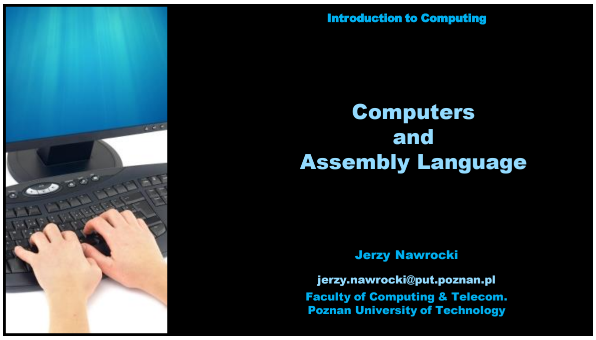
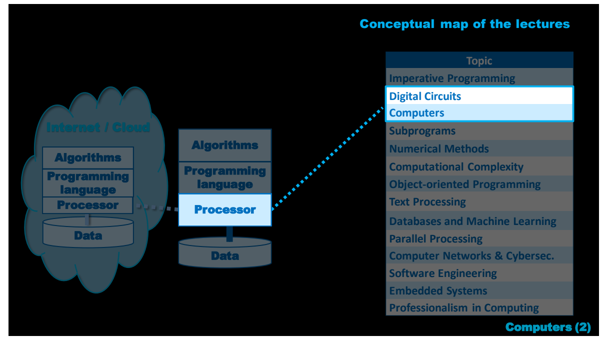
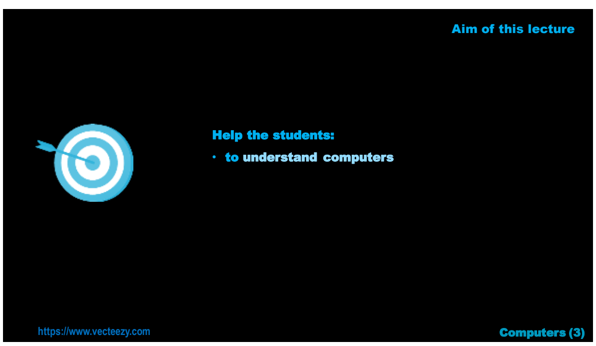
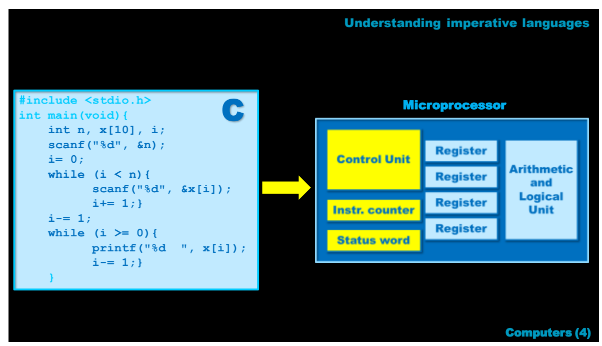
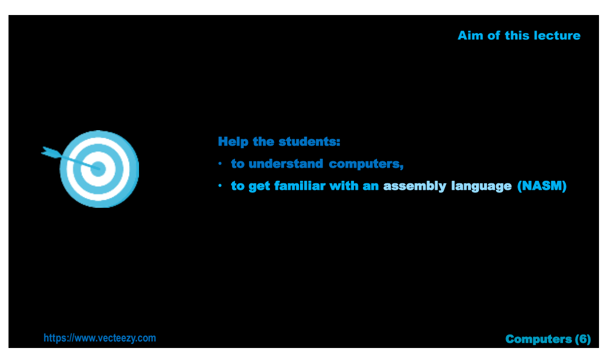

Introduction to Computing
Computers
and
Assembly Language
Jerzy Nawrocki
jerzy.nawrocki@put.poznan.pl
Faculty of Computing & Telecom.
Poznan University of Technology

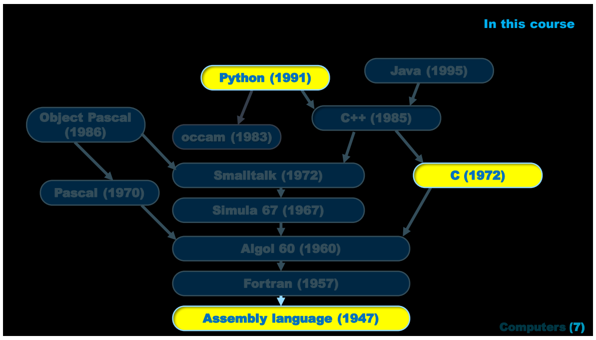

Conceptual map of the lectures
Internet / Cloud
Algorithms
Programming language
Processor
Data
Algorithms
Programming language
Processor
Data
Topic
Imperative Programming
Digital Circuits
Computers
Subprograms
Numerical Methods
Computational Complexity
Object-oriented Programming
Text Processing
Databases and Machine Learning
Parallel Processing
Computer Networks & Cybersec.
Software Engineering
Embedded Systems
Professionalism in Computing
Computers (2)

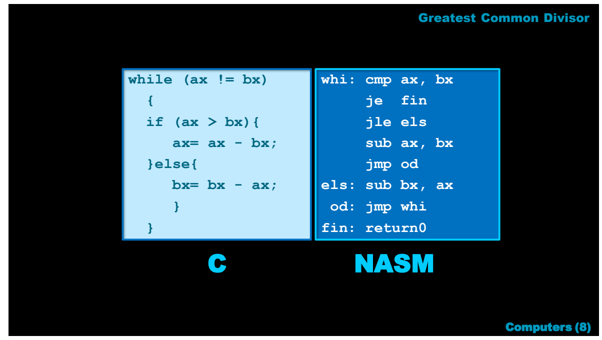

Aim of this lecture
Help the students:
• to understand computers
https://www.vecteezy.com
Computers (3)

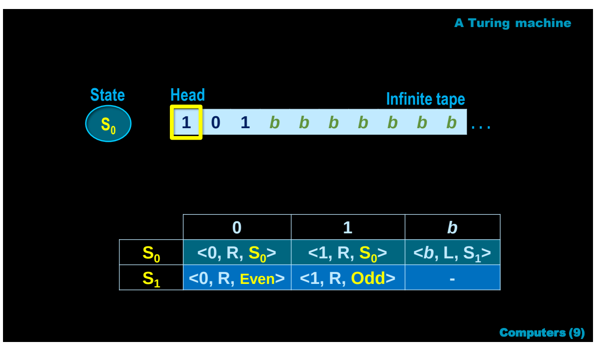

Understanding imperative languages
C
#include <stdio.h>
int main(void){
int n, x[10], i;
scanf("%d", &n);
i= 0;
while (i < n){
scanf("%d", &x[i]);
i+= 1;}
i-= 1;
while (i >= 0){
printf("%d ", x[i]);
i-= 1;}
}
Microprocessor
Control Unit
Instr. counter
Status word
Register
Register
Register
Register
Arithmetic and Logical Unit
Computers (4)

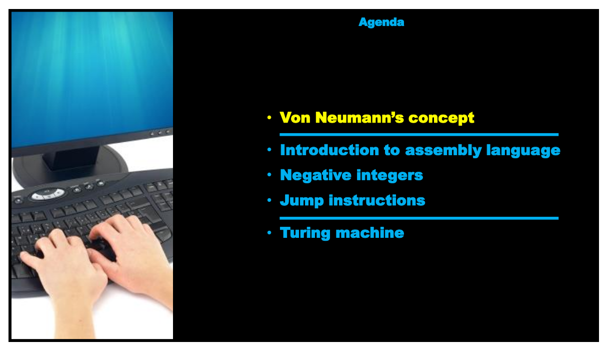

von Neumann's concept
Program as data
John Luis von Neumann
1903 – 1957
Institute for Advanced Studies
Los Alamos Laboratory
Computers (5)

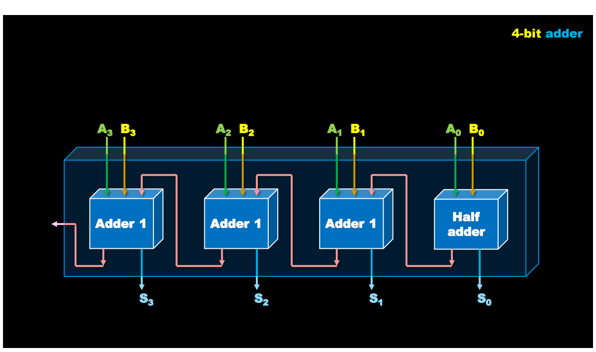

Aim of this lecture
Help the students:
• to understand computers,
• to get familiar with an assembly language (NASM)
https://www.vecteezy.com
Computers (6)

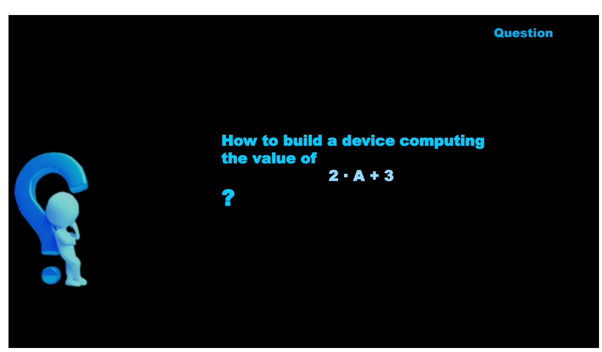

In this course
Python (1991)
Java (1995)
Object Pascal (1986)
occam (1983)
C++ (1985)
Smalltalk (1972)
C (1972)
Pascal (1970)
Simula 67 (1967)
Algol 60 (1960)
Fortran (1957)
Assembly language (1947)
Computers (7)

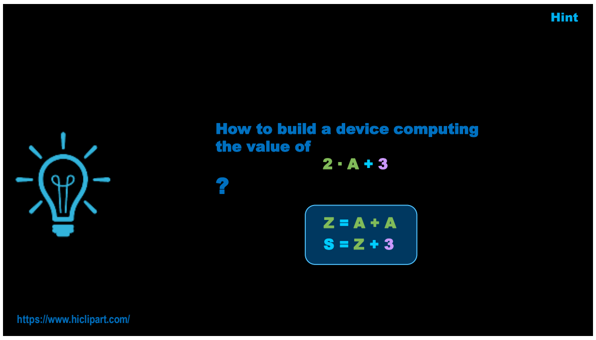

Greatest Common Divisor
while (ax != bx)
{
if (ax > bx){
ax= ax - bx;
}else{
bx= bx - ax;
}
}
whi: cmp ax, bx
je fin
jle els
sub ax, bx
jmp od
els: sub bx, ax
od: jmp whi
fin: return0
C
NASM
Computers (8)

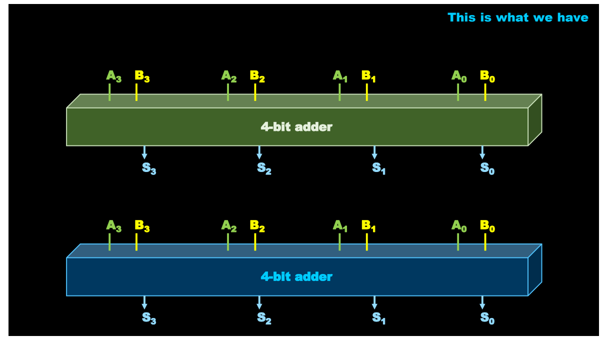

A Turing machine
State
S_0
Head
Infinite tape
1 0 1 b b b b b b b ...
0 | 1 | b
S_0 | <0, R, S_0> | <1, R, S_0> | <b, L, S_1>
S_1 | <0, R, Even> | <1, R, Odd> | -
Computers (9)

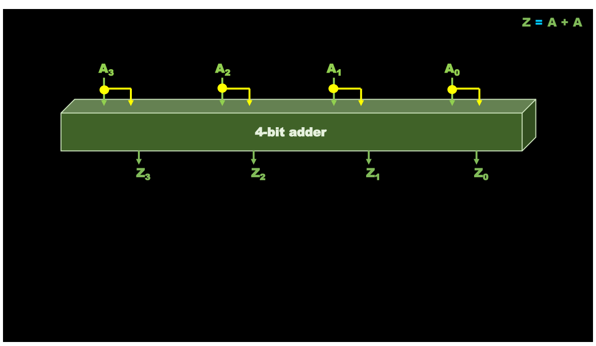

Agenda
• Von Neumann's concept
• Introduction to assembly language
• Negative integers
• Jump instructions
• Turing machine

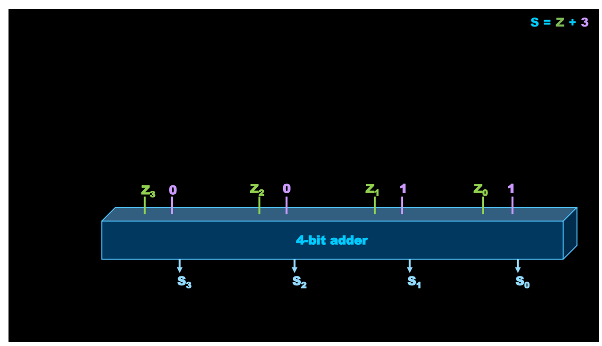

4-bit adder
A_3 B_3
A_2 B_2
A_1 B_1
A_0 B_0
Adder 1
Adder 1
Adder 1
Half adder
S_3
S_2
S_1
S_0

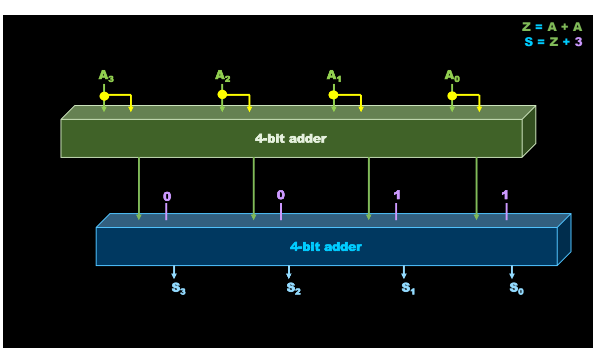

Question
How to build a device computing the value of
2 · A + 3
?

