Principles of Computers
7th Lecture

http://d3s.mff.cuni.cz/~jezek

Department of Distributed and Dependable Systems

D3S

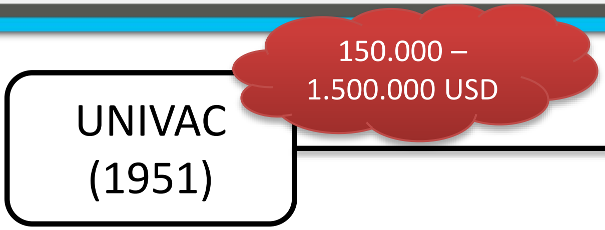
Pavel Ježek, Ph.D.

pavel.jezek@d3s.mff.cuni.cz

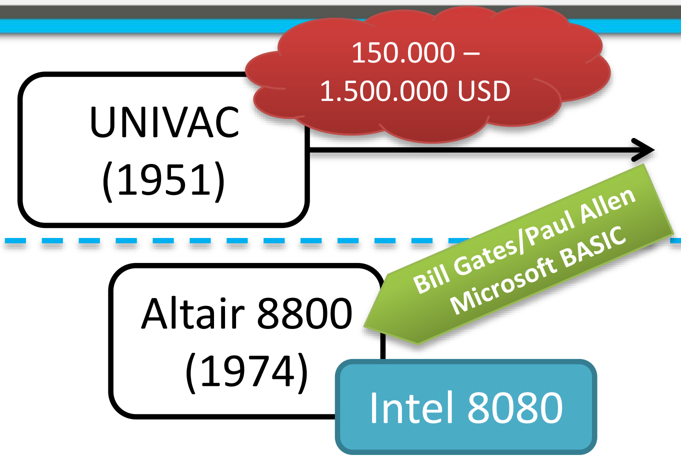
CHARLES UNIVERSITY IN PRAGUE

faculty of mathematics and physics

History

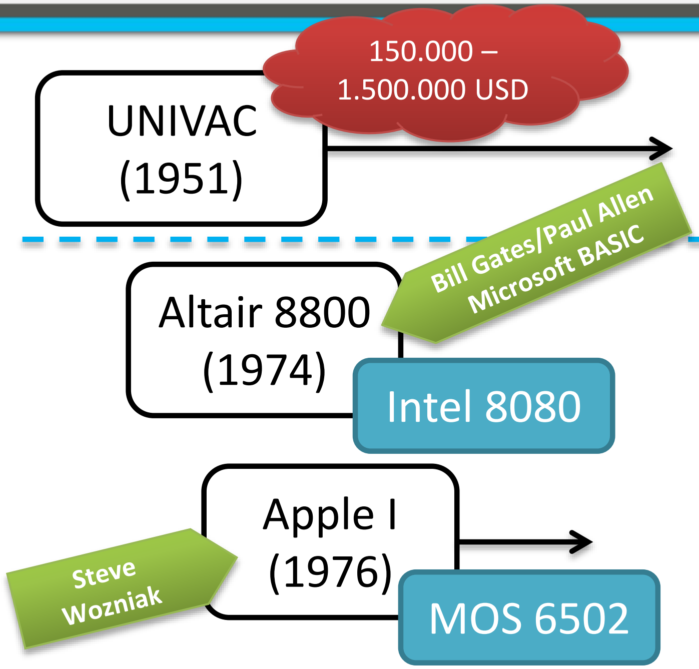


History

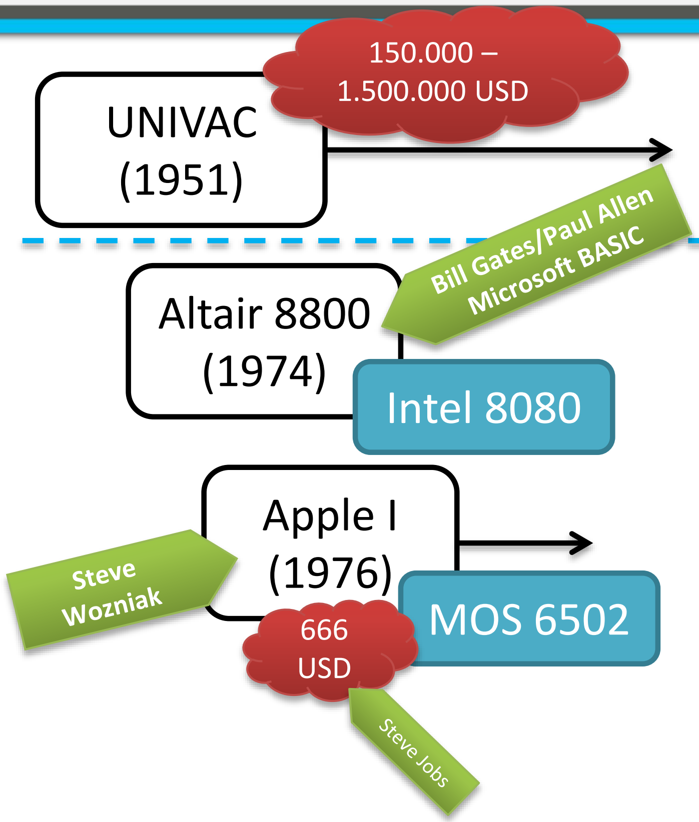


UNIVAC (1951)
150.000 – 1.500.000 USD
Altair 8800 (1974)
Bill Gates/Paul Allen Microsoft BASIC
Intel 8080

