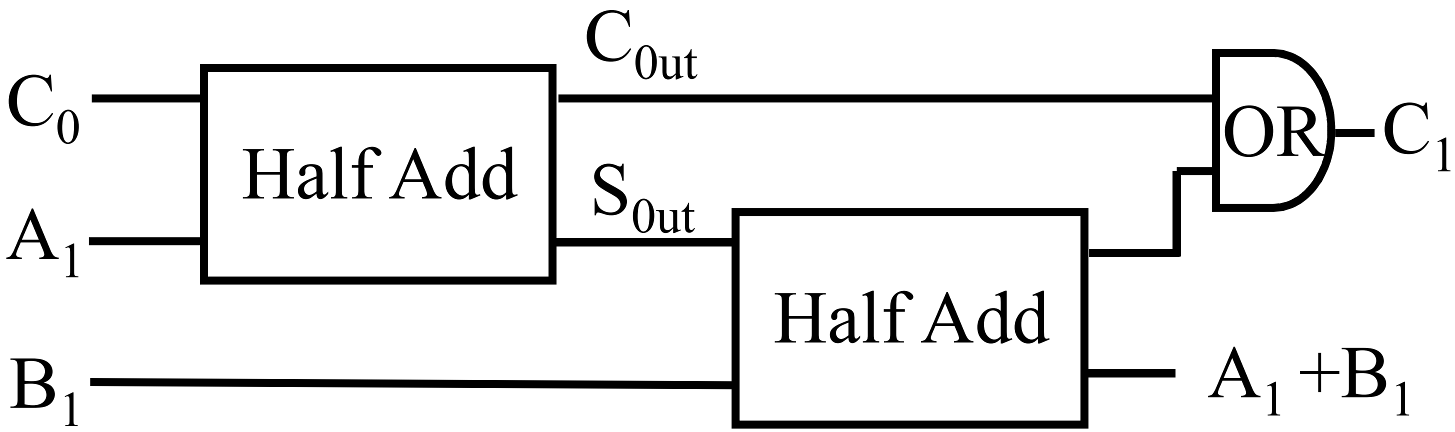
The Adder



Half Adder

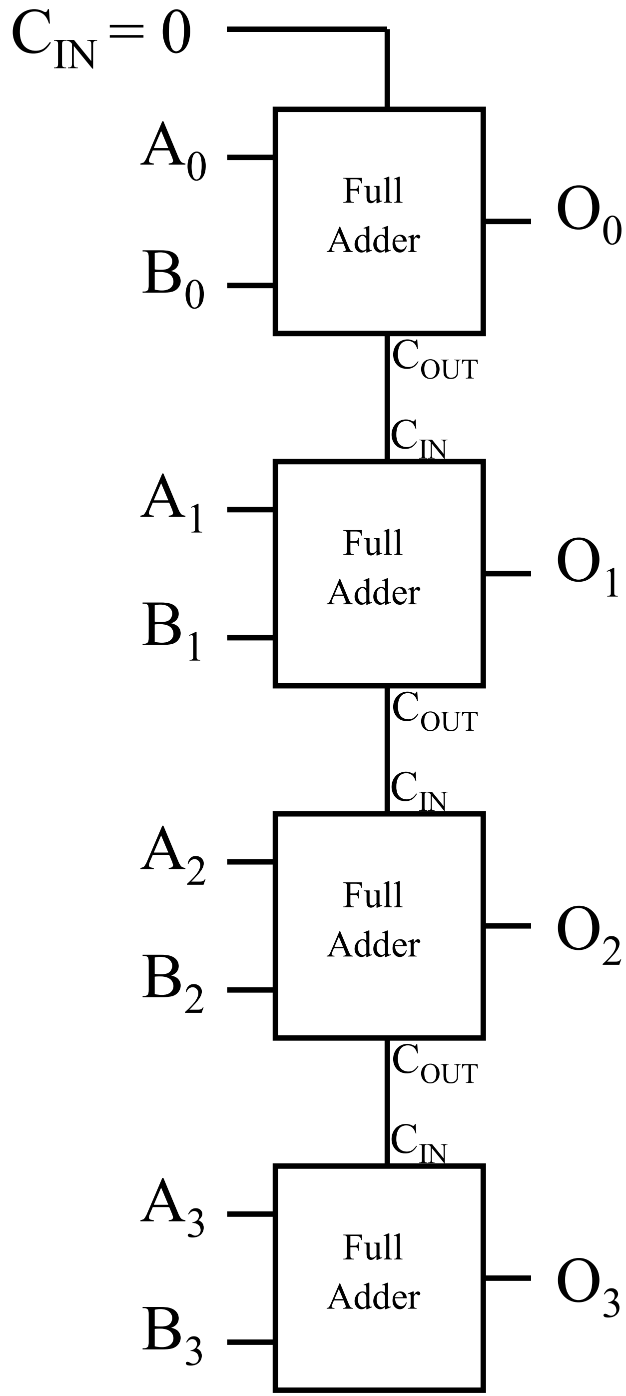
Note: The half adder can not take carry in bits.

Combining two half adders creates a full adder capable of accepting carry in.