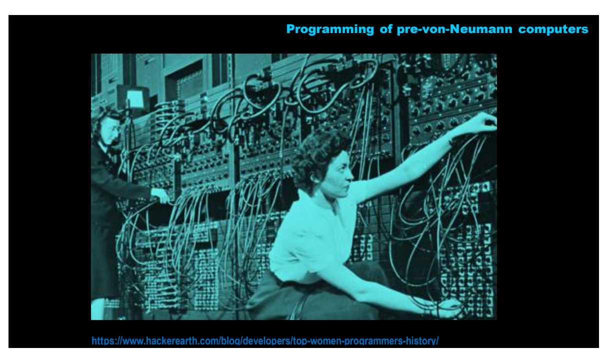

Hint
How to build a device computing the value of
2 · A + 3
?
Z = A + A
S = Z + 3
https://www.hiclipart.com/

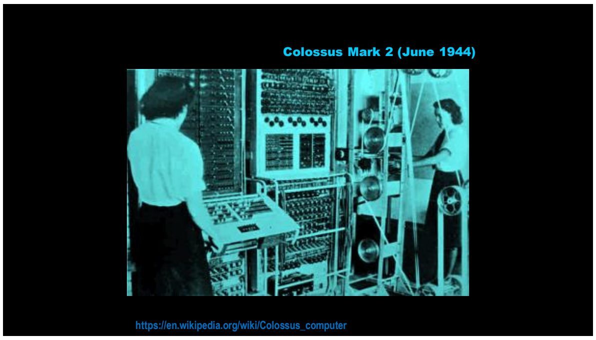

This is what we have
A_3 B_3 A_2 B_2 A_1 B_1 A_0 B_0
4-bit adder
S_3 S_2 S_1 S_0
A_3 B_3 A_2 B_2 A_1 B_1 A_0 B_0
4-bit adder
S_3 S_2 S_1 S_0

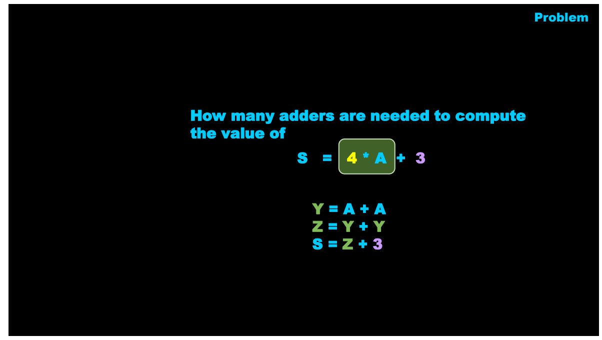

Z = A + A
A_3 A_2 A_1 A_0
4-bit adder
Z_3 Z_2 Z_1 Z_0

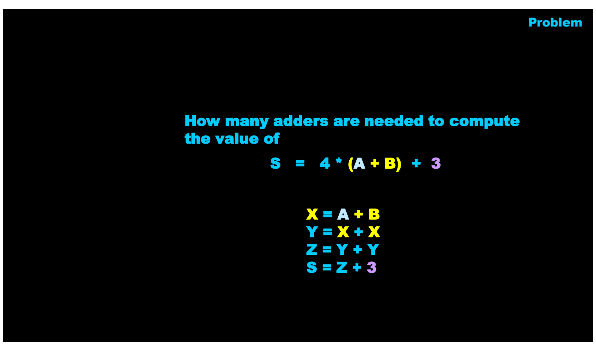

S = Z + 3
Z_3 0 Z_2 0 Z_1 1 Z_0 1
4-bit adder
S_3 S_2 S_1 S_0

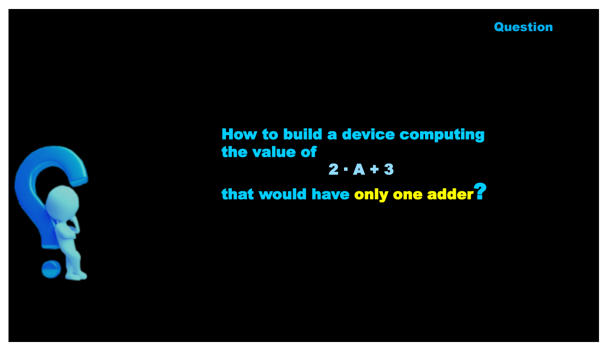

Z = A + A
S = Z + 3
A_3 A_2 A_1 A_0
4-bit adder
0 0 1 1
4-bit adder
S_3 S_2 S_1 S_0

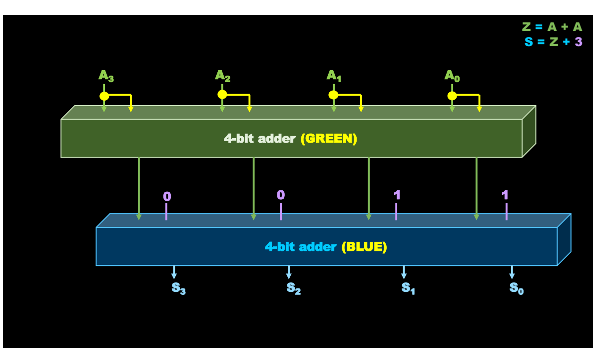

Programming of pre-von-Neumann computers
https://www.hackerearth.com/blog/developers/top-women-programmers-history/

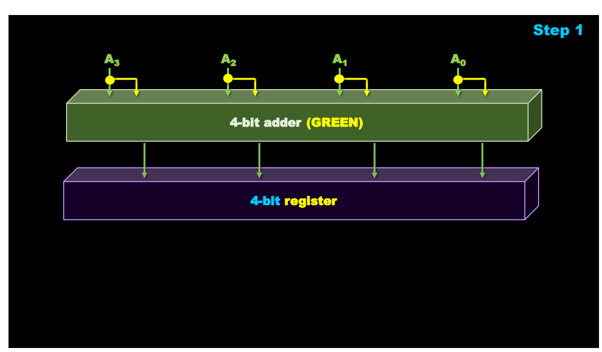

Colossus Mark 2 (June 1944)
https://en.wikipedia.org/wiki/Colossus_computer

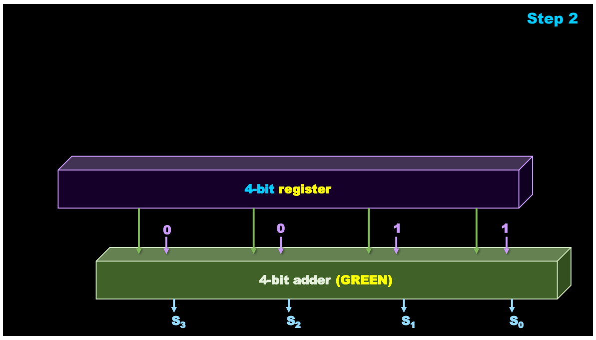

Problem
How many adders are needed to compute the value of
S = 4 * A + 3
Y = A + A
Z = Y + Y
S = Z + 3

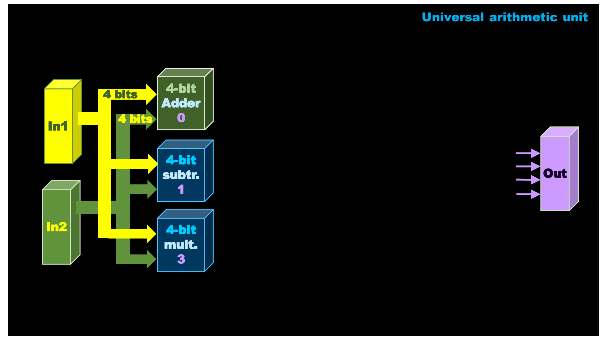

Problem
How many adders are needed to compute the value of
S = 4 * (A + B) + 3
X = A + B
Y = X + X
Z = Y + Y
S = Z + 3

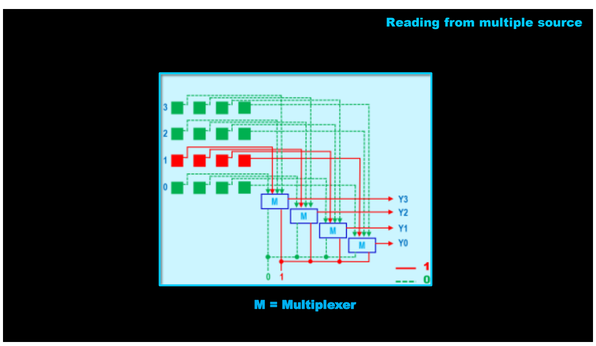

Question
How to build a device computing the value of
2 · A + 3
that would have only one adder?

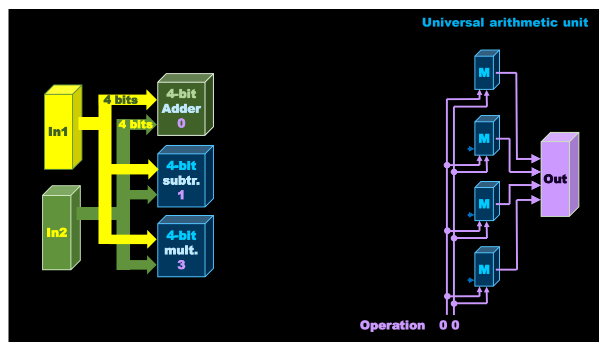

Z = A + A
S = Z + 3
A_3
A_2
A_1
A_0
4-bit adder (GREEN)
0
0
1
1
4-bit adder (BLUE)
S_3
S_2
S_1
S_0

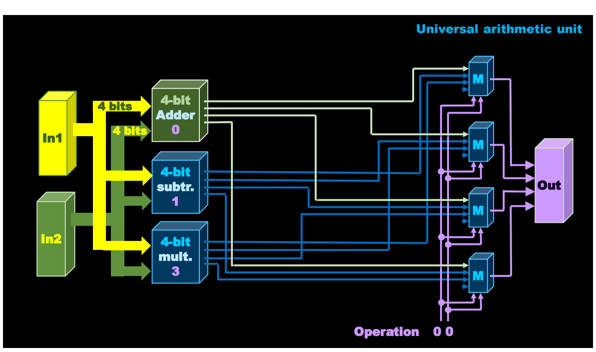

Step 1
A_3
A_2
A_1
A_0
4-bit adder (GREEN)
4-bit register

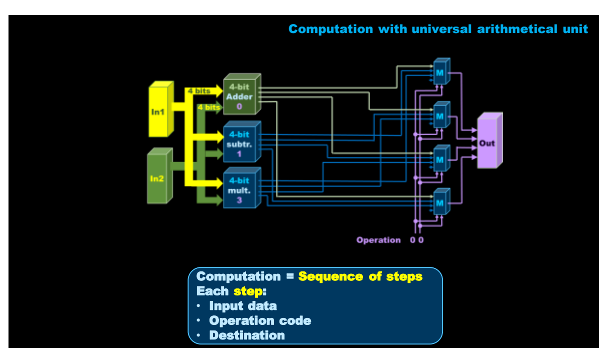

Step 2
4-bit register
0
0
1
1
4-bit adder (GREEN)
S_3
S_2
S_1
S_0

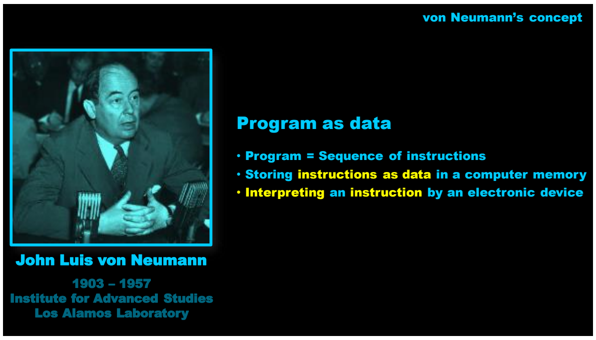

Universal arithmetic unit
In1
In2
4 bits
4 bits
4-bit Adder 0
4-bit subtr. 1
4-bit mult. 3
Out


Reading from multiple source
M = Multiplexer

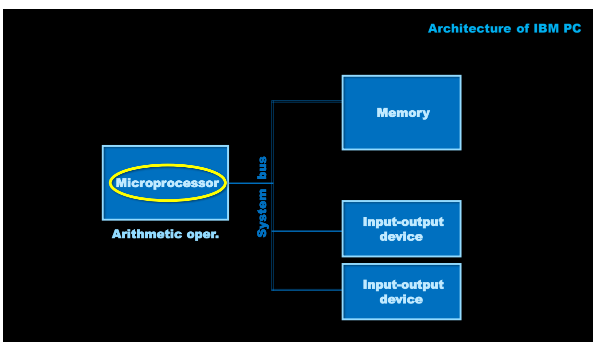

Universal arithmetic unit
In1
In2
4 bits
4 bits
4-bit Adder 0
4-bit subtr. 1
4-bit mult. 3
M
Out
Operation 0 0

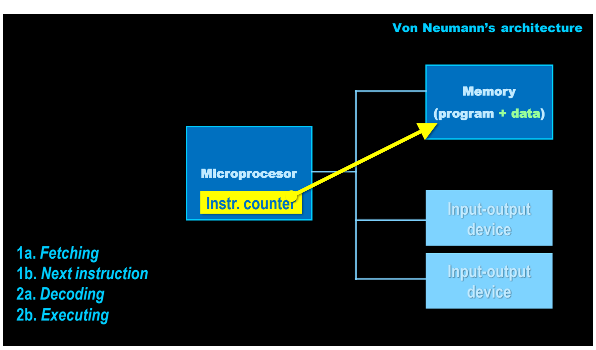

Universal arithmetic unit
In1
In2
4 bits
4 bits
4-bit Adder 0
4-bit subtr. 1
4-bit mult. 3
M
Out
Operation 0 0