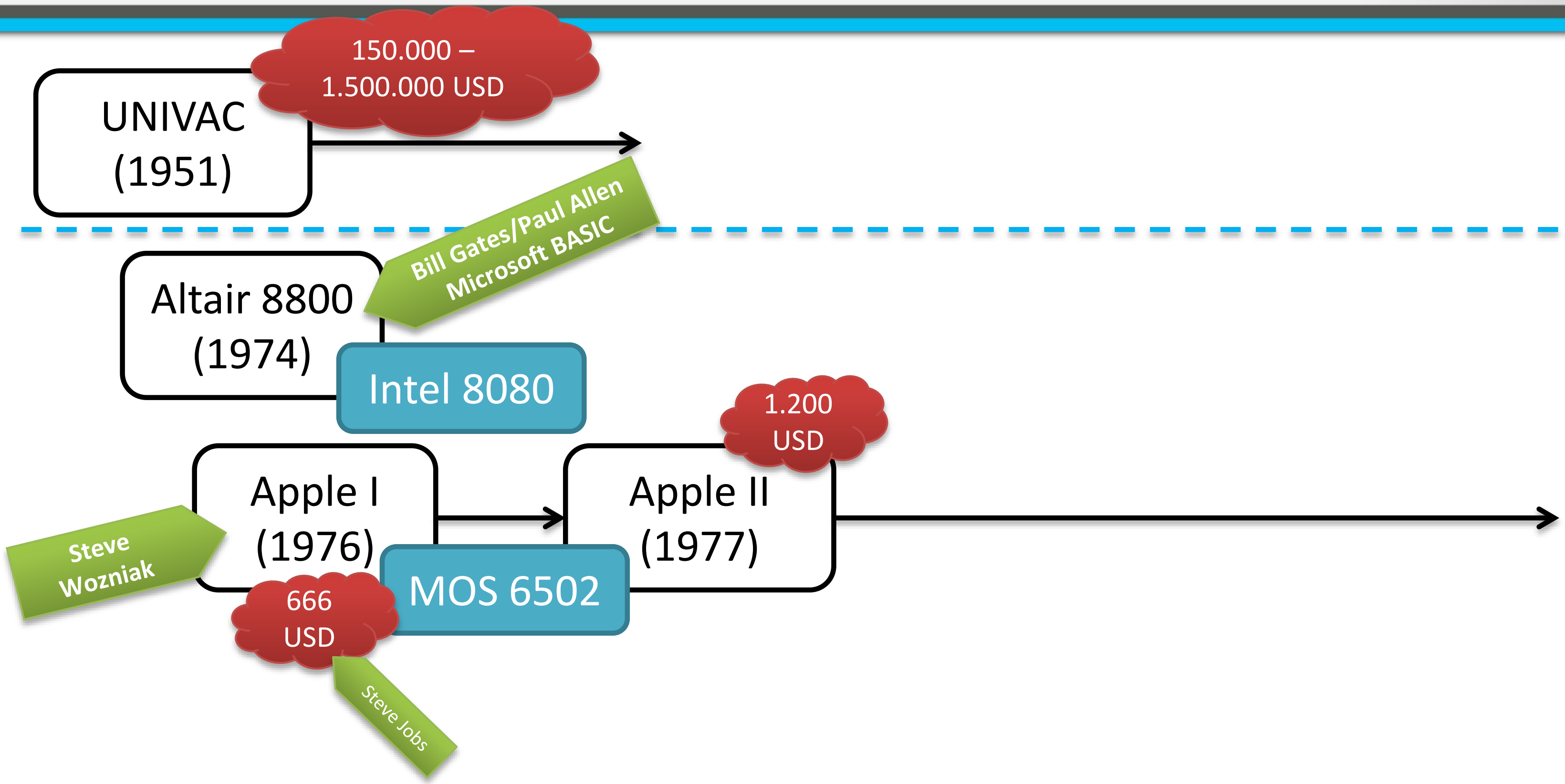
History


150.000 –
1.500.000 USD
UNIVAC (1951)
Bill Gates/Paul Allen
Microsoft BASIC
Altair 8800 (1974)
Intel 8080
Apple I (1976)
Steve Wozniak
MOS 6502