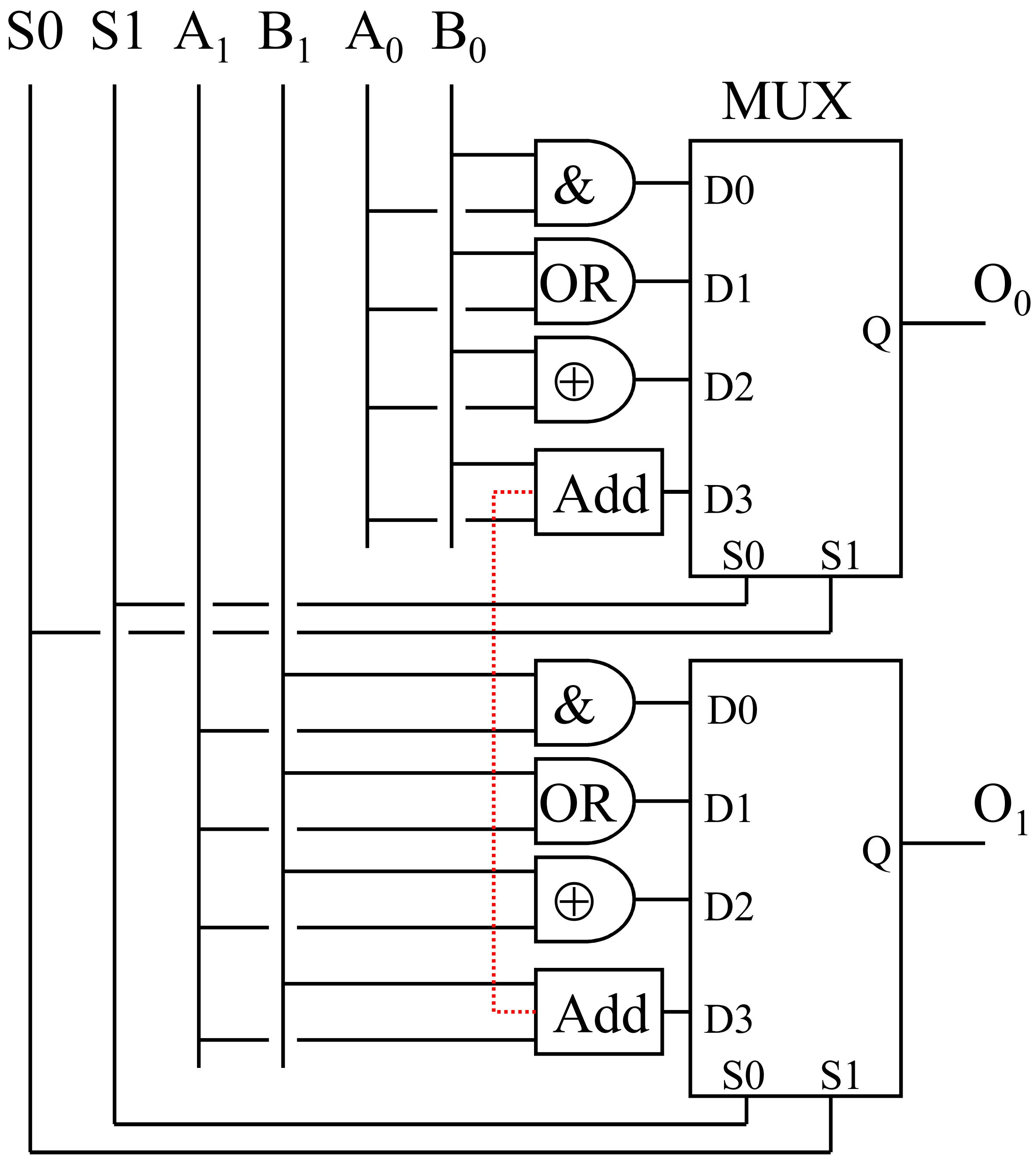


Full Adder

4 Bit Binary Adder



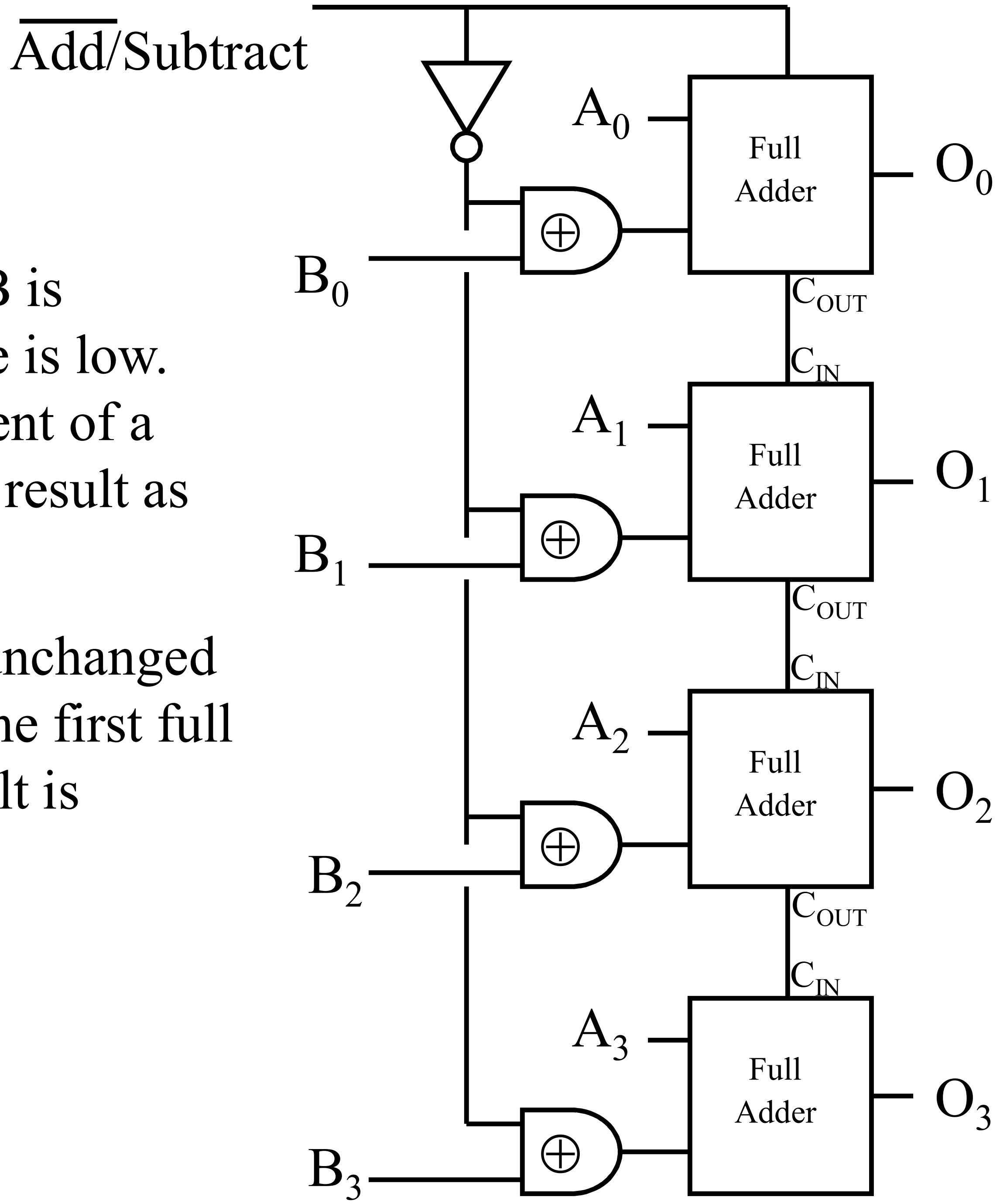
Two bit ALU



S0 S1 A1 B1 A0 B0
MUX
&
OR
⊕
Add
D0
D1
D2
D3
Q
S0 S1
O0
&
OR
⊕
Add
D0
D1
D2
D3
Q
S0 S1
O1
C1

4 Bit Binary

Adder/Subtract

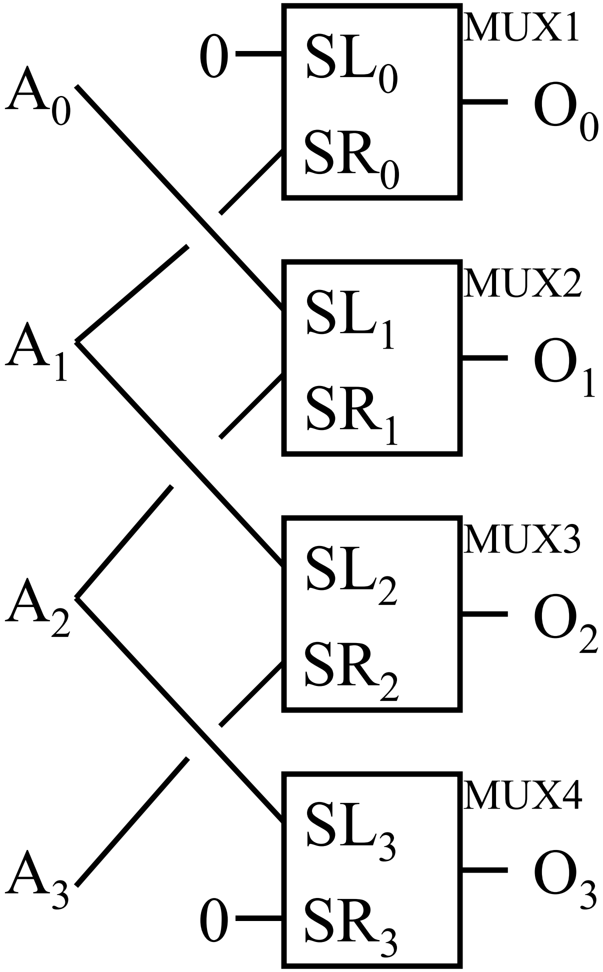


Twos Complement of B is generated when AS line is low. Adding twos complement of a number gives the same result as subtracting the number.

When AS is high B is unchanged and the carry in bit to the first full adder is zero. The result is normal addition.