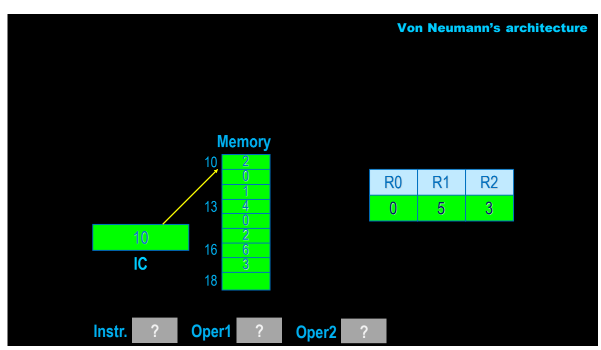

Computation with universal arithmetical unit
Computation = Sequence of steps
Each step:
• Input data
• Operation code
• Destination

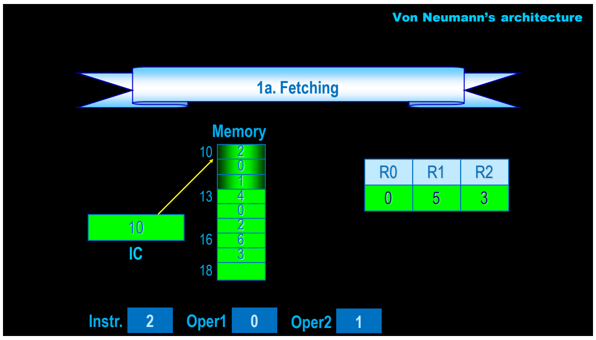


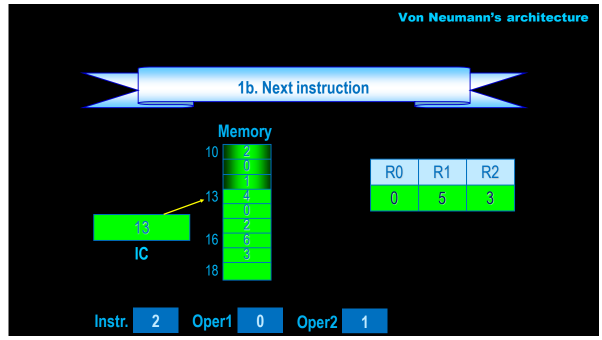
Code	Instruction	Example	Meaning
1	MoveRegCon (R, C)	1 1 1	R1 ← 1
2	MoveRegReg (Rd, Rs)	2 0 1	R0 ← R1
3	AddRegReg (Rd, Rs)	3 1 2	R1 ← R1 + R2
4	SubRegReg (Rd, Rs)	4 0 2	R0 ← R0 – R2
5	NegReg (R)	5 2	R2 ← – R2
6	Inter (C)	6 3	

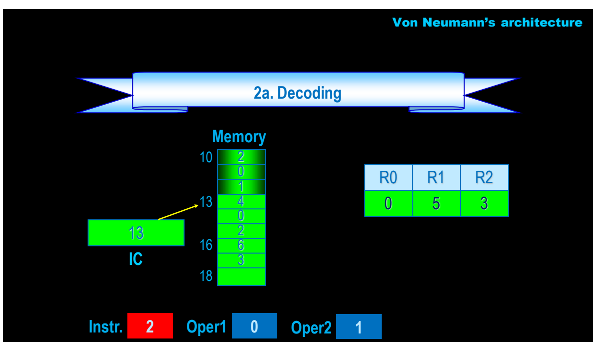






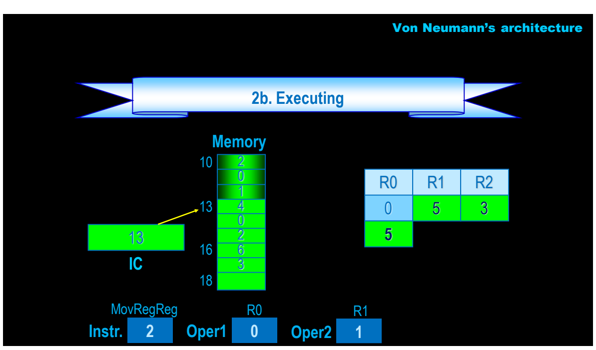






Exemplary codes

Code	Instruction	Example	Meaning
1	MoveRegCon (R, C)	1 1 1	R1 ← 1
2	MoveRegReg (Rd, Rs)	2 0 1	R0 ← R1
3	AddRegReg (Rd, Rs)	3 1 2	R1 ← R1 + R2
4	SubRegReg (Rd, Rs)	4 0 2	R0 ← R0 – R2
5	NegReg (R)	5 2	R2 ← – R2
6	Inter (C)	6 3	

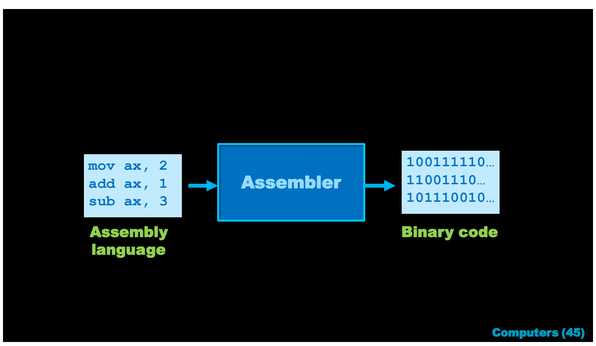


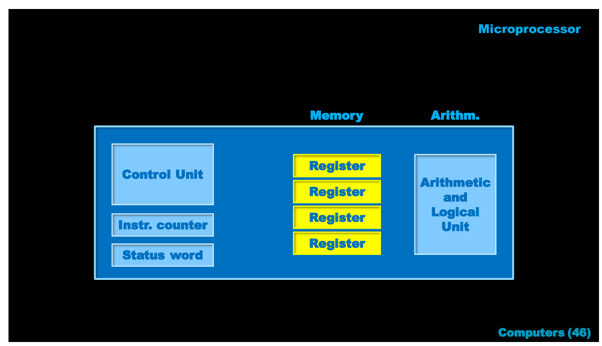
Agenda

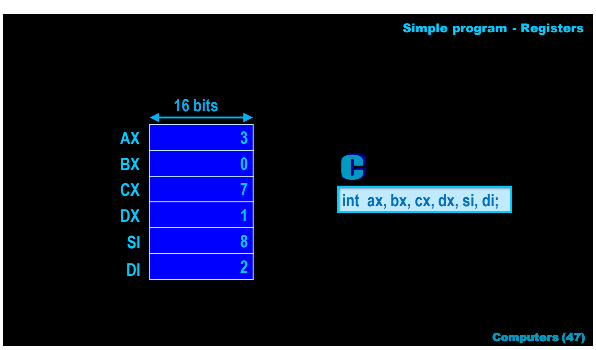
- Von Neumann's concept
- Introduction to assembly language
- Negative integers
- Jump instructions
- Turing machine

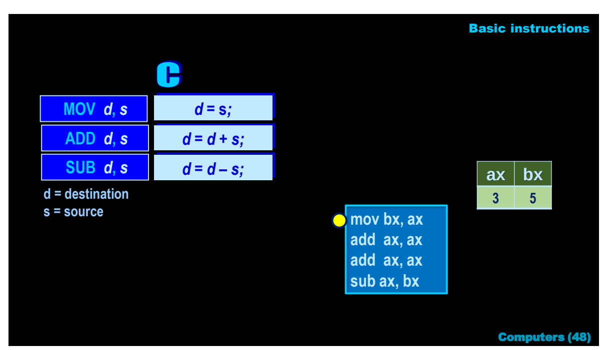
Code	Instruction	Example	Meaning
1	MoveRegCon (R, C)	1 1 1	R1 ← 1
2	MoveRegReg (Rd, Rs)	2 0 1	R0 ← R1
3	AddRegReg (Rd, Rs)	3 1 2	R1 ← R1 + R2
4	SubRegReg (Rd, Rs)	4 0 2	R0 ← R0 – R2
5	NegReg (R)	5 2	R2 ← – R2
6	Inter (C)	6 3	

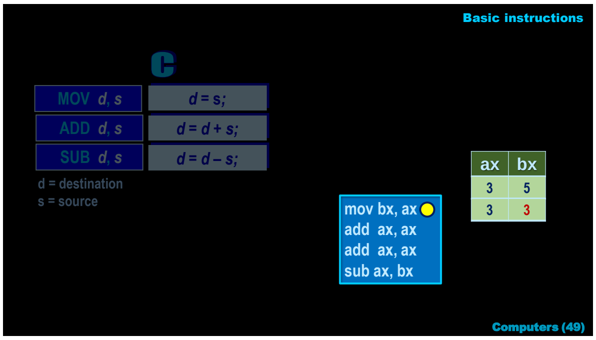
Code	Instruction	Example	Meaning
1	MoveRegCon (R, C)	1 1 1	R1 ← 1
2	MoveRegReg (Rd, Rs)	2 0 1	R0 ← R1
3	AddRegReg (Rd, Rs)	3 1 2	R1 ← R1 + R2
4	SubRegReg (Rd, Rs)	4 0 2	R0 ← R0 – R2
5	NegReg (R)	5 2	R2 ← – R2
6	Inter (C)	6 3	

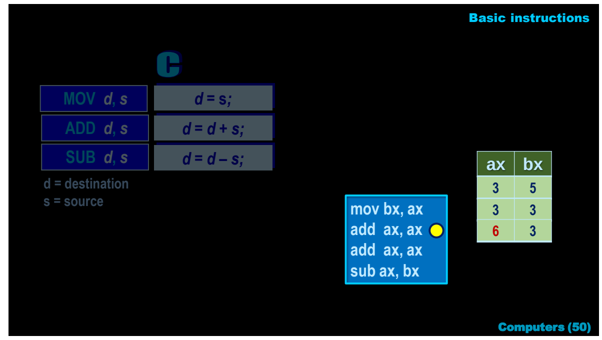


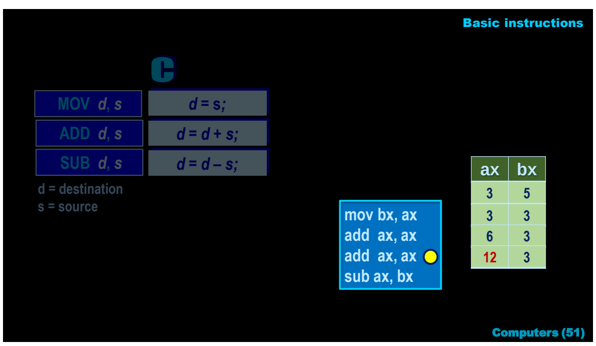


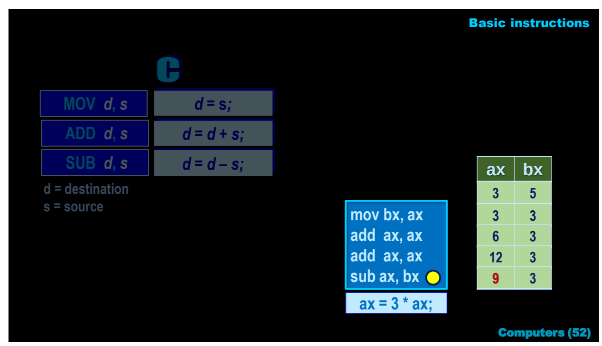


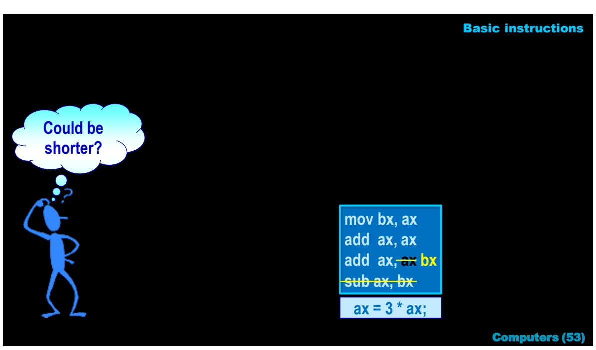


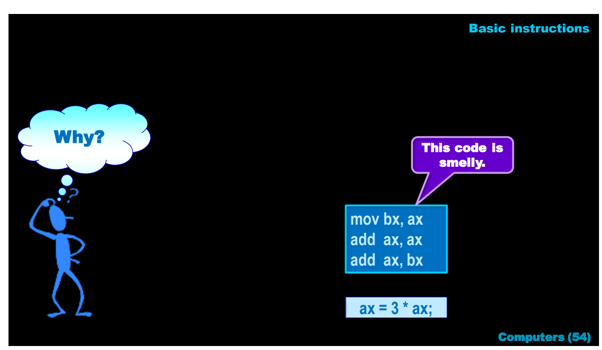

Basic instructions
C
MOV d, s
d = s;
ADD d, s
d = d + s;
SUB d, s
d = d – s;
d = destination
s = source
mov bx, ax
add ax, ax
add ax, ax
sub ax, bx
ax bx
3 5
3 3
Computers (49)


Basic instructions
C
MOV d, s
d = s;
ADD d, s
d = d + s;
SUB d, s
d = d – s;
d = destination
s = source
mov bx, ax
add ax, ax
add ax, ax
sub ax, bx
ax bx
3 5
3 3
6 3
Computers (50)


Basic instructions
C
MOV d, s
d = s;
ADD d, s
d = d + s;
SUB d, s
d = d – s;
d = destination
s = source
mov bx, ax
add ax, ax
add ax, ax
sub ax, bx
ax bx
3 5
3 3
6 3
12 3
Computers (51)