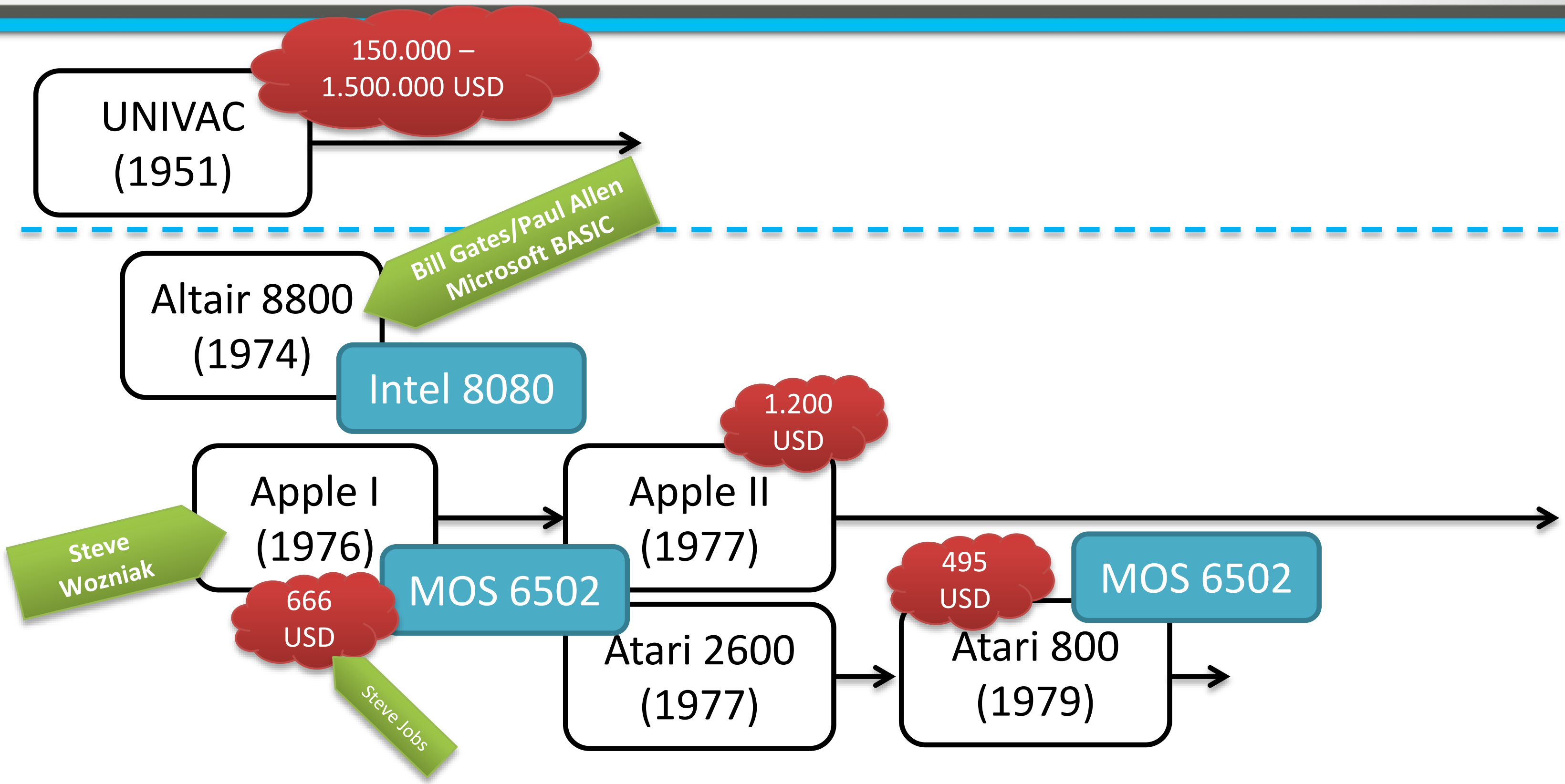
History


UNIVAC (1951)
150.000 – 1.500.000 USD
Bill Gates/Paul Allen Microsoft BASIC
Altair 8800 (1974)
Intel 8080
Apple I (1976)
Steve Wozniak
666 USD
MOS 6502
Steve Jobs