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

Shift Left or Right



Select lines on MUX not shown.

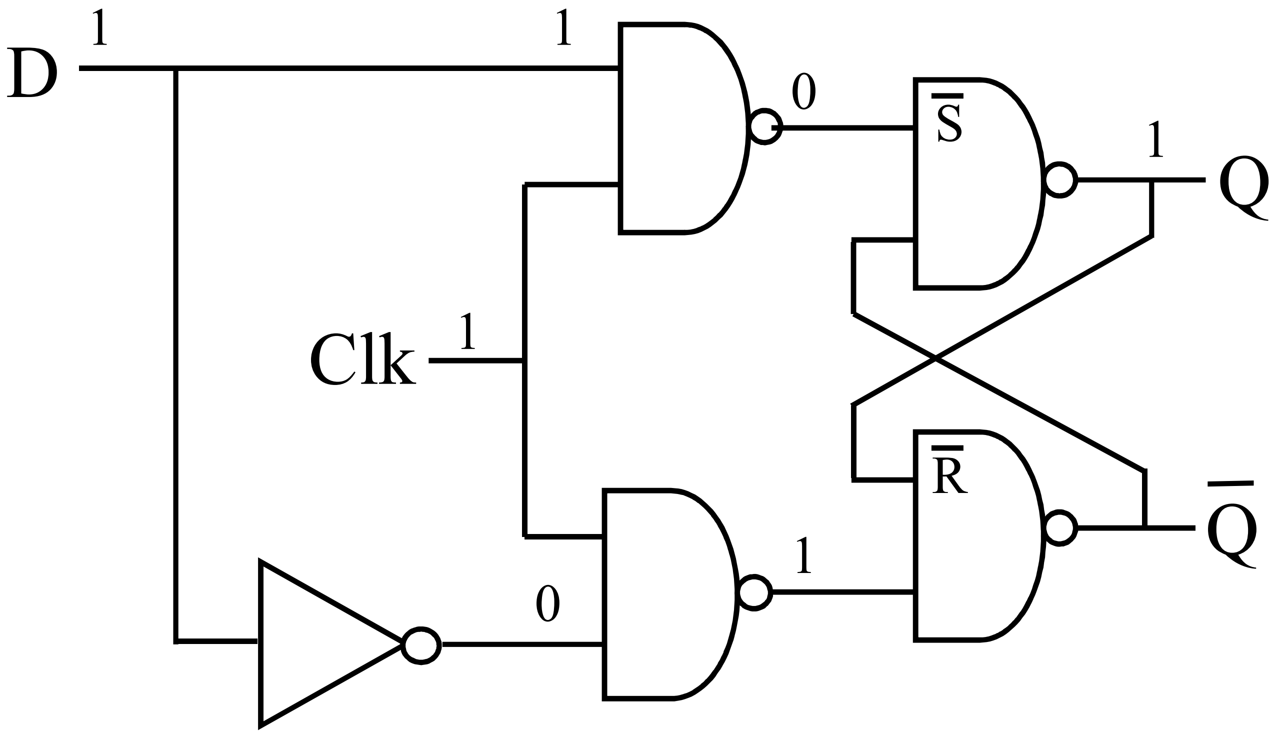
The shift right function of the ALU can be created by wiring A_1 input to the multiplexer with the O_0 output, the A_2 input to a multiplexer with the O_1 output etc.

Shift right is the same as divide by two.

A similar strategy can be used for shift left.

Note: The shift does not require clocked JK flip flops.