Basic instructions
C
MOV d, s
d = s;
ADD d, s
d = d + s;
SUB d, s
d = d – s;
d = destination
s = source
mov bx, ax
add ax, ax
add ax, ax
sub ax, bx
ax = 3 * ax;
ax bx
3 5
3 3
6 3
12 3
9 3
Computers (52)


Basic instructions
Could be shorter?
mov bx, ax
add ax, ax
add ax, ~~ax~~ bx
~~sub ax, bx~~
ax = 3 * ax;
Computers (53)


Basic instructions
Why?
This code is smelly.
mov bx, ax
add ax, ax
add ax, bx
ax = 3 * ax;
Computers (54)

Multiplication and division

MUL s
DIV s

A	B	A*B
9	9	81
99	9	891
999	9	8 991
9999	9	89 991
99	99	9 801
999	99	98 901
9999	99	989 901
999	999	998 001

Computers (55)

Multiplication & division

C

MUL z	(dx:ax) = ax * z;
DIV z	ax = (dx:ax) / z; dx = (dx:ax) % z;

```
mul ax
mov bx, 10
div bx
mov bx, dx
```

dx	ax	bx
7	7	7

Computers (56)

Multiplication & division

C

MUL z	(dx:ax) = ax * z;
DIV z	ax = (dx:ax) / z; dx = (dx:ax) % z;

```
mul ax
mov bx, 10
div bx
mov bx, dx
```

dx	ax	bx
7	7	7
0	49	7

Computers (57)

Multiplication & division

C

MUL z	(dx:ax) = ax * z;
DIV z	ax = (dx:ax) / z; dx = (dx:ax) % z;

```
mul ax
mov bx, 10
div bx
mov bx, dx
```

dx	ax	bx
7	7	7
0	49	7
0	49	10

Computers (58)

Multiplication & division

C

MUL z	(dx:ax) = ax * z;
DIV z	ax = (dx:ax) / z; dx = (dx:ax) % z;

```
mul ax
mov bx, 10
div bx
mov bx, dx
```

dx	ax	bx
7	7	7
0	49	7
0	49	10
9	4	10

Computers (59)

Multiplication & division

C

MUL z	(dx:ax) = ax * z;
DIV z	ax = (dx:ax) / z; dx = (dx:ax) % z;

```
mul ax
mov bx, 10
div bx
mov bx, dx
```

bx = (ax*ax)%10;

dx	ax	bx
7	7	7
0	49	7
0	49	10
9	4	10
9	4	9

Computers (60)

Auxiliary macros

AuxMacros.asm

```
%macro Print1Hex 2
%deftok Reg %1
    push Reg
    push ax
    shr Reg, %2*4
    and Reg, 0xF
    mov al, [HexDig + Reg]
    mov [msg + len - 2 - %2], al
    pop ax
    pop Reg
%endmacro
%macro printReg 1
    %defstr Regi %1
    %strcat eRegi 'e', Regi
    push eax
    Print1Hex eRegi, 0
    Print1Hex eRegi, 1
    Print1Hex eRegi, 2
    Print1Hex eRegi, 3
    mov ax, Regi
    mov [msg], ax ; msg= 'rr = hhhh'
    push ebx
    push ecx
    push edx
    mov edx, len
    mov ecx, msg
    mov ebx, 1
    mov eax, 4
    int    0x80       ;sys_write
    pop edx
    pop ecx
    pop ebx
    pop eax
%endmacro
%macro return0 0
    mov    eax, 1
    int    0x80        ;sys_exit
%endmacro
```

```
%include AuxMacros.asm
 section .text
  global _start
_start:
    mov dx, 258        ; dx= 258;
    printReg dx        ; printf(("%h\n", dx);
    return0            ; return 0;
section     .data
HexDig  db  '0', '1', '2', '3'
        db  '4', '5', '6', '7'
        db  '8', '9', 'A', 'B'
        db  'C', 'D', 'E', 'F'
msg     db  '12 = 0000', 0xa
len     equ $ - msg
```

Computers (61)

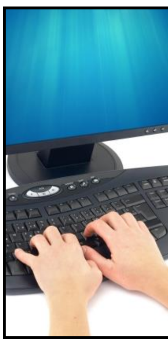
Example of a simple program

```
mov bx, ax
add ax, ax
add ax, bx
```

```
%include AuxMacros.asm
 section .text
  global _start
_start:
    mov ax, 2        ; input data
    mov bx, ax
    add ax, ax
    add ax, bx
    printReg dx      ; printf(("%h\n", dx);
    return0          ; return 0;
section     .data
HexDig  db  '0', '1', '2', '3'
        db  '4', '5', '6', '7'
        db  '8', '9', 'A', 'B'
        db  'C', 'D', 'E', 'F'
msg     db  '12 = 0000', 0xa
len     equ $ - msg
```

Computers (62)

Agenda



- Von Neumann's concept
- Introduction to assembly language
- **Negative integers**
- Jump instructions
- Turing machine

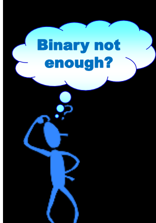
Binary arithmetic

101_2

$1*2^2 + 0*2^1 + 1*2^0$

$= 4 + 0 + 1$

Problem



$\underbrace{1\,0\,0\,0\,0\,0\,0\,0\,0\,1_2}_{\text{10 digits}} = \underbrace{512_{10}}_{\text{3 digits}}$

but ... $\underbrace{9_{10}}_{\text{1 digit}} = \underbrace{1001_2}_{\text{4 digits}}$

1010
1011
1100
1101
1110
1111

Not represented with 1 decimal digit

Computers (65)

Another option

Byte: $\underbrace{0\,0\,1\,1}_{\text{4 digits}}\,\vdots\,\underbrace{0\,1\,1\,1_2}_{\text{4 digits}}$

4 binary digits ⟷ 1 hexadecimal digit

Computers (66)

Hexadecimal digits

Binary	Hex	Decimal
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9

Binary	Hex	Decimal
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

Computers (67)

Hexadecimal arithmetic

101_{16}

$1*16^2 + 0*16^1 + 1*16^0$

$= 256 + 0 + 1$

Computers (68)

Hexadecimal arithmetic

BAD

$11*16^2 + 10*16^1 + 13*16^0$

$= 2\,816 + 160 + 13 = 2\,989$

Computers (69)

Indirect addition

		11
$28F_{16}$	$2*256 + 8*16 + 15$	655_{10}
$+\ 37F_{16}$	$3*256 + 7*16 + 15$	$+\ 895_{10}$
$60E_{16}$	$6*256 + 0*16 + 14$	1550_{10}

Computers (70)

Direct addition

1
28F
+ 37F
E

$F_{16} + F_{16} =$

$15_{10} + 15_{10} = 30_{10}$

$30_{10} : 16_{10} =$

1 reminder 14_{10} =

1 reminder E_{16}

Computers (71)

Direct addition

011
28F
+ 37F
60E

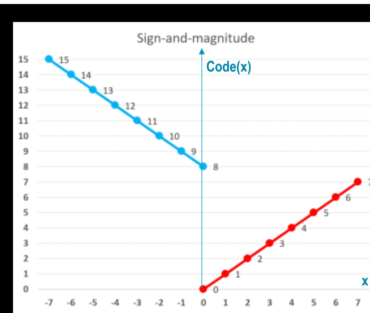
Computers (72)

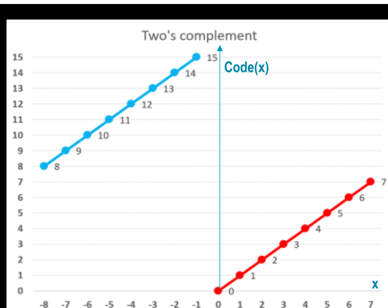
Sign-and-Magnitude – 4 bits

0	0	1	1	= 3
1	0	1	1	= – 3

```
if (a<0 && b>=0)
   if (abs(a) > abs(b)) {
      ResultAbs= abs(a) – abs(b);
      ResultSign= Negative;
      }
. . .
```

Computers (73)





2's complement

$$\text{Code}(b) = \begin{cases} b & \text{jeśli } b \geq 0 \\ 2^n - |b| & \text{jeśli } b < 0 \end{cases}$$

n **bits correspond to the following range:**

$[-2^{n-1}, 2^{n-1} - 1]$

Computers (76)

Negative numbers – 3 bits

Number	Code
3	3
2	2
1	1
0	0
-1	7
-2	6
-3	5
-4	4

Codes for -1 … -4: $2^3 - |b|$

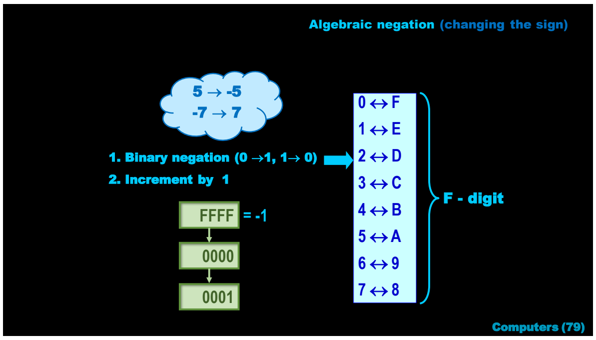
Computers (77)

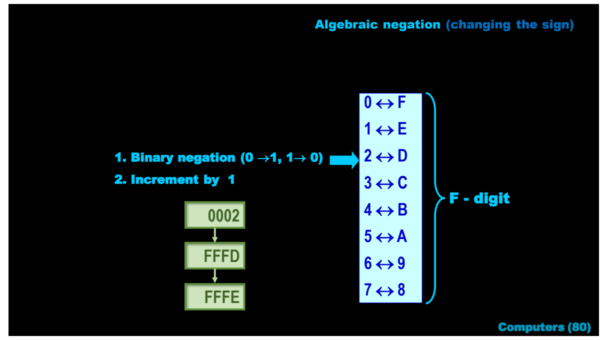
Negative numbers – 16 bits

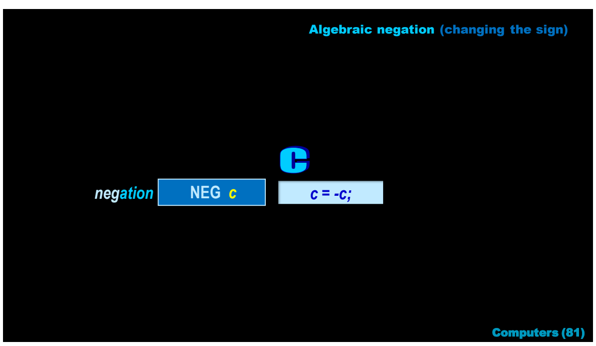
$$\begin{array}{r} 1111 \\ \text{FFFF} \\ +\quad 2 \\ \hline 0001 \end{array} = \begin{array}{r} -1 \\ +\quad 2 \\ \hline 1 \end{array}$$

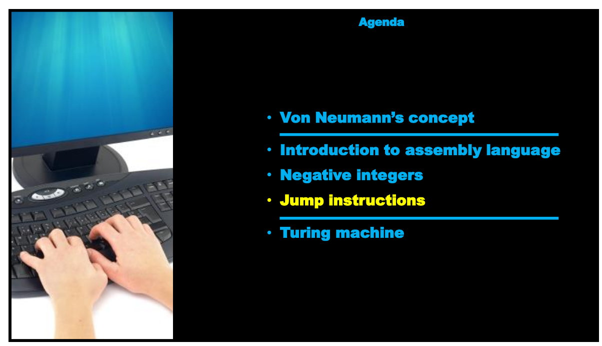
32767	7FFF
. . .	. . .
1	1
0	0
-1	FFFF
. . .	. . .
-32767	8001
-32768	8000

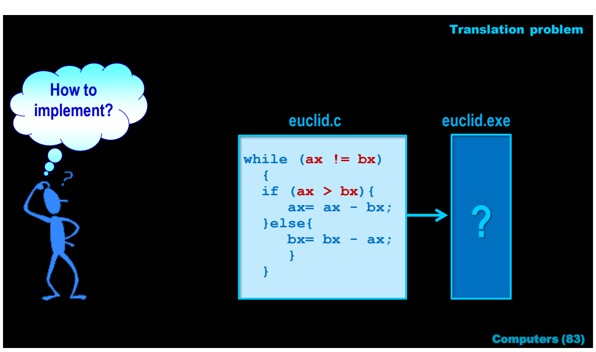
Computers (78)


Algebraic negation (changing the sign)
5 → -5
-7 → 7
1. Binary negation (0 →1, 1→ 0)
2. Increment by 1
FFFF = -1
0000
0001
0 ↔ F
1 ↔ E
2 ↔ D
3 ↔ C
4 ↔ B
5 ↔ A
6 ↔ 9
7 ↔ 8
F - digit
Computers (79)

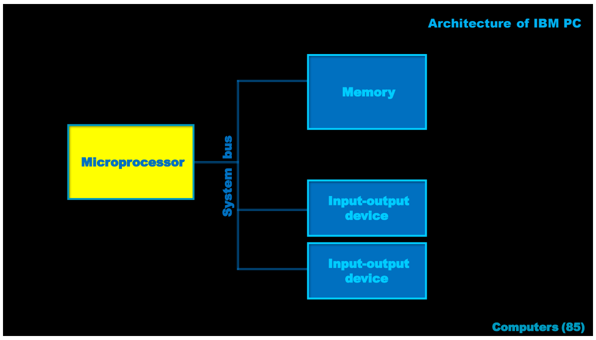

Algebraic negation (changing the sign)
1. Binary negation (0 →1, 1→ 0)
2. Increment by 1
0002
FFFD
FFFE
0 ↔ F
1 ↔ E
2 ↔ D
3 ↔ C
4 ↔ B
5 ↔ A
6 ↔ 9
7 ↔ 8
F - digit
Computers (80)