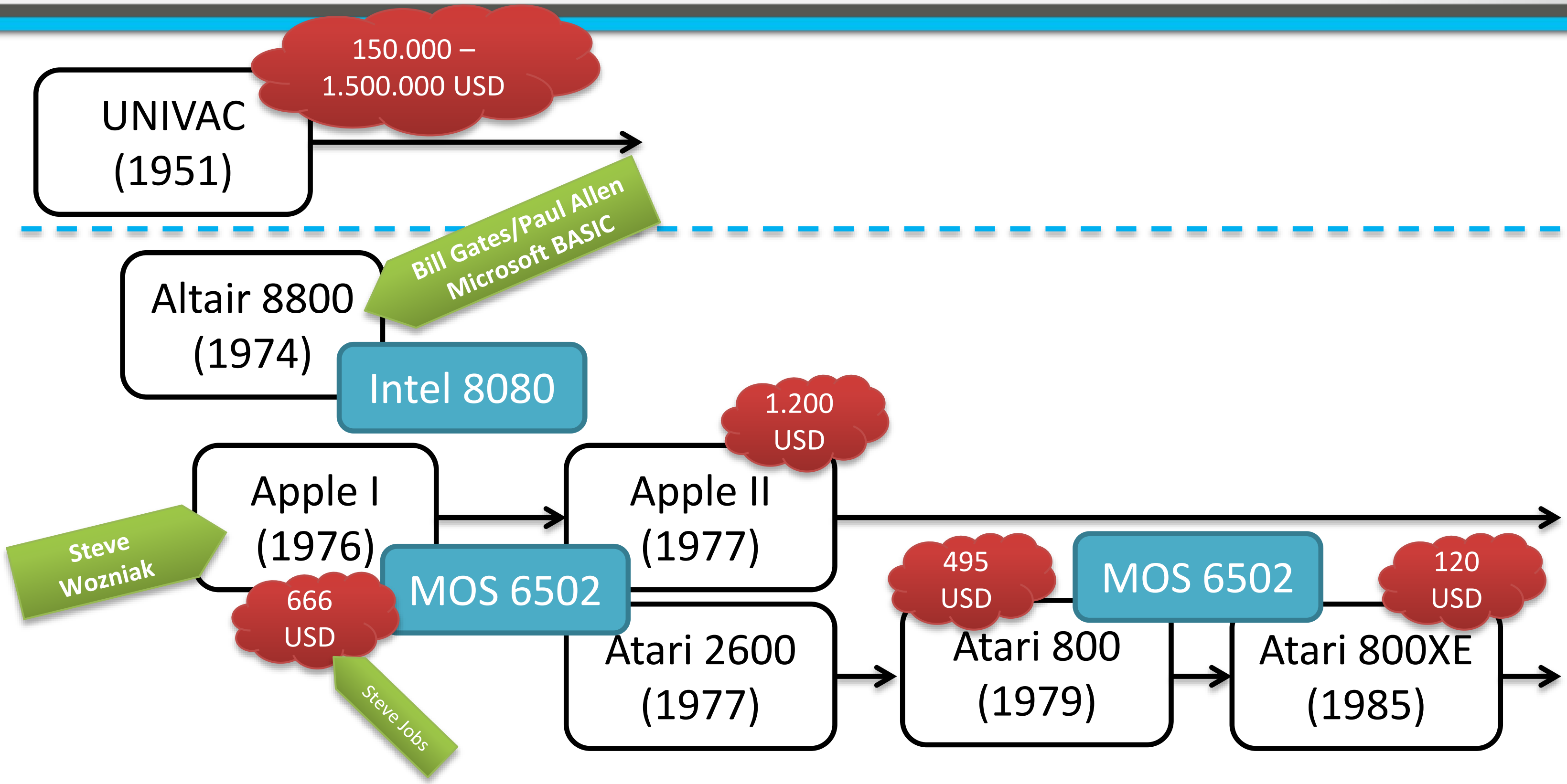
History


UNIVAC (1951)
150.000 – 1.500.000 USD
Altair 8800 (1974)
Bill Gates/Paul Allen Microsoft BASIC
Intel 8080
Apple I (1976)
Steve Wozniak
666 USD
Steve Jobs
MOS 6502
Apple II (1977)
1.200 USD