D Type Flip Flop

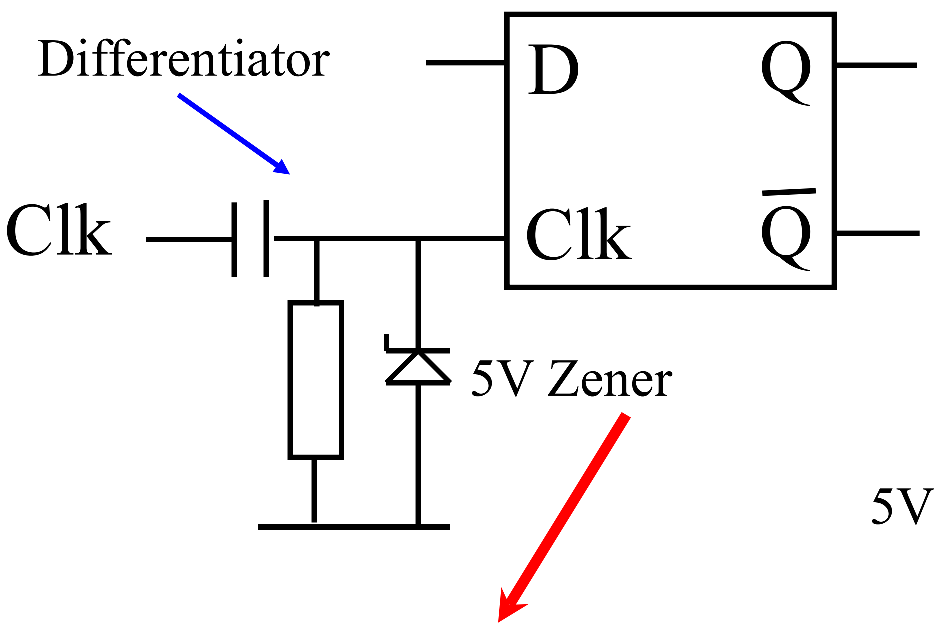


Data (D)=1, Clock=1, Output (Q)=1

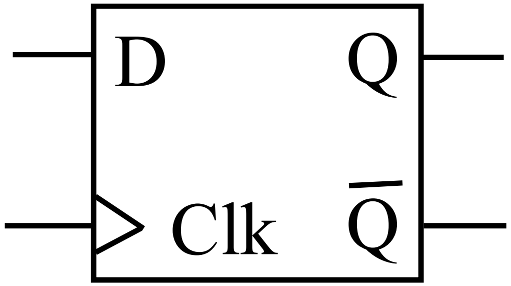
Q=D when Clk=1

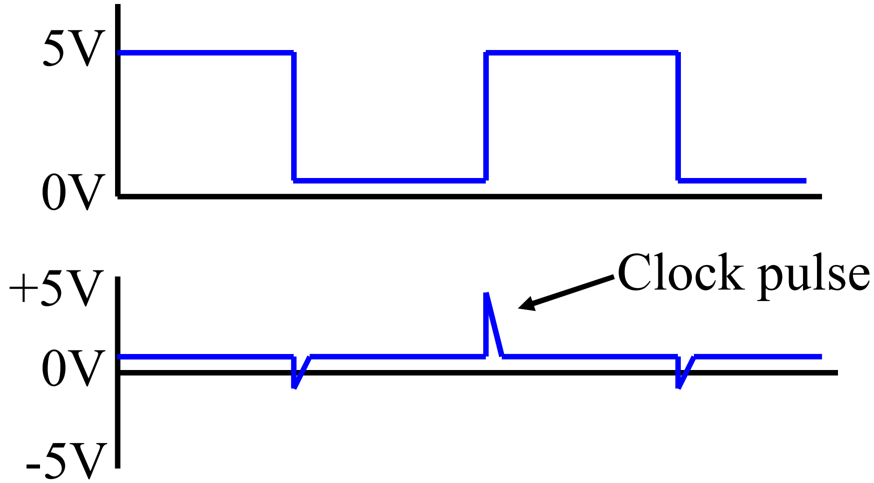
Q=last D when Clk=0

Edge Triggered Flip Flop



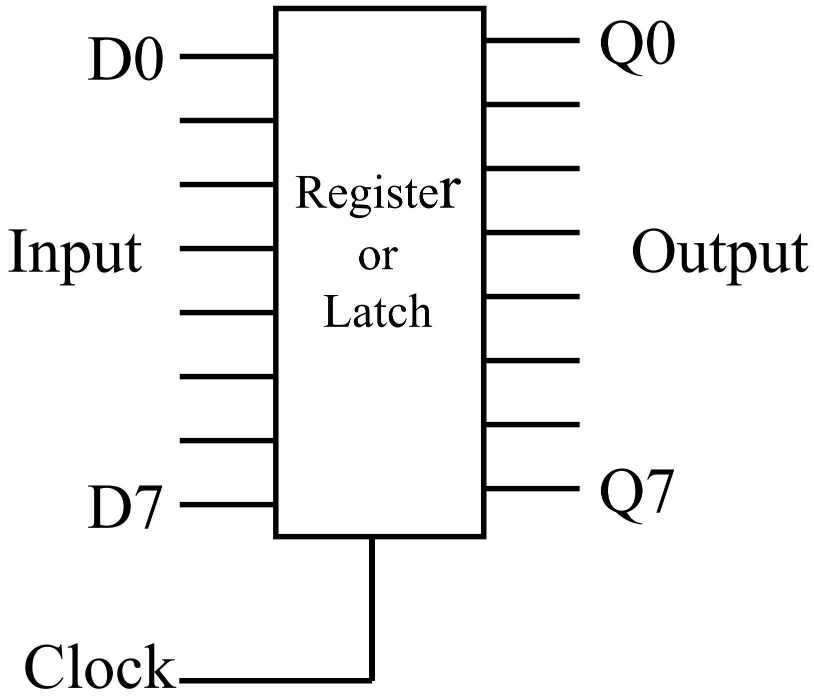
Clk	D	Q
0	x	No change
1	x	No change
↑	0	0
↑	1	1
↓	x	No change





Latch/Register

The CPU contains a number of memory locations made of D type latches. Each memory location contains 8 or 16 bits (typically) and is known as a register. Each register is designed for a specific purpose.