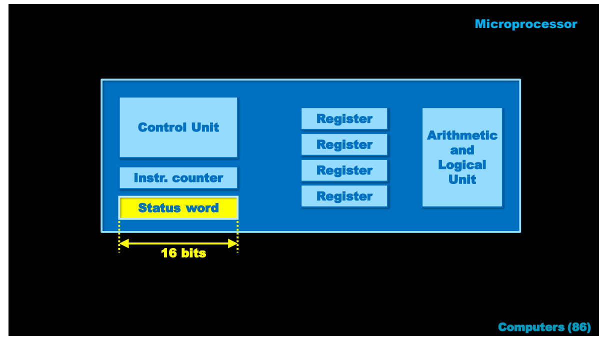

Algebraic negation (changing the sign)
C
negation
NEG c
c = -c;
Computers (81)

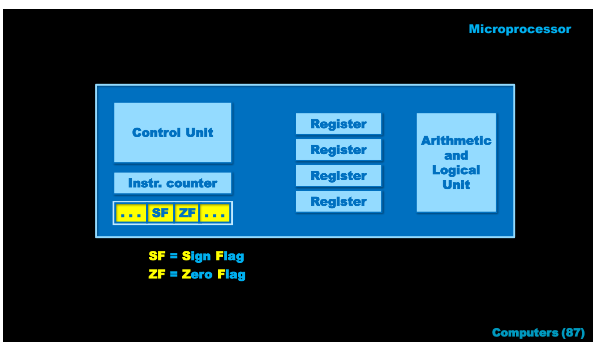

Agenda
Von Neumann's concept
Introduction to assembly language
Negative integers
Jump instructions
Turing machine

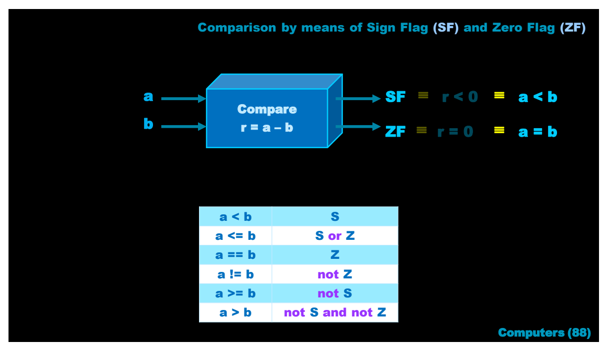

Translation problem
How to implement?
euclid.c
euclid.exe
while (ax != bx)
{
if (ax > bx){
ax= ax - bx;
}else{
bx= bx - ax;
}
}
?
Computers (83)

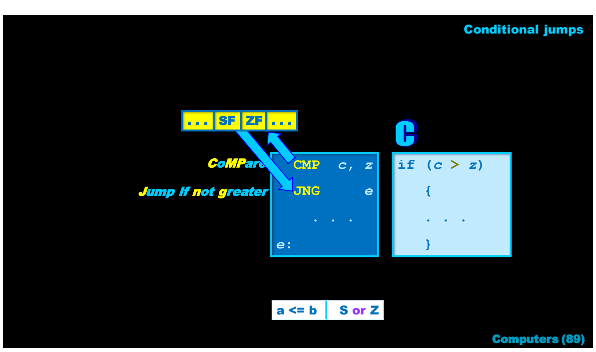

Jump instructions
jump if equal
JE e
JNE e
jump if not equal
jump if greater
JG e
JNG e
jump if not greater
jump if less
JL e
JNL e
jump if not less
JMP e
jump
Computers (84)

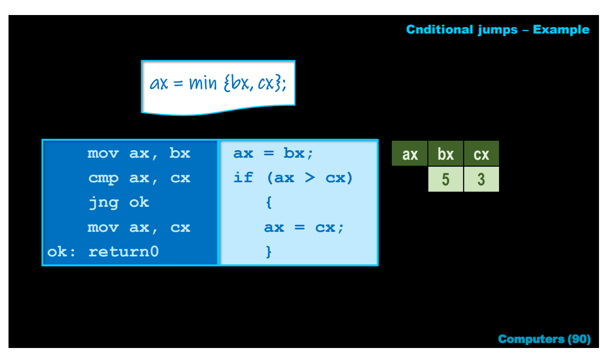

Architecture of IBM PC
Memory
Microprocessor
System bus
Input-output device
Input-output device
Computers (85)

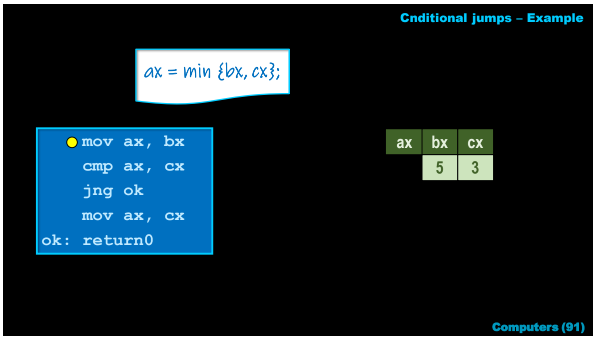

Microprocessor
Control Unit
Instr. counter
Status word
16 bits
Register
Register
Register
Register
Arithmetic and Logical Unit
Computers (86)

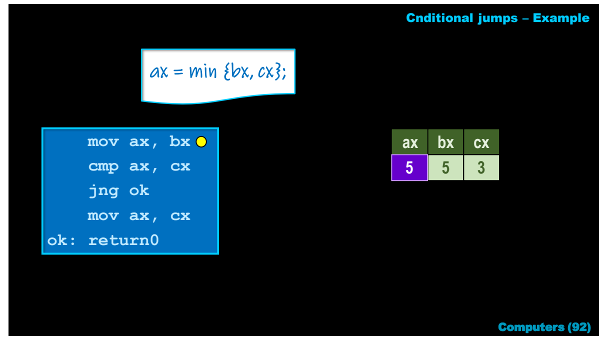

Microprocessor
Control Unit
Instr. counter
... SF ZF ...
Register
Register
Register
Register
Arithmetic and Logical Unit
SF = Sign Flag
ZF = Zero Flag
Computers (87)

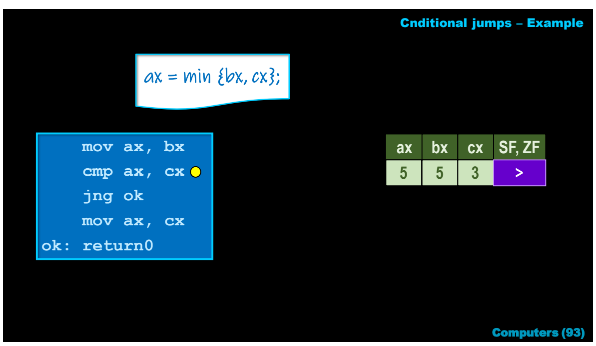

Comparison by means of Sign Flag (SF) and Zero Flag (ZF)
a
b
Compare
r = a – b
SF ≡ r < 0 ≡ a < b
ZF ≡ r = 0 ≡ a = b
a < b | S
a <= b | S or Z
a == b | Z
a != b | not Z
a >= b | not S
a > b | not S and not Z
Computers (88)

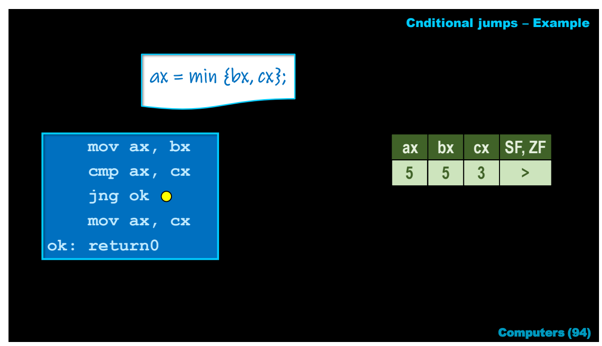

Conditional jumps
... SF ZF ...
CoMPare
Jump if not greater
CMP c, z
JNG e
. . .
e:
C
if (c > z)
{
. . .
}
a <= b | S or Z
Computers (89)

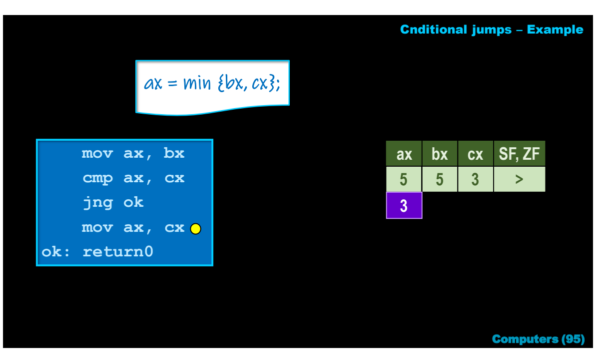

Cnditional jumps – Example
ax = min {bx, cx};
mov ax, bx
cmp ax, cx
jng ok
mov ax, cx
ok: return0
ax = bx;
if (ax > cx)
{
ax = cx;
}
ax bx cx
5 3
Computers (90)

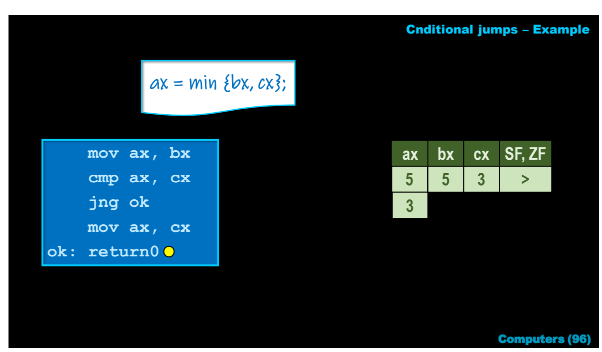

Cnditional jumps – Example
ax = min {bx, cx};
mov ax, bx
cmp ax, cx
jng ok
mov ax, cx
ok: return0
ax bx cx
5 3
Computers (91)

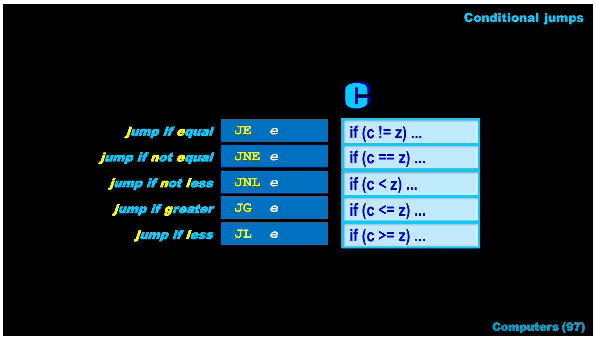

Cnditional jumps – Example
ax = min {bx, cx};
mov ax, bx
cmp ax, cx
jng ok
mov ax, cx
ok: return0
ax bx cx
5 5 3
Computers (92)


Cnditional jumps – Example
ax = min {bx, cx};
mov ax, bx
cmp ax, cx
jng ok
mov ax, cx
ok: return0
ax bx cx SF, ZF
5 5 3 >
Computers (93)

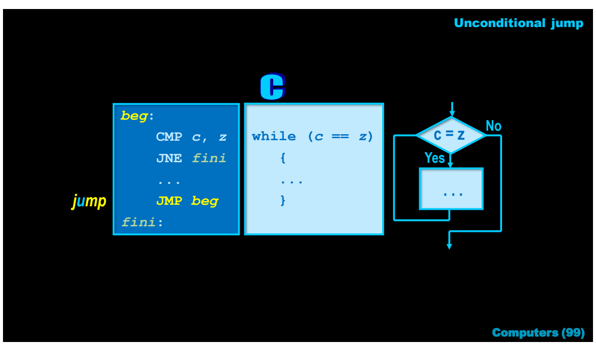

Cnditional jumps – Example
ax = min {bx, cx};
mov ax, bx
cmp ax, cx
jng ok
mov ax, cx
ok: return0
ax bx cx SF, ZF
5 5 3 >
Computers (94)

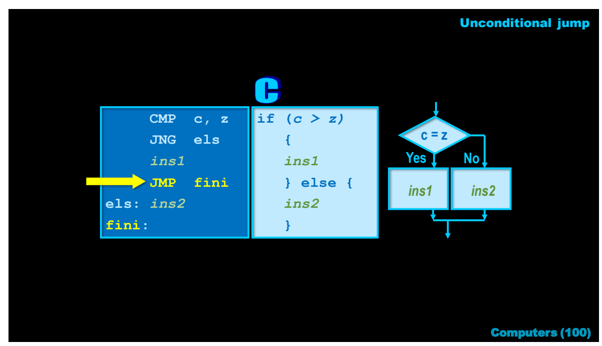

Cnditional jumps – Example
ax = min {bx, cx};
mov ax, bx
cmp ax, cx
jng ok
mov ax, cx
ok: return0
ax bx cx SF, ZF
5 5 3 >
3
Computers (95)

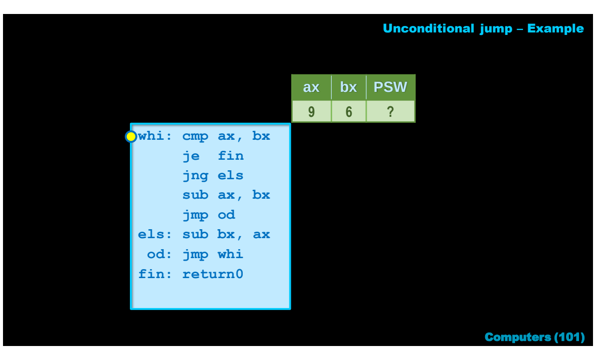

Cnditional jumps – Example
ax = min {bx, cx};
mov ax, bx
cmp ax, cx
jng ok
mov ax, cx
ok: return0
ax bx cx SF, ZF
5 5 3 >
3
Computers (96)