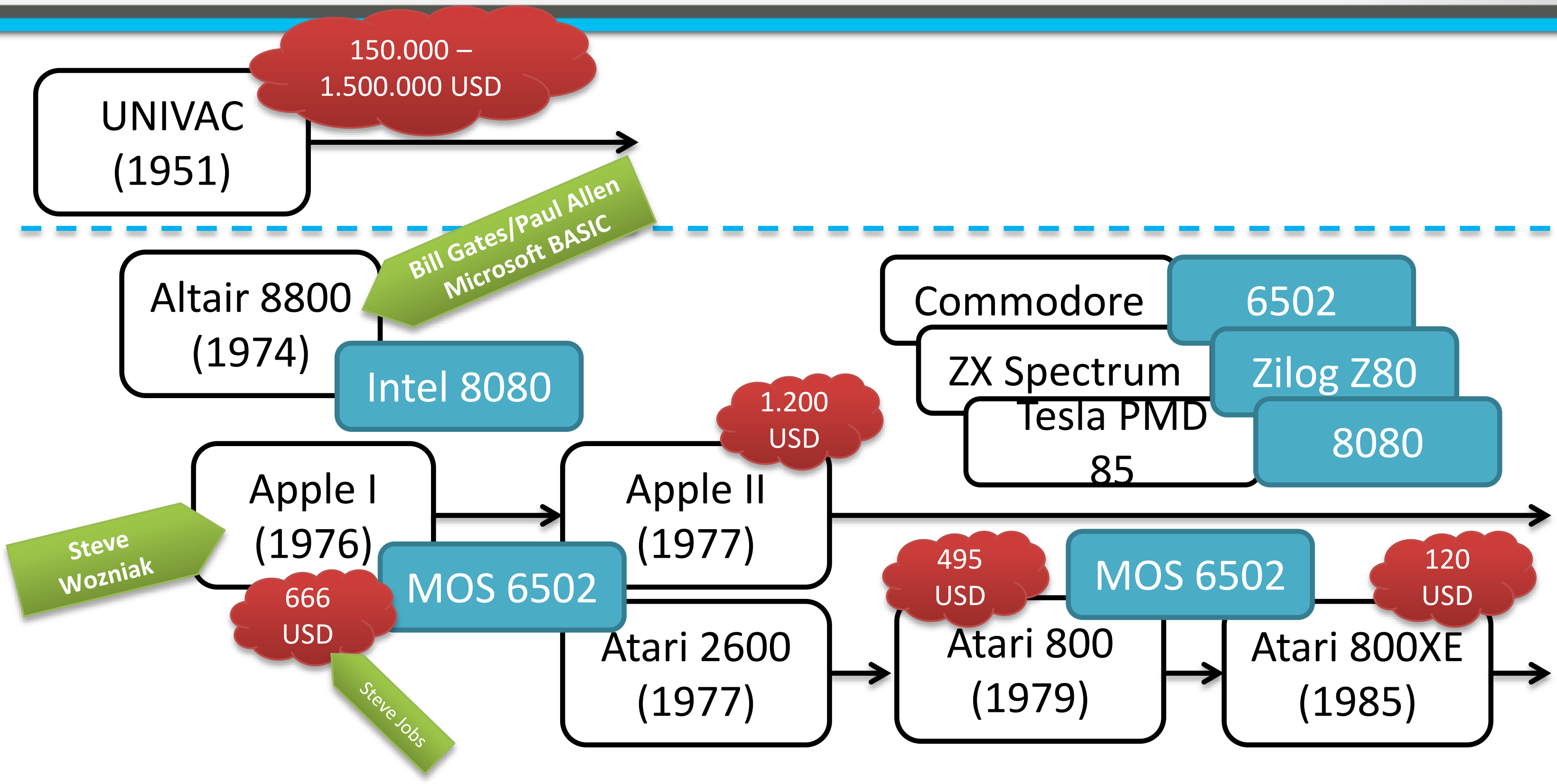
History


UNIVAC (1951)
150.000 – 1.500.000 USD
Altair 8800 (1974)
Bill Gates/Paul Allen Microsoft BASIC
Intel 8080
Apple I (1976)
Steve Wozniak
666 USD
Steve Jobs
MOS 6502
Apple II (1977)
1.200 USD
Atari 2600 (1977)
Atari 800 (1979)
495 USD
MOS 6502

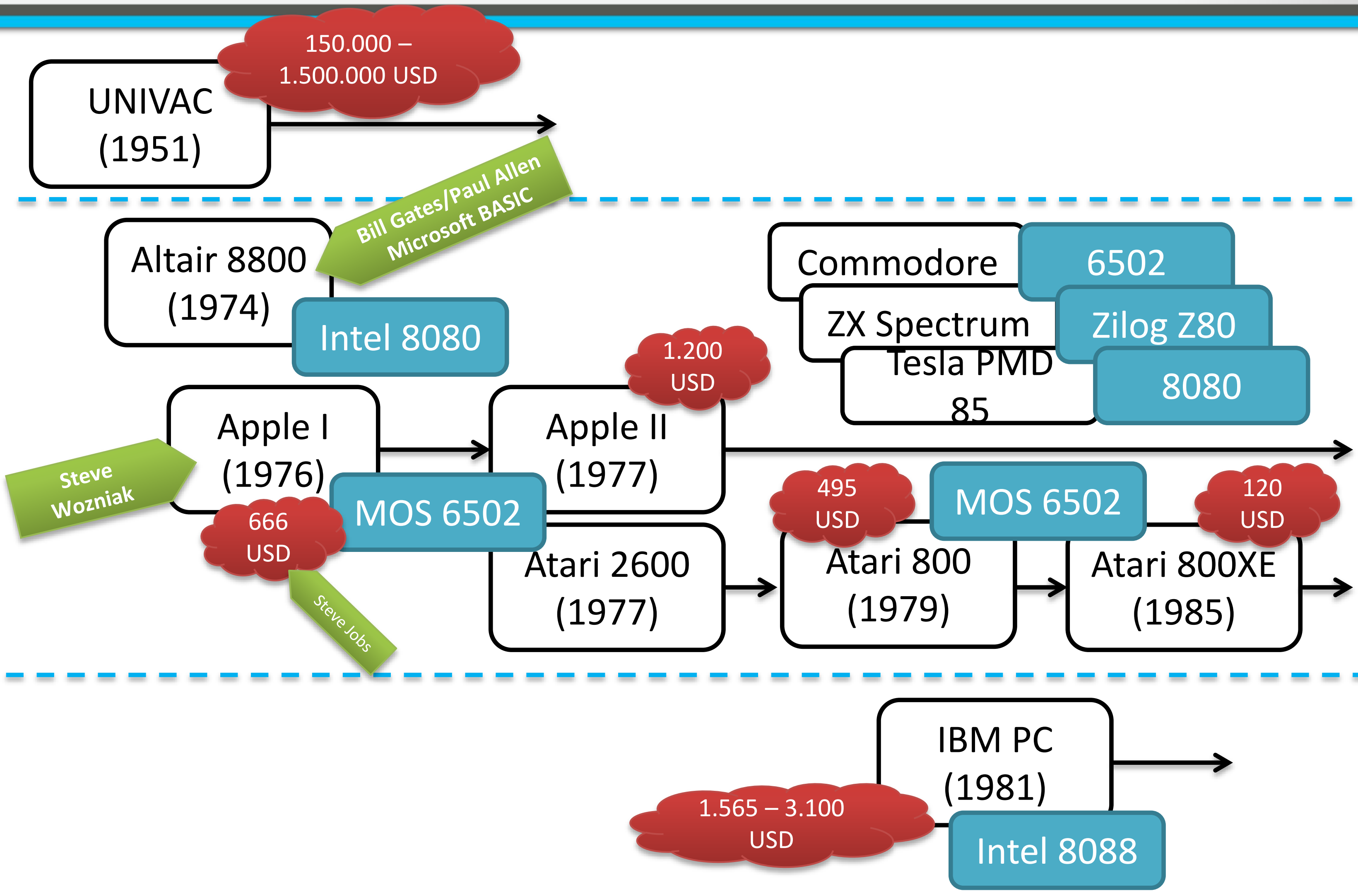
History


UNIVAC (1951)
150.000 – 1.500.000 USD
Altair 8800 (1974)
Bill Gates/Paul Allen Microsoft BASIC
Intel 8080
Apple I (1976)
Steve Wozniak
666 USD
Steve Jobs
MOS 6502
Apple II (1977)
1.200 USD
Atari 2600 (1977)
Atari 800 (1979)
495 USD
MOS 6502
Atari 800XE (1985)
120 USD