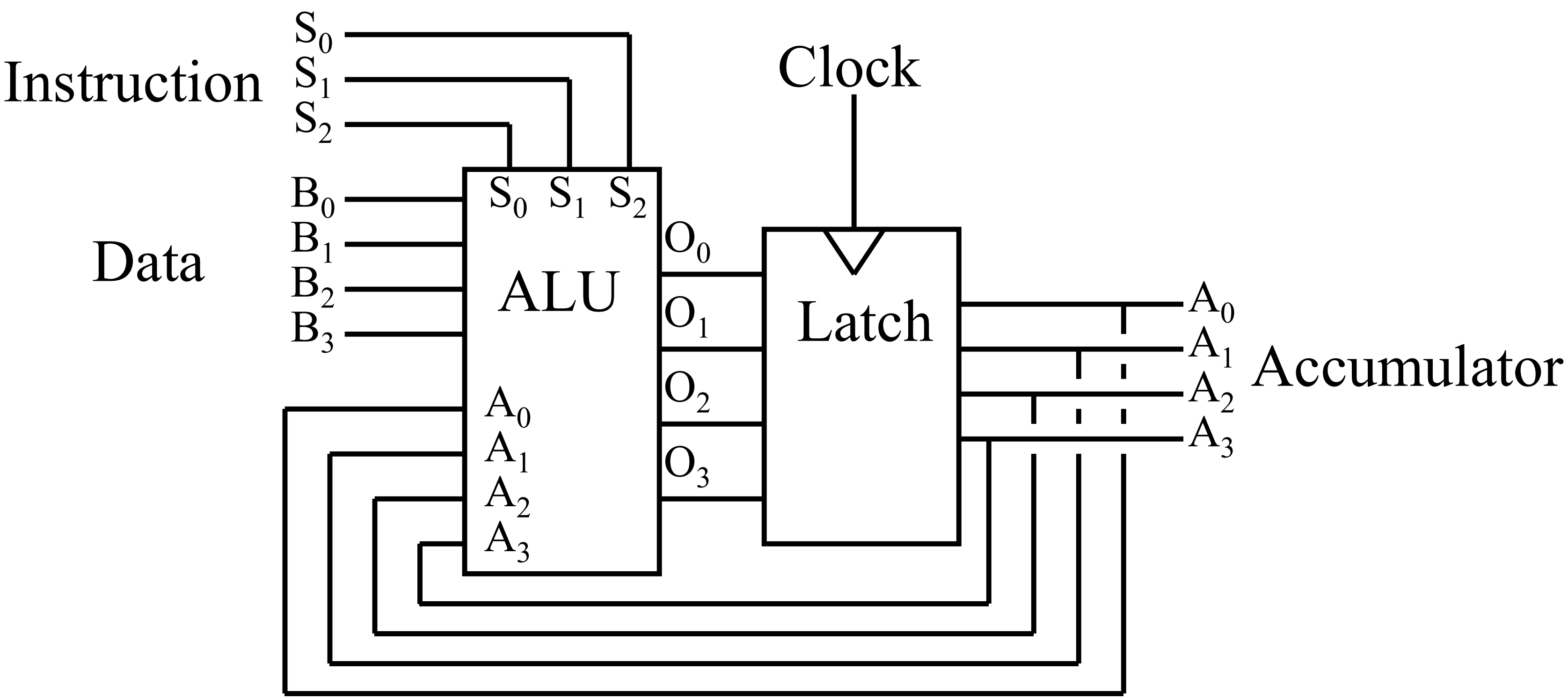


Information (a binary number) is transferred (stored) to the output on next rising edge of the clock.

A Simple Calculator

Most Microprocessors have a special register that stores the result of the calculations executed by the ALU, this register is known as the accumulator. The result of a calculation can also be sent to the stack, other registers or even machine memory.

Using the contents of the accumulator as the input to the ALU creates a device that can do complex calculations.

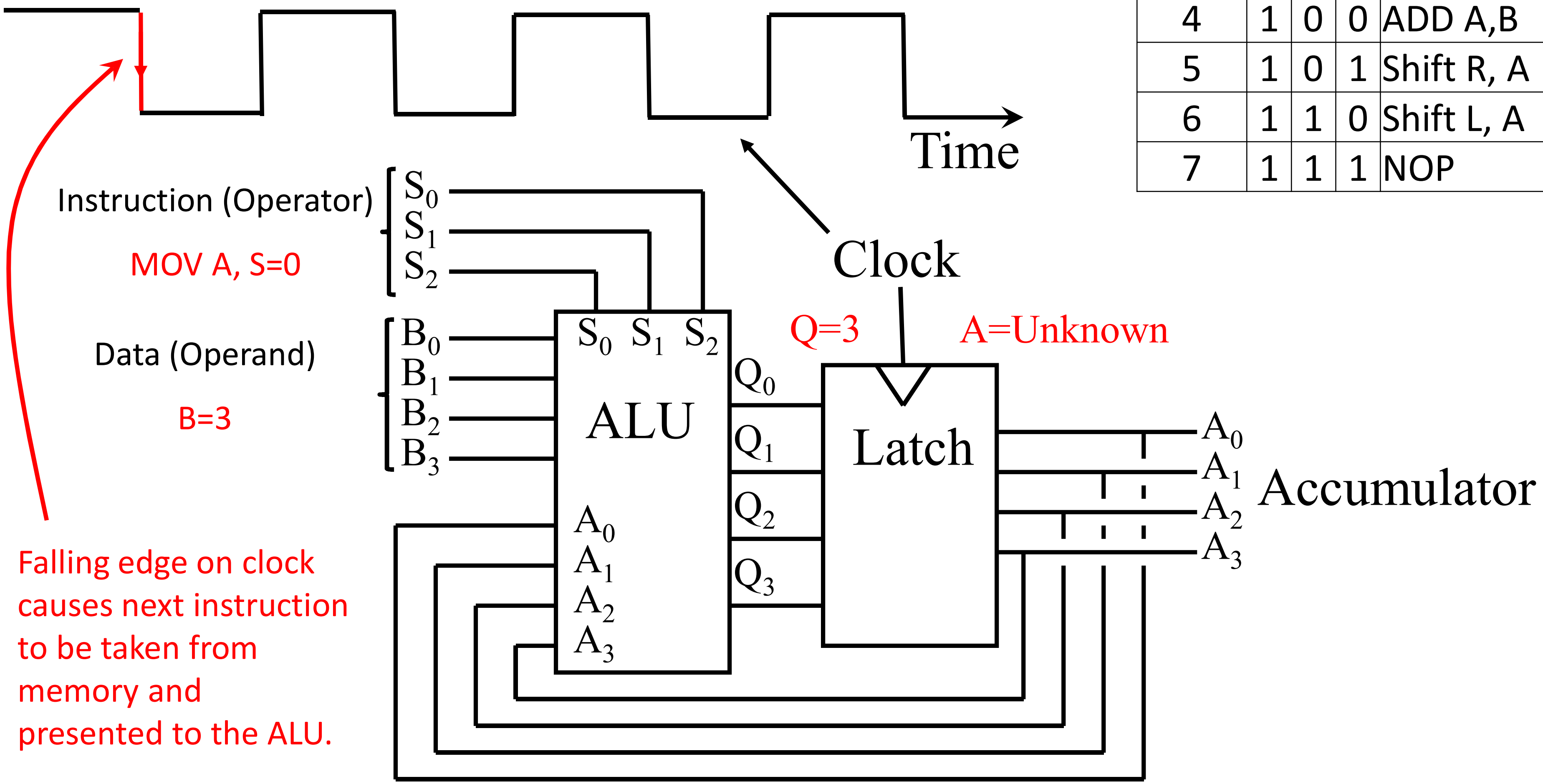


Program to evaluate 5-3

Assembly	Machine code	Accumulator
MOV A, 3	S=0, B=3	A=0011b=3
XOR A,15	S=3, B=15	A=1100b=12
ADD A,1	S=4, B=1	A=1101b=13=-3 TC
ADD A,5	S=4, B=5	A=0010b=2