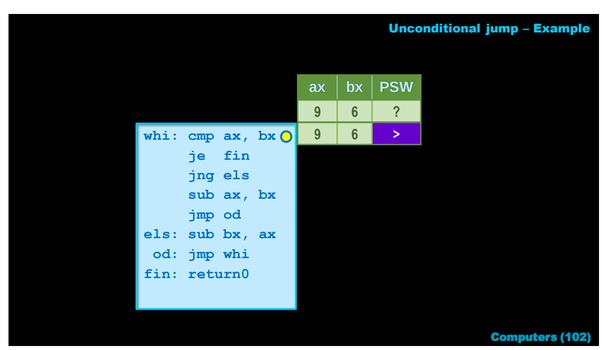

Conditional jumps
C
jump if equal JE e
jump if not equal JNE e
jump if not less JNL e
jump if greater JG e
jump if less JL e
if (c != z) ...
if (c == z) ...
if (c < z) ...
if (c <= z) ...
if (c >= z) ...
Computers (97)


Jump instructions
jump if equal JE e
JNE e jump if not equal
jump if greater JG e
JNG e jump if not greater
jump if less JL e
JNL e jump if not less
JMP e jump
Computers (98)


Unconditional jump
C
beg:
CMP c, z
JNE fini
...
jump
JMP beg
fini:
while (c == z)
{
...
}
c = z
No
Yes
...
Computers (99)


Unconditional jump
C
CMP c, z
JNG els
ins1
JMP fini
els: ins2
fini:
if (c > z)
{
ins1
} else {
ins2
}
c = z
Yes
No
ins1
ins2
Computers (100)


Unconditional jump – Example
ax bx PSW
9 6 ?
whi: cmp ax, bx
je fin
jng els
sub ax, bx
jmp od
els: sub bx, ax
od: jmp whi
fin: return0
Computers (101)


Unconditional jump – Example
ax bx PSW
9 6 ?
9 6 >
whi: cmp ax, bx
je fin
jng els
sub ax, bx
jmp od
els: sub bx, ax
od: jmp whi
fin: return0
Computers (102)

Unconditional jump – Example

ax	bx	PSW
9	6	?
9	6	>
9	6	>

```
whi: cmp ax, bx
     je  fin
     jng els
     sub ax, bx
     jmp od
els: sub bx, ax
 od: jmp whi
fin: return0
```

Computers (103)

Unconditional jump – Example

ax	bx	PSW
9	6	?
9	6	>
9	6	>
9	6	>

```
whi: cmp ax, bx
     je  fin
     jng els
     sub ax, bx
     jmp od
els: sub bx, ax
 od: jmp whi
fin: return0
```

Computers (104)

Unconditional jump – Example

ax	bx	PSW
9	6	?
9	6	>
9	6	>
9	6	>
3	6	<

```
whi: cmp ax, bx
     je  fin
     jng els
     sub ax, bx
     jmp od
els: sub bx, ax
 od: jmp whi
fin: return0
```

Computers (105)

Unconditional jump – Example

ax	bx	PSW
9	6	?
9	6	>
9	6	>
9	6	>
3	6	<
3	6	<

```
whi: cmp ax, bx
     je  fin
     jng els
     sub ax, bx
     jmp od
els: sub bx, ax
 od: jmp whi
fin: return0
```

Computers (106)

Unconditional jump – Example

ax	bx	PSW
9	6	?
9	6	>
9	6	>
9	6	>
3	6	<
3	6	<
3	6	<

```
whi: cmp ax, bx
     je  fin
     jng els
     sub ax, bx
     jmp od
els: sub bx, ax
 od: jmp whi
fin: return0
```

Computers (107)

Unconditional jump – Example

ax	bx	PSW
3	6	<

```
whi: cmp ax, bx
     je  fin
     jng els
     sub ax, bx
     jmp od
els: sub bx, ax
 od: jmp whi
fin: return0
```

Computers (108)

Unconditional jump – Example

ax	bx	PSW
3	6	<
3	6	<

```
whi: cmp ax, bx
     je  fin
     jng els
     sub ax, bx
     jmp od
els: sub bx, ax
 od: jmp whi
fin: return0
```

Computers (109)

Unconditional jump – Example

ax	bx	PSW
3	6	<
3	6	<
3	6	<

```
whi: cmp ax, bx
     je  fin
     jng els
     sub ax, bx
     jmp od
els: sub bx, ax
 od: jmp whi
fin: return0
```

Computers (110)

Unconditional jump – Example

ax	bx	PSW
3	6	<
3	6	<
3	6	<
3	6	<

```
whi: cmp ax, bx
     je  fin
     jng els
     sub ax, bx
     jmp od
els: sub bx, ax
 od: jmp whi
fin: return0
```

Computers (111)

Unconditional jump – Example

ax	bx	PSW
3	6	<
3	6	<
3	6	<
3	6	<

```
whi: cmp ax, bx
     je  fin
     jng els
     sub ax, bx
     jmp od
els: sub bx, ax
 od: jmp whi
fin: return0
```

Computers (112)

Unconditional jump – Example

ax	bx	PSW
3	6	<
3	6	<
3	6	<
3	6	<
3	3	=

```
whi: cmp ax, bx
     je  fin
     jng els
     sub ax, bx
     jmp od
els: sub bx, ax
 od: jmp whi
fin: return0
```

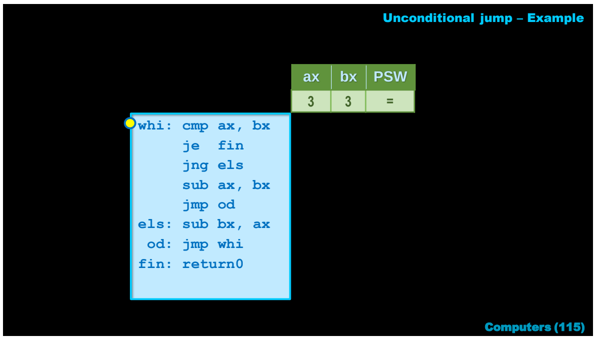
Computers (113)

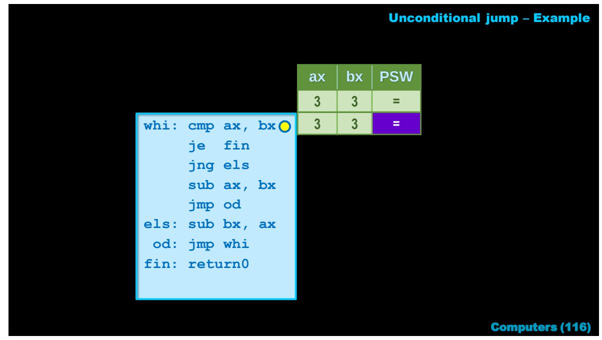
Unconditional jump – Example

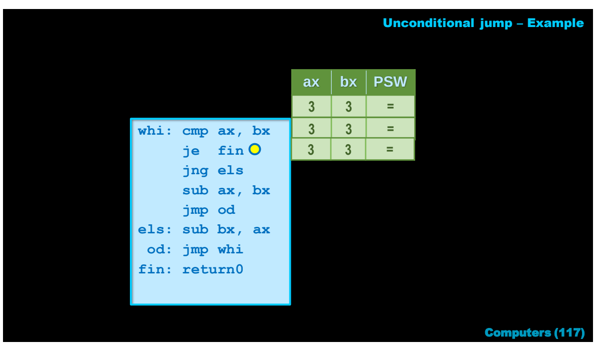
ax	bx	PSW
3	6	<
3	6	<
3	6	<
3	6	<
3	3	=
3	3	=

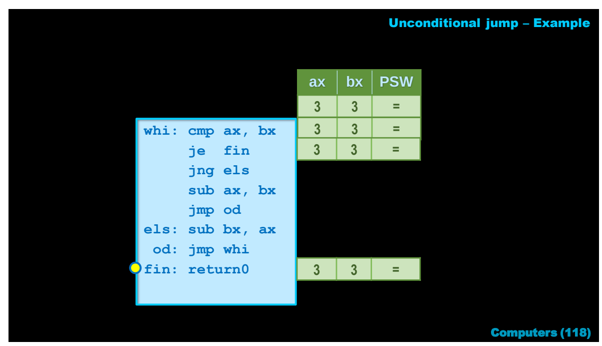
```
whi: cmp ax, bx
     je  fin
     jng els
     sub ax, bx
     jmp od
els: sub bx, ax
 od: jmp whi
fin: return0
```

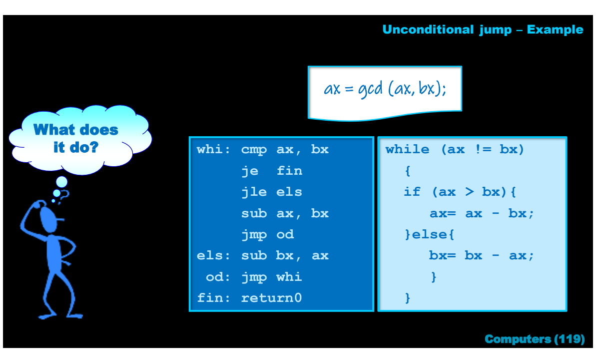
Computers (114)

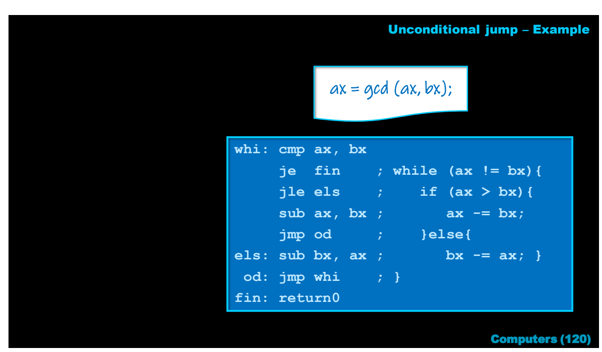

Unconditional jump – Example
ax bx PSW
3 3 =
whi: cmp ax, bx
je fin
jng els
sub ax, bx
jmp od
els: sub bx, ax
od: jmp whi
fin: return0
Computers (115)

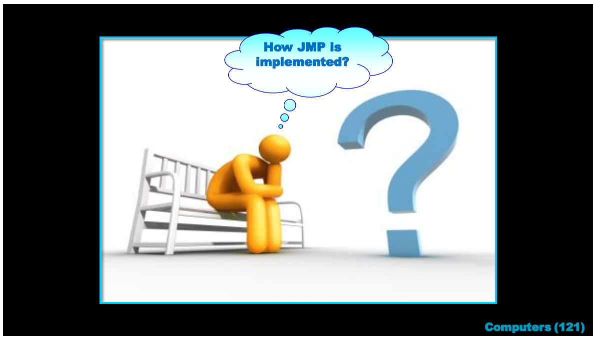

Unconditional jump – Example
ax bx PSW
3 3 =
3 3 =
whi: cmp ax, bx
je fin
jng els
sub ax, bx
jmp od
els: sub bx, ax
od: jmp whi
fin: return0
Computers (116)

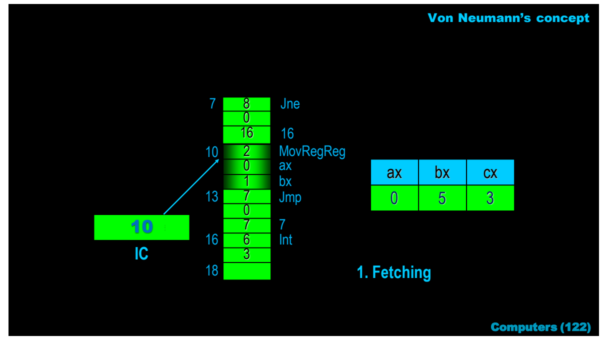

Unconditional jump – Example
ax bx PSW
3 3 =
3 3 =
3 3 =
whi: cmp ax, bx
je fin
jng els
sub ax, bx
jmp od
els: sub bx, ax
od: jmp whi
fin: return0
Computers (117)

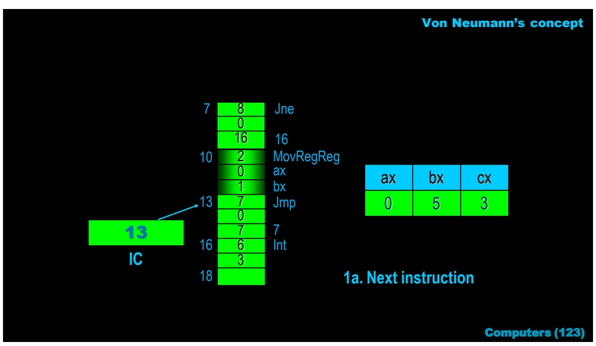

Unconditional jump – Example
ax bx PSW
3 3 =
3 3 =
3 3 =
whi: cmp ax, bx
je fin
jng els
sub ax, bx
jmp od
els: sub bx, ax
od: jmp whi
fin: return0
3 3 =
Computers (118)

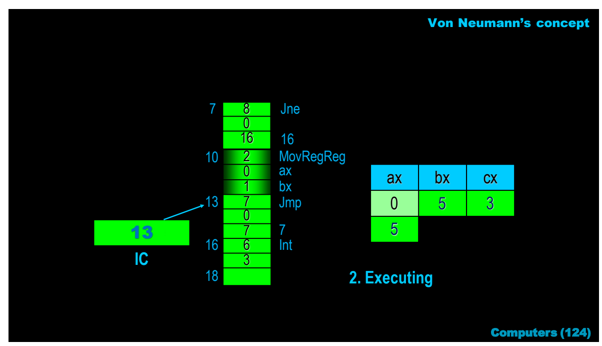

Unconditional jump – Example
ax = gcd (ax, bx);
What does it do?
whi: cmp ax, bx
je fin
jle els
sub ax, bx
jmp od
els: sub bx, ax
od: jmp whi
fin: return0
while (ax != bx)
{
if (ax > bx){
ax= ax - bx;
}else{
bx= bx - ax;
}
}
Computers (119)