History

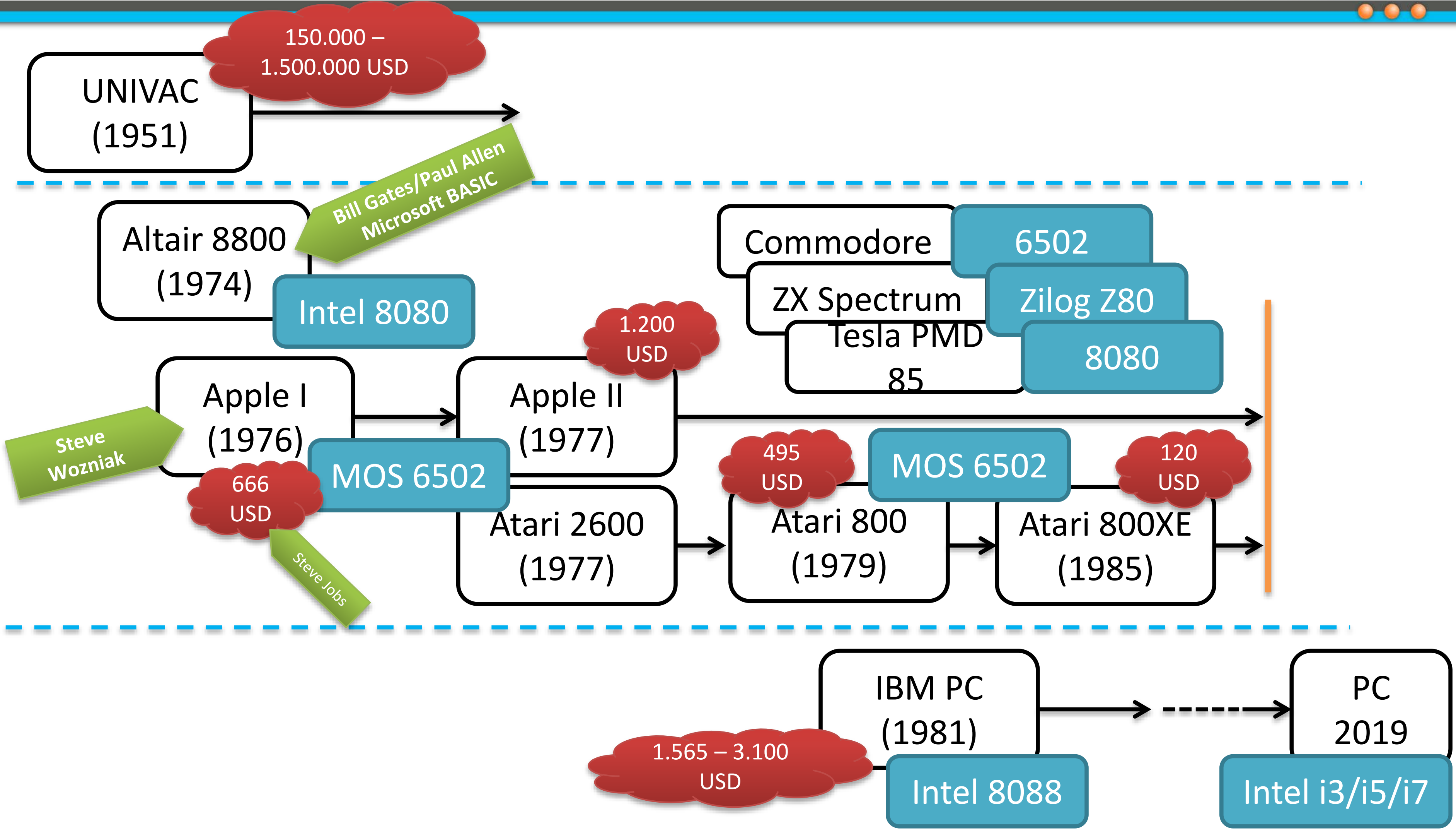

UNIVAC (1951)
150.000 – 1.500.000 USD
Bill Gates/Paul Allen Microsoft BASIC
Altair 8800 (1974)
Intel 8080
Commodore
6502
ZX Spectrum
Zilog Z80
Tesla PMD 85
8080
1.200 USD
Apple I (1976)
Apple II (1977)
Steve Wozniak
666 USD
MOS 6502
Steve Jobs
Atari 2600 (1977)
495 USD
Atari 800 (1979)
MOS 6502
120 USD
Atari 800XE (1985)

History



Department of Distributed and Dependable Systems

History



Department of Distributed and Dependable Systems

Basic Instructions (6502 vs. x86)

6502 machine code	Intel x86 (IA-32) machine code	Comment
		← Offset from instruction’s start (base) address ← Actual bytes of instruction’s machine code

Basic Instructions (6502 vs. x86)

6502 machine code	Intel x86 (IA-32) machine code	Comment

`0`
`$EA`
	`0`
	`$90`
		← Offset from instruction's start (base) address
		← Actual bytes of instruction's machine code
		
		No operation (just do nothing)

Basic Instructions (6502 vs. x86)

6502 machine code	Intel x86 (IA-32) machine code	Comment
`0`	`0`	← Offset from instruction's start (base) address
`$EA`	`$90`	← Actual bytes of instruction's machine code
`PC := PC + 1`	`EIP := EIP + 1`	*No operation (just do nothing **and continue to next instruction**)*