ALU Instruction set

S(Hex)	S2	S1	S0	Function
0	0	0	0	MOV A,N
1	0	0	1	AND A,B
2	0	1	0	OR A,B
3	0	1	1	XOR A,B
4	1	0	0	ADD A,B
5	1	0	1	Shift R, A
6	1	1	0	Shift L, A
7	1	1	1	NOP

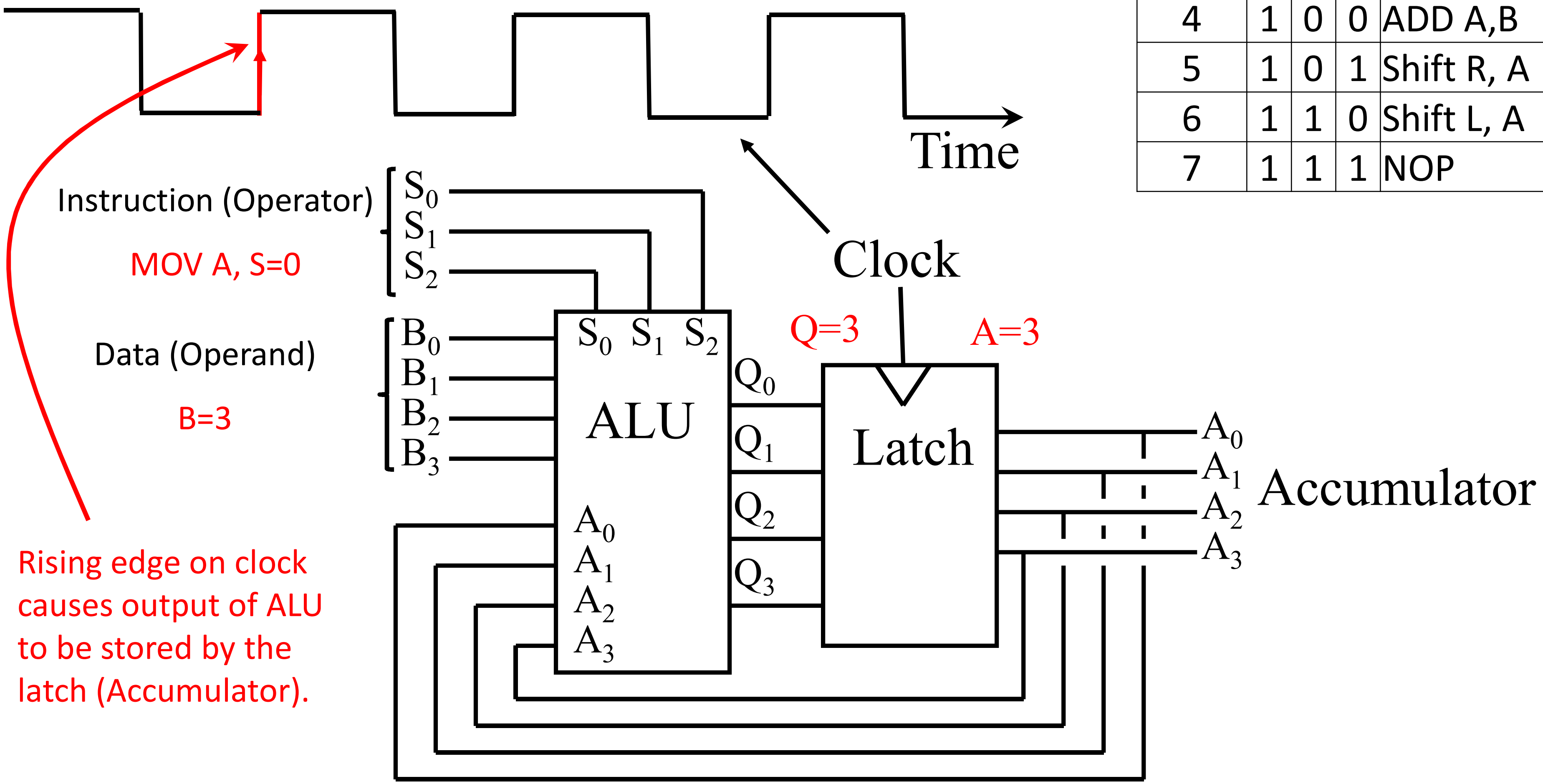


Program to evaluate 5-3

Assembly	Machine code	Accumulator
MOV A, 3	S=0, B=3	A=0011b=3
XOR A,15	S=3, B=15	A=1100b=12
ADD A,1	S=4, B=1	A=1101b=13=-3 TC
ADD A,5	S=4, B=5	A=0010b=2

ALU Instruction set

S(Hex)	S2	S1	S0	Function
0	0	0	0	MOV A,N
1	0	0	1	AND A,B
2	0	1	0	OR A,B
3	0	1	1	XOR A,B
4	1	0	0	ADD A,B
5	1	0	1	Shift R, A
6	1	1	0	Shift L, A
7	1	1	1	NOP