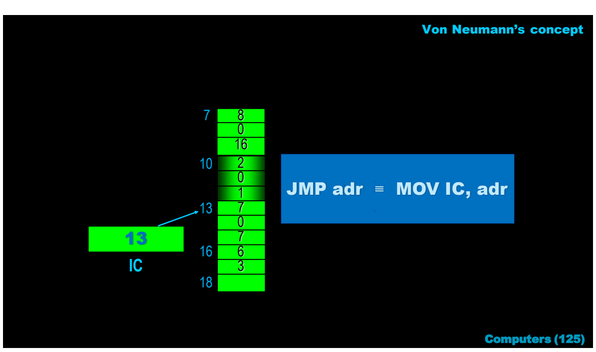

Unconditional jump – Example
ax = gcd (ax, bx);
whi: cmp ax, bx
je fin ; while (ax != bx){
jle els ; if (ax > bx){
sub ax, bx ; ax -= bx;
jmp od ; }else{
els: sub bx, ax ; bx -= ax; }
od: jmp whi ; }
fin: return0
Computers (120)

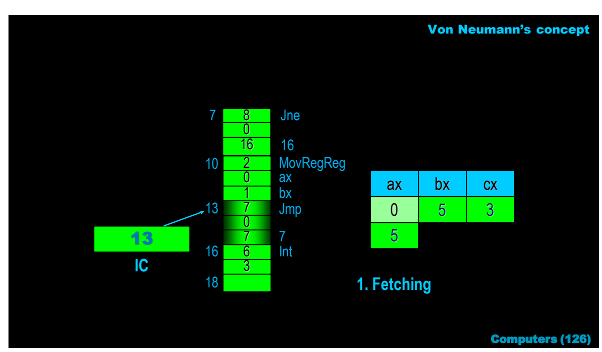

How JMP is implemented?
Computers (121)

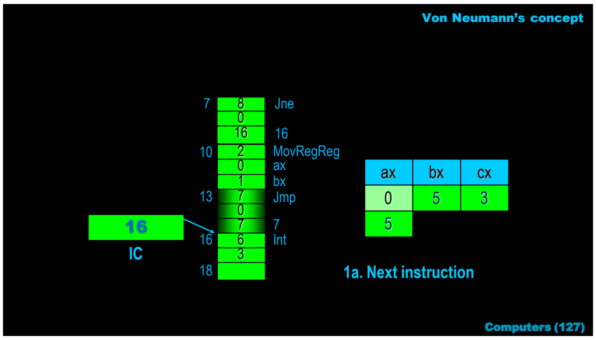

Von Neumann's concept
7 8 Jne
0
16 16
10 2 MovRegReg
0 ax
1 bx
13 7 Jmp
0
7 7
16 6 Int
3
18
10
IC
ax bx cx
0 5 3
1. Fetching
Computers (122)

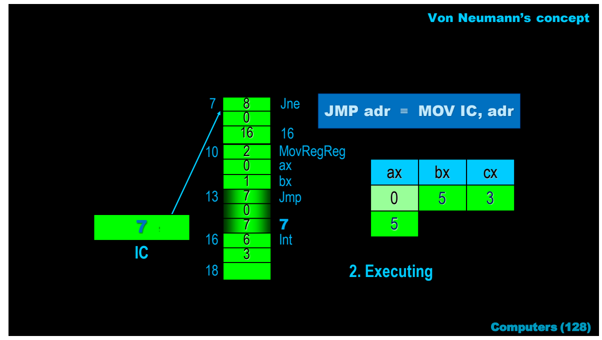

Von Neumann's concept
7 8 Jne
0
16 16
10 2 MovRegReg
0 ax
1 bx
13 7 Jmp
0
7 7
16 6 Int
3
18
13
IC
ax bx cx
0 5 3
1a. Next instruction
Computers (123)

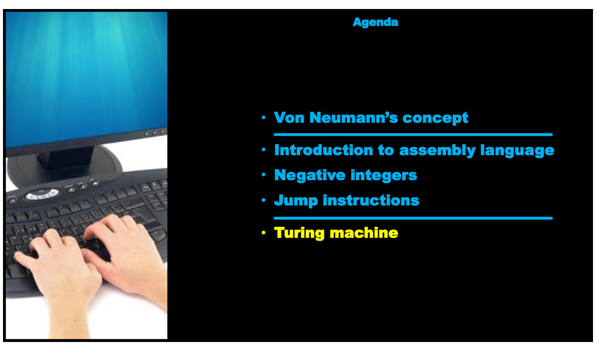

Von Neumann's concept
7 8 Jne
0
16 16
10 2 MovRegReg
0 ax
1 bx
13 7 Jmp
0
7 7
16 6 Int
3
18
13
IC
ax bx cx
0 5 3
5
2. Executing
Computers (124)

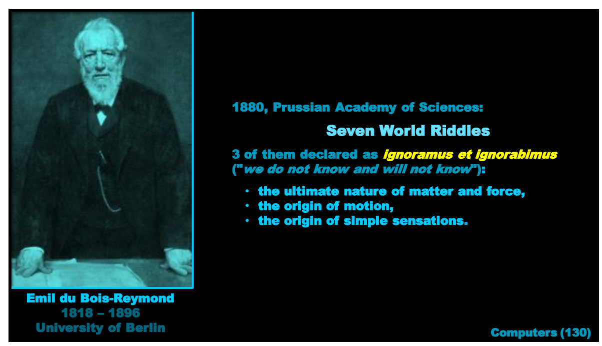

Von Neumann's concept
7 8
0
16
10 2
0
1
13 7
0
7
16 6
3
18
13
IC
JMP adr ≡ MOV IC, adr
Computers (125)

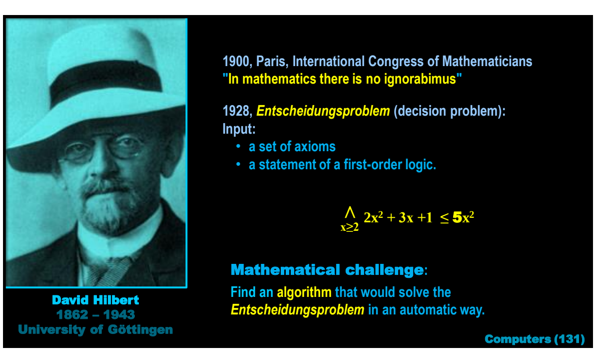

Von Neumann's concept
7 8 Jne
0
16 16
10 2 MovRegReg
0 ax
1 bx
13 7 Jmp
0
7 7
16 6 Int
3
18
13
IC
ax bx cx
0 5 3
5
1. Fetching
Computers (126)

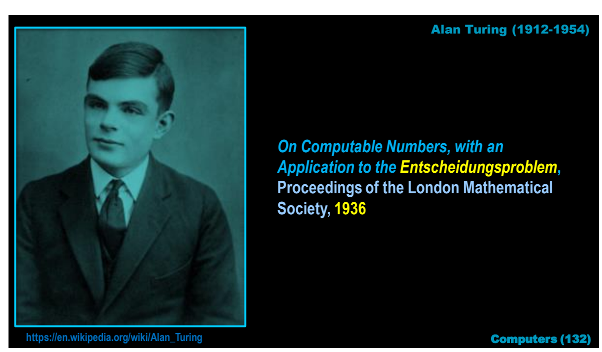

Von Neumann's concept
7
8
0
16
10
2
0
1
13
7
0
7
16
6
3
18
Jne
16
MovRegReg
ax
bx
Jmp
7
Int
16
IC
ax
bx
cx
0
5
3
5
1a. Next instruction
Computers (127)

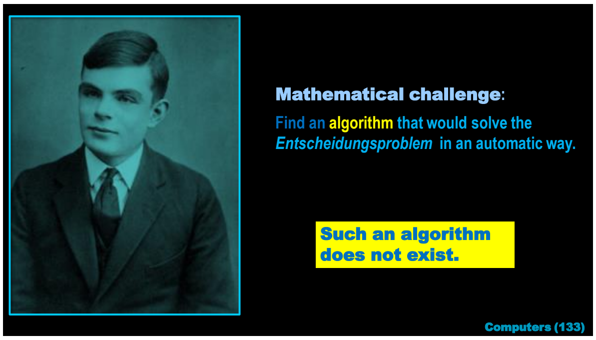

Von Neumann's concept
JMP adr = MOV IC, adr
7
8
0
16
10
2
0
1
13
7
0
7
16
6
3
18
Jne
16
MovRegReg
ax
bx
Jmp
7
Int
7
IC
ax
bx
cx
0
5
3
5
2. Executing
Computers (128)

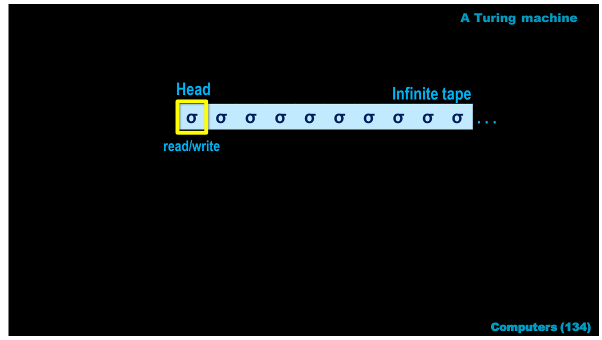

Agenda
Von Neumann's concept
Introduction to assembly language
Negative integers
Jump instructions
Turing machine

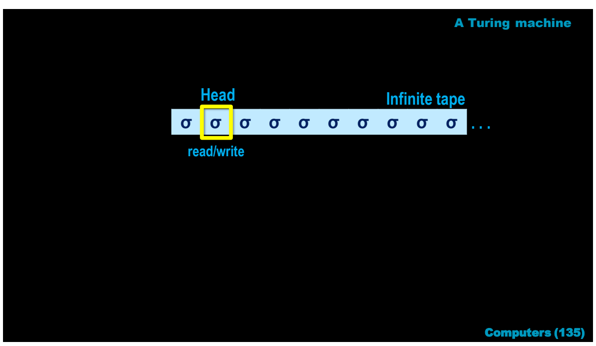

1880, Prussian Academy of Sciences:
Seven World Riddles
3 of them declared as Ignoramus et ignorabimus
("we do not know and will not know"):
the ultimate nature of matter and force,
the origin of motion,
the origin of simple sensations.
Emil du Bois-Reymond
1818 – 1896
University of Berlin
Computers (130)

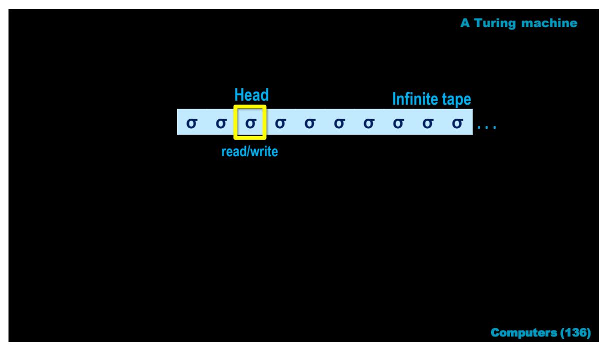

1900, Paris, International Congress of Mathematicians
"In mathematics there is no ignorabimus"
1928, Entscheidungsproblem (decision problem):
Input:
a set of axioms
a statement of a first-order logic.
$\bigwedge_{x\geq 2} 2x^2 + 3x + 1 \leq 5x^2$
Mathematical challenge:
Find an algorithm that would solve the Entscheidungsproblem in an automatic way.
David Hilbert
1862 – 1943
University of Göttingen
Computers (131)

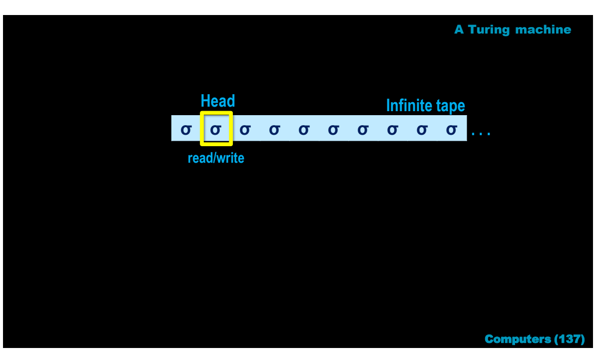

Alan Turing (1912-1954)
On Computable Numbers, with an Application to the Entscheidungsproblem, Proceedings of the London Mathematical Society, 1936
https://en.wikipedia.org/wiki/Alan_Turing
Computers (132)

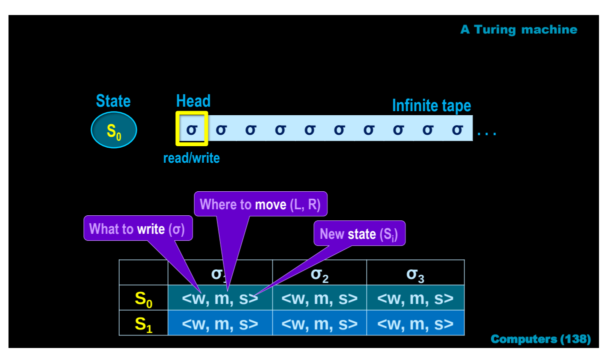

Mathematical challenge:
Find an algorithm that would solve the *Entscheidungsproblem* in an automatic way.
Such an algorithm does not exist.
Computers (133)