Instruction size in bytes

Department of Distributed and Dependable Systems

Basic Instructions (6502 vs. x86)

6502 machine code	Intel x86 (IA-32) machine code	Comment

`0`
`$EA`
 	
`PC := PC + 1`
	`0`
	`$90`
	
	`EIP := EIP + 1`
		← Offset from instruction's start (base) address
		← Actual bytes of instruction's machine code
		
		No operation (just do nothing and continue to next instruction)

`0 1 2` **`$4C`** xx_0 xx_1 `PC :=` $\$xx_1xx_0$	`0 1 2 3 4` **`$E9`** xx_0 xx_1 xx_2 xx_3 `EIP :=` $\$xx_3xx_2xx_1xx_0$	← Offset from instruction's start (base) address ← Actual bytes of instruction's machine code *Direct jump to target address x*

Jump/branch instruction

Basic Instructions (6502 vs. x86)

6502 machine code	Intel x86 (IA-32) machine code	Comment

`0`
`$EA`
 	
`PC := PC + 1`
	`0`
	`$90`
	
	`EIP := EIP + 1`
		← Offset from instruction's start (base) address
		← Actual bytes of instruction's machine code
		
		No operation (just do nothing and continue to next instruction)

`0   1   2` `$4C` xx_0 xx_1 `PC :=` $\$xx_1xx_0$	`0   1   2   3   4` `$E9` xx_0 xx_1 xx_2 xx_3 `EIP :=` $\$xx_3xx_2xx_1xx_0$	← Offset from instruction's start (base) address ← Actual bytes of instruction's machine code *Direct jump to target address x*

16-bit PC → 2 byte argument

32-bit EIP → 4 byte argument

Jump/branch instruction

Basic Instructions (6502 vs. x86)

6502 machine code	Intel x86 (IA-32) machine code	Comment
0	0	← Offset from instruction's start (base) address
$EA	$90	← Actual bytes of instruction's machine code
`PC := PC + 1`	`EIP := EIP + 1`	*No operation (just do nothing and continue to next instruction)*
0 1 2	0 1 2 3 4	← Offset from instruction's start (base) address
$\$4C\ xx_0\ xx_1$	$\$E9\ xx_0\ xx_1\ xx_2\ xx_3$	← Actual bytes of instruction's machine code
$PC := \$xx_1xx_0$	$EIP := \$xx_3xx_2xx_1xx_0$	*Direct jump to target address x*

16-bit PC → 2 byte argument

32-bit EIP → 4 byte argument

6502 is **LE** CPU arch.

x86 is **LE** CPU arch.

Jump/branch instruction

Basic Instructions (6502 vs. x86)

6502 machine code	Intel x86 (IA-32) machine code	Comment

`0`
`$EA`
 	
`PC := PC + 1`
 	
6502 assembler:
NOP
	`0`
	`$90`
	
	`EIP := EIP + 1`
	
	Intel assembler:
	NOP
		← Offset from instruction's start (base) address
		← Actual bytes of instruction's machine code
		
		No operation (just do nothing and continue to next instruction)

`0 1 2` `$4C` xx_0 xx_1 PC := $\$xx_1xx_0$ *6502 assembler:* JMP $\$xx_1xx_0$	`0 1 2 3 4` `$E9` xx_0 xx_1 xx_2 xx_3 EIP := $\$xx_3xx_2xx_1xx_0$ *Intel assembler:* JMP $xx_3xx_2xx_1xx_0$h	← Offset from instruction's start (base) address ← Actual bytes of instruction's machine code *Direct jump to target address x*

Basic Instructions (6502 vs. x86)

6502 machine code	Intel x86 (IA-32) machine code	Comment
0 **$EA** PC := PC + 1 *6502 assembler:* **NOP**	0 **$90** EIP := EIP + 1 *Intel assembler:* **NOP**	← Offset from instruction's start (base) address ← Actual bytes of instruction's machine code *No operation (just do nothing and continue to next instruction)*
0 1 2 **$4C** xx_0 xx_1 PC := \$$xx_1xx_0$ *6502 assembler:* **JMP** \$$xx_1xx_0$	0 1 2 3 4 **$E9** xx_0 xx_1 xx_2 xx_3 EIP := \$$xx_3xx_2xx_1xx_0$ *Intel assembler:* **JMP** $xx_3xx_2xx_1xx_0$**h**	← Offset from instruction's start (base) address ← Actual bytes of instruction's machine code *Direct jump to target address x*

JMP 00000005h in assembler is 15 bytes in UTF-8 encoding including newline:

```
00000000: 4A 4D 50 20 30 30 30 30|30 30 30 35 68 0D 0A      |JMP 00000005h..
```

In machine code = 5 bytes:

E9 05 00 00 00

Basic Instructions (6502 vs. x86)

6502 machine code	Intel x86 (IA-32) machine code	Comment
0 **$EA** `PC := PC + 1` *6502 assembler:* **NOP**	0 **$90** `EIP := EIP + 1` *Intel assembler:* **NOP**	← Offset from instruction's start (base) address ← Actual bytes of instruction's machine code *No operation (just do nothing and continue to next instruction)*
0 1 2 **$4C** xx_0 xx_1 PC := $\$xx_1xx_0$ *6502 assembler:* **JMP** $\$xx_1xx_0$	0 1 2 3 4 **$E9** xx_0 xx_1 xx_2 xx_3 EIP := $\$xx_3xx_2xx_1xx_0$ *Intel assembler:* **JMP** $xx_3xx_2xx_1xx_0$**h**	← Offset from instruction's start (base) address ← Actual bytes of instruction's machine code *Direct jump to target address x*