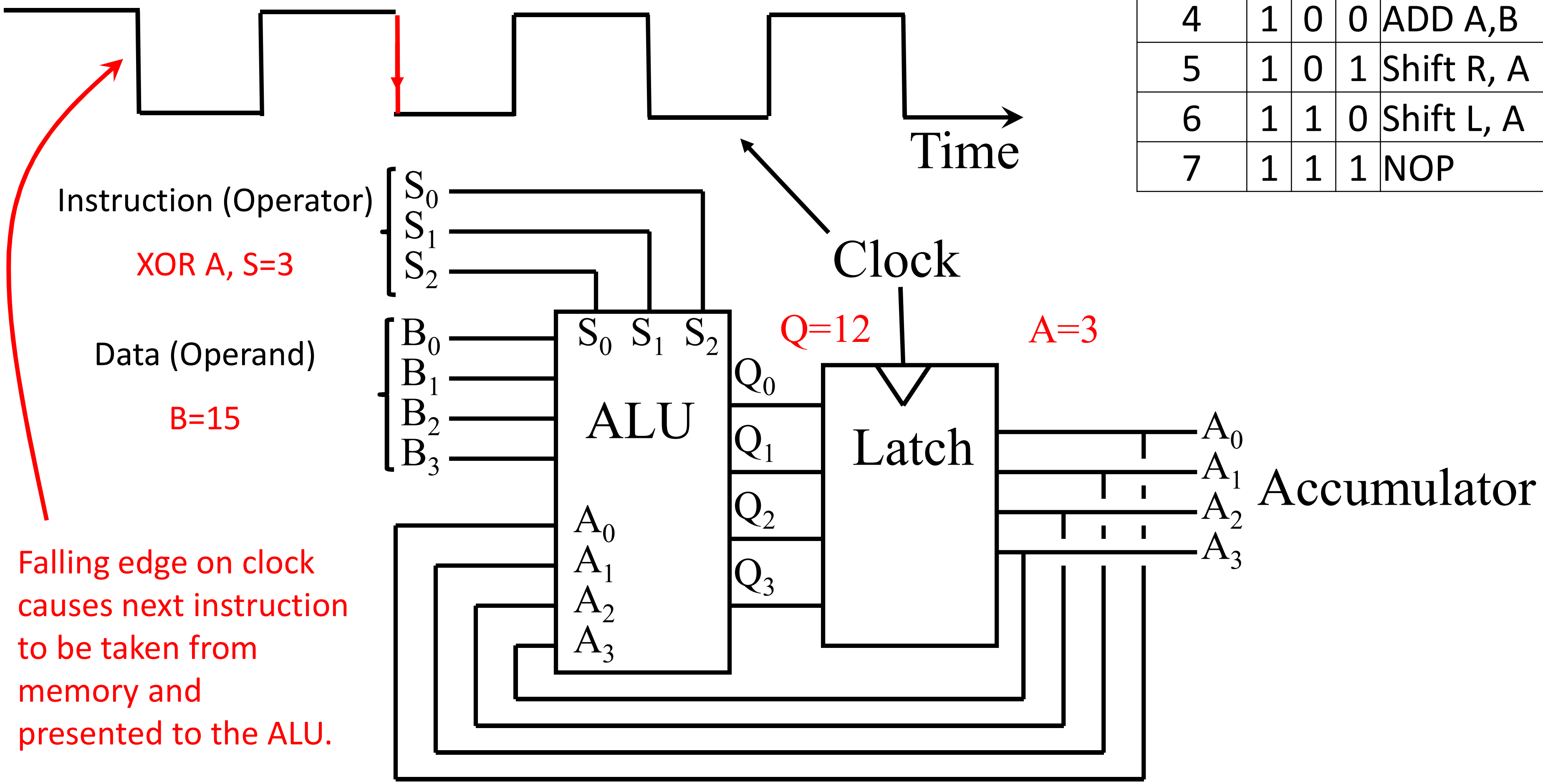


Program to evaluate 5-3

```
Assembly     Machine code   Accumulator
MOV A, 3     S=0, B=3       A=0011b=3
XOR A,15     S=3, B=15      A=1100b=12
ADD A,1      S=4, B=1       A=1101b=13=-3 TC
ADD A,5      S=4, B=5       A=0010b=2
```

ALU Instruction set

S(Hex)	S2	S1	S0	Function
0	0	0	0	MOV A,N
1	0	0	1	AND A,B
2	0	1	0	OR A,B
3	0	1	1	XOR A,B
4	1	0	0	ADD A,B
5	1	0	1	Shift R, A
6	1	1	0	Shift L, A
7	1	1	1	NOP