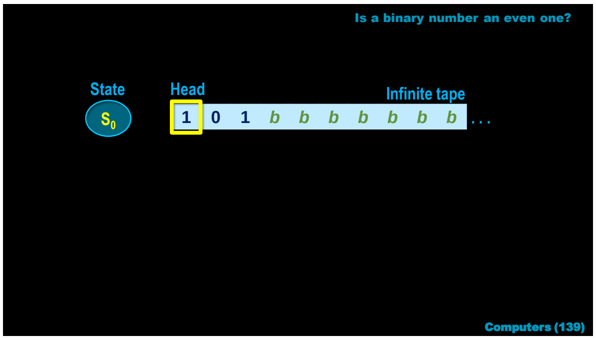

A Turing machine
Head
Infinite tape
σ σ σ σ σ σ σ σ σ σ . . .
read/write
Computers (134)

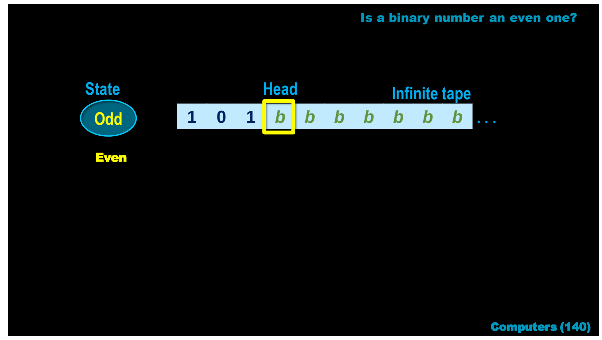

A Turing machine
Head
Infinite tape
σ σ σ σ σ σ σ σ σ σ . . .
read/write
Computers (135)

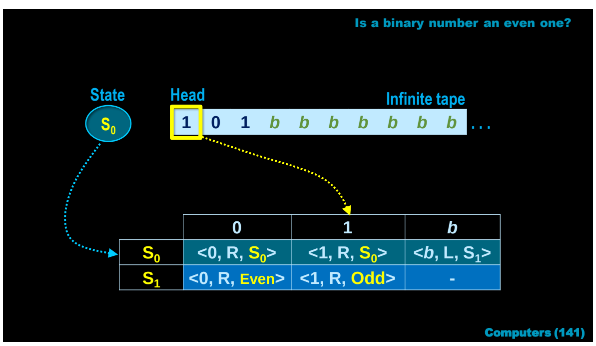

A Turing machine
Head
Infinite tape
σ σ σ σ σ σ σ σ σ σ . . .
read/write
Computers (136)

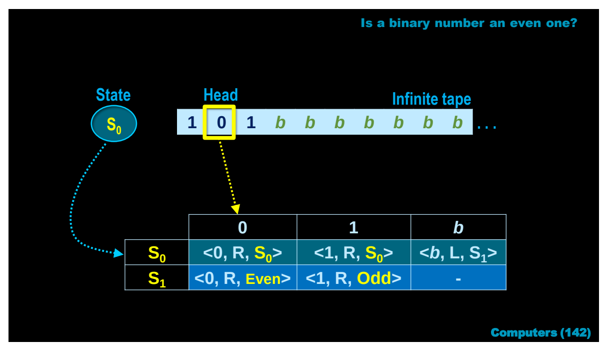

A Turing machine
Head
Infinite tape
σ σ σ σ σ σ σ σ σ σ . . .
read/write
Computers (137)

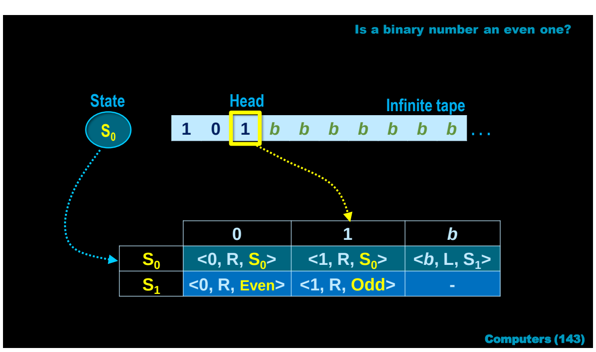

A Turing machine
State
S_0
Head
Infinite tape
σ σ σ σ σ σ σ σ σ σ . . .
read/write
What to write (σ)
Where to move (L, R)
New state (S_j)
σ_1 σ_2 σ_3
S_0 <w, m, s> <w, m, s> <w, m, s>
S_1 <w, m, s> <w, m, s> <w, m, s>
Computers (138)

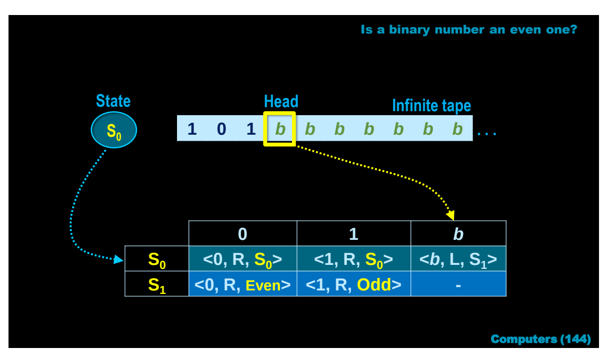

Is a binary number an even one?
State
S_0
Head
Infinite tape
1 0 1 b b b b b b b ...
Computers (139)

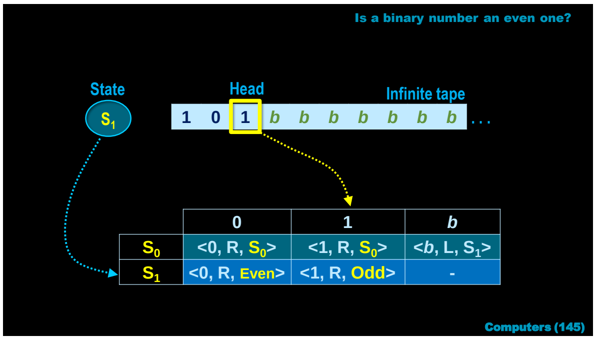

Is a binary number an even one?
State
Odd
Even
Head
Infinite tape
1 0 1 b b b b b b b ...
Computers (140)

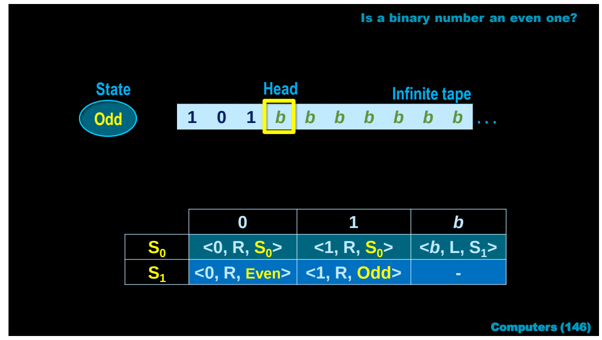

Is a binary number an even one?
State
S_0
Head
Infinite tape
1 0 1 b b b b b b b ...
0 | 1 | b
S_0 | <0, R, S_0> | <1, R, S_0> | <b, L, S_1>
S_1 | <0, R, Even> | <1, R, Odd> | -
Computers (141)

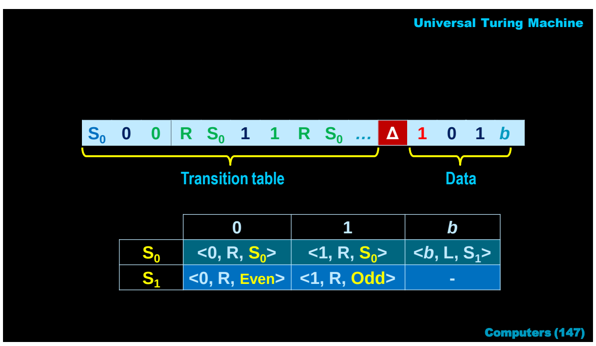

Is a binary number an even one?
State
S_0
Head
Infinite tape
1 0 1 b b b b b b b ...
0 | 1 | b
S_0 | <0, R, S_0> | <1, R, S_0> | <b, L, S_1>
S_1 | <0, R, Even> | <1, R, Odd> | -
Computers (142)

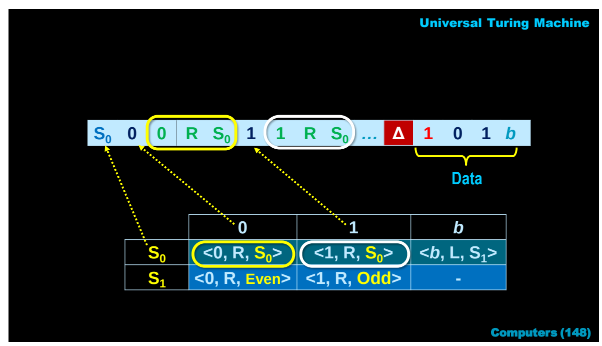

Is a binary number an even one?
State
S_0
Head
Infinite tape
1 0 1 b b b b b b b ...
0 | 1 | b
S_0 | <0, R, S_0> | <1, R, S_0> | <b, L, S_1>
S_1 | <0, R, Even> | <1, R, Odd> | -
Computers (143)

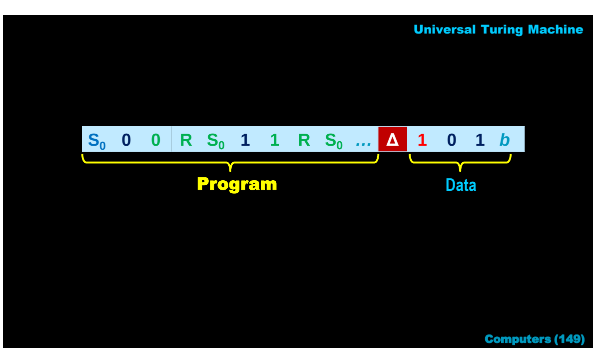

Is a binary number an even one?
State
S_0
Head
Infinite tape
1 0 1 b b b b b b b ...
0 | 1 | b
S_0 | <0, R, S_0> | <1, R, S_0> | <b, L, S_1>
S_1 | <0, R, Even> | <1, R, Odd> | -
Computers (144)

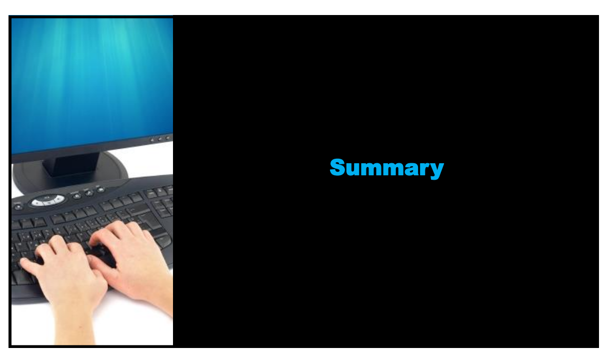

Is a binary number an even one?
State
S_1
Head
Infinite tape
1 0 1 b b b b b b b . . .
	0	1	b
S_0	<0, R, S_0>	<1, R, S_0>	<b, L, S_1>
S_1	<0, R, Even>	<1, R, Odd>	-
Computers (145)

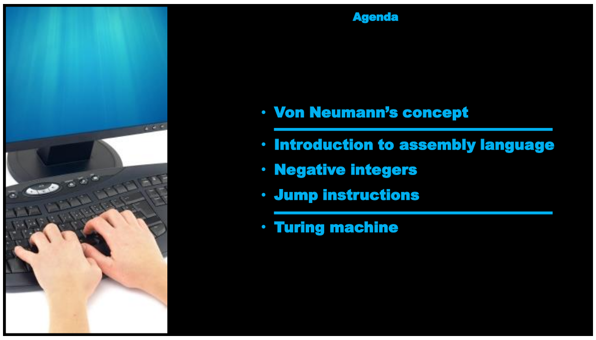

Is a binary number an even one?
State
Odd
Head
Infinite tape
1 0 1 b b b b b b b . . .
	0	1	b
S_0	<0, R, S_0>	<1, R, S_0>	<b, L, S_1>
S_1	<0, R, Even>	<1, R, Odd>	-
Computers (146)

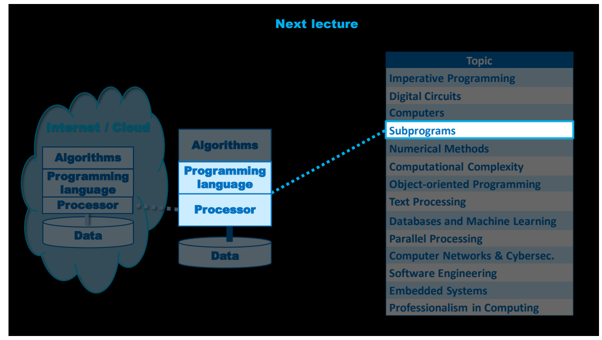

Universal Turing Machine
S_0 0 0 R S_0 1 1 R S_0 … Δ 1 0 1 b
Transition table
Data
	0	1	b
S_0	<0, R, S_0>	<1, R, S_0>	<b, L, S_1>
S_1	<0, R, Even>	<1, R, Odd>	-
Computers (147)


Universal Turing Machine
S_0 0 0 R S_0 1 1 R S_0 … Δ 1 0 1 b
Data
	0	1	b
S_0	<0, R, S_0>	<1, R, S_0>	<b, L, S_1>
S_1	<0, R, Even>	<1, R, Odd>	-
Computers (148)


Universal Turing Machine
S_0 0 0 R S_0 1 1 R S_0 … Δ 1 0 1 b
Program
Data
Computers (149)


Summary


Agenda
• Von Neumann's concept
• Introduction to assembly language
• Negative integers
• Jump instructions
• Turing machine


Next lecture
Internet / Cloud
Algorithms
Programming language
Processor
Data
Algorithms
Programming language
Processor
Data
Topic
Imperative Programming
Digital Circuits
Computers
Subprograms
Numerical Methods
Computational Complexity
Object-oriented Programming
Text Processing
Databases and Machine Learning
Parallel Processing
Computer Networks & Cybersec.
Software Engineering
Embedded Systems
Professionalism in Computing