JMP 00000005h in assembler is 15 bytes in UTF-8 encoding:

```
00000000: 4A 4D 50 20 30 30 30 30|30 30 30 35 68 0D 0A      | JMP 00000005h..
```

x86 assembler (compiler)

In machine code = 5 bytes:

E9 05 00 00 00

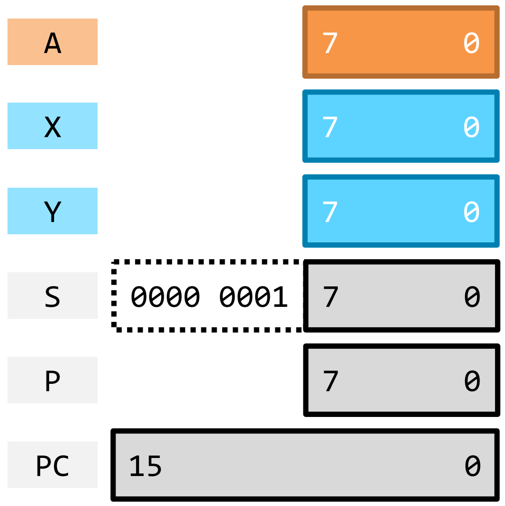
Department of Distributed and Dependable Systems

Typical ISA Arithmetic Instructions

MIPS: a := b *op* c

x86, 6502: a := a *op* b

6502 Registers (Accumulator Architecture)



6502: **8-bit CPU** with **16-bit** logical and physical **address spaces** (1:1 mapping between logical and physical addresses, i.e. logical address = physical address)

Load Instructions (6502)

6502 machine code	Comment
0 1 **\$A9** xx$_0$ A := \$xx$_0$ PC := PC + **2** *6502 assembler:* **LDA #\$**xx$_0$	← Offset from instruction’s start (base) address ← Actual bytes of instruction’s machine code *Load 8-bit constant \$xx$_0$ into A register.* *(Immediate load instruction)*

Load Instructions (6502)

6502 machine code	Comment
0 1 **$A9** xx_0 A := $\$xx_0$ PC := PC + **2** *6502 assembler:* **LDA #$**$xx_0$	← Offset from instruction's start (base) address ← Actual bytes of instruction's machine code *Load 8-bit constant* $\$xx_0$ *into A register.* *(Immediate load instruction)*
0 1 2 **$AD** xx_0 xx_1 A := MemReadByte($\$xx_1xx_0$) PC := PC + **3** *6502 assembler:* **LDA $**$xx_1xx_0$	← Offset from instruction's start (base) address ← Actual bytes of instruction's machine code *Read 8-bit value from (16-bit) address* $\$xx_1xx_0$ *and load the 8-bit value into A register.* *(Load from absolute address)*

Load Instructions (6502)

6502 machine code (LDA instruction variants)	Comment
0 1 $\$A9\ xx_0$ $A := \$xx_0$ $LDA\ \#\$xx_0$	← Offset from instruction's start (base) address ← Actual bytes of instruction's machine code ← Instruction behavior *8-bit immediate load into A register*
0 1 2 $\$AD\ xx_0\ xx_1$ $A := MemReadByte(\$xx_1xx_0)$ $LDA\ \$xx_1xx_0$	← Offset from instruction's start (base) address ← Actual bytes of instruction's machine code ← Instruction behavior *8-bit load from absolute address into A register*

6502 machine code (LDX instruction variants)	Comment
0 1 $\$A2\ xx_0$ $X := \$xx_0$ $LDX\ \#\$xx_0$	← Offset from instruction's start (base) address ← Actual bytes of instruction's machine code ← Instruction behavior *8-bit immediate load into X register*
0 1 2 $\$AE\ xx_0\ xx_1$ $X := MemReadByte(\$xx_1xx_0)$ $LDX\ \$xx_1xx_0$	← Offset from instruction's start (base) address ← Actual bytes of instruction's machine code ← Instruction behavior *8-bit load from absolute address into X register*

Load Value Into Register

```
LDA #$xx

LDA $xxxx



LDX imm/addr

LDY imm/addr
```

```
A := xx

A := ($xxxx)^



X := imm/addr

Y := imm/addr
```

2 different variants implies 2 different OPCODEs for LDX!

2 different variants implies additional 2 different OPCODEs for LDY!

Department of Distributed and Dependable Systems

& Store Value From Register

`LDA #$xx`	`A := xx`
`LDA $xxxx`	`A := ($xxxx)^`
`LDX imm/addr`	`X := imm/addr`
`LDY imm/addr`	`Y := imm/addr`
`STA $xxxx`	**`($xxxx)^ := A`**
`STX addr`	**`($addr)^ := X`**
`STY addr`	**`($addr)^ := Y`**

Copy (Transfer) Value Between Registers

`LDA #$xx`	`A := xx`
`LDA $xxxx`	`A := ($xxxx)^`
`LDX imm/addr`	`X := imm/addr`
`LDY imm/addr`