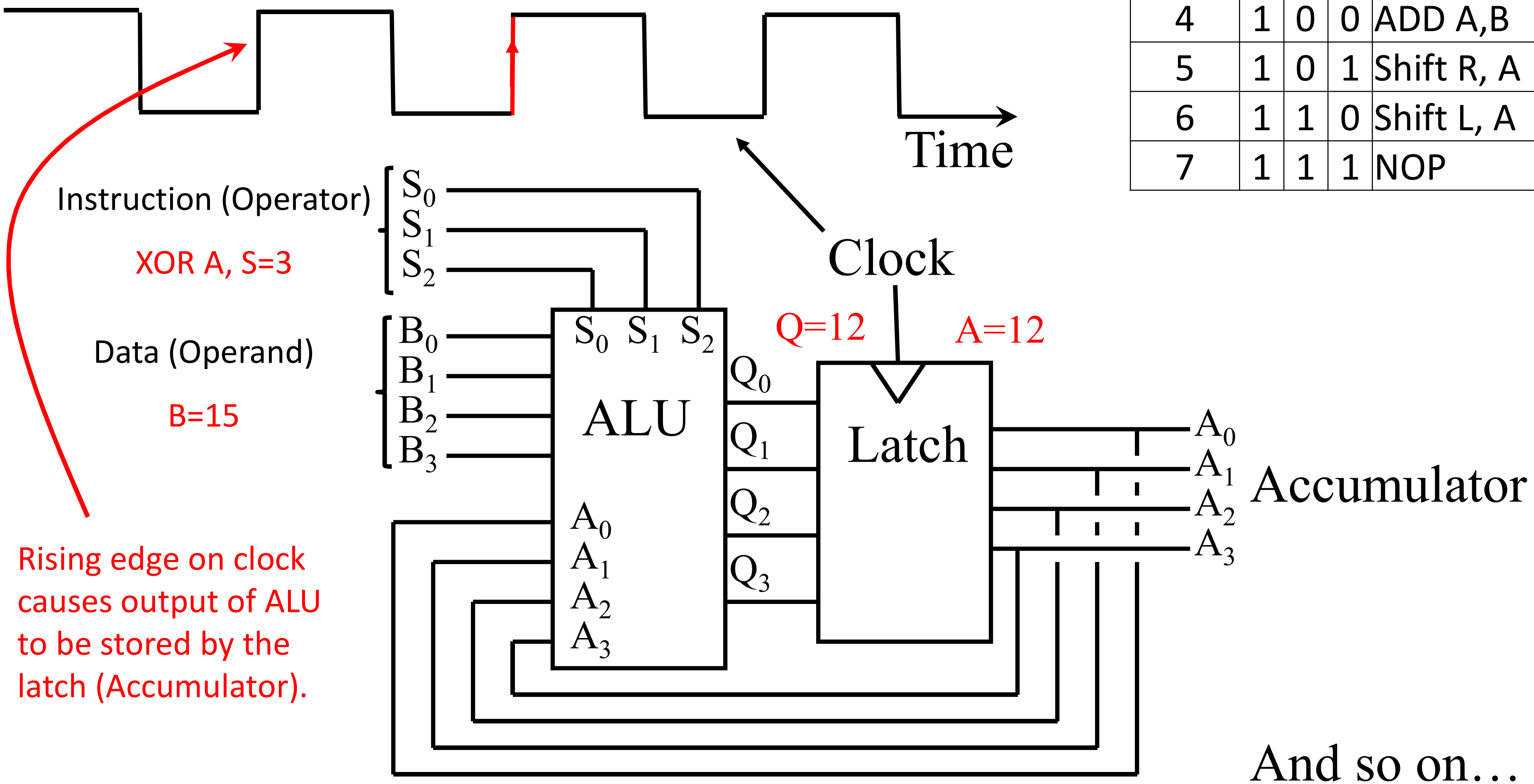


Program to evaluate 5-3

Assembly	Machine code	Accumulator
MOV A, 3	S=0, B=3	A=0011b=3
XOR A,15	S=3, B=15	A=1100b=12
ADD A,1	S=4, B=1	A=1101b=13=-3 TC
ADD A,5	S=4, B=5	A=0010b=2

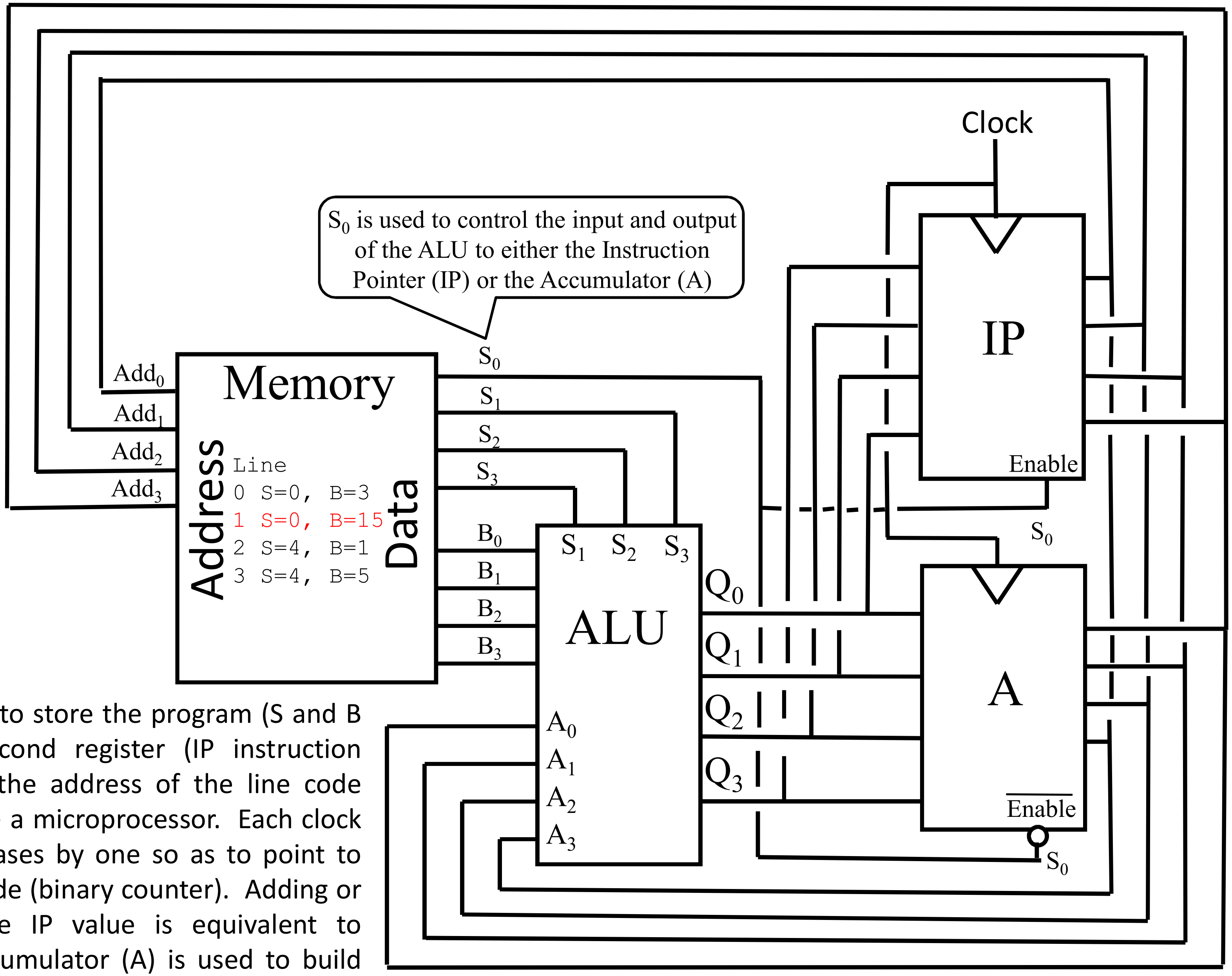
ALU Instruction set

S(Hex)	S2	S1	S0	Function
0	0	0	0	MOV A,N
1	0	0	1	AND A,B
2	0	1	0	OR A,B
3	0	1	1	XOR A,B
4	1	0	0	ADD A,B
5	1	0	1	Shift R, A
6	1	1	0	Shift L, A
7	1	1	1	NOP



And so on…

Extending the calculator to become a simple microprocessor



Adding a memory to store the program (S and B values) and a second register (IP instruction pointer) to store the address of the line code allows us to create a microprocessor. Each clock pulse the IP increases by one so as to point to the next line of code (binary counter). Adding or subtracting to the IP value is equivalent to jumping. The Accumulator (A) is used to build the answer. The IP keeps track of the line of code.

Circuit not complete, however it does convey the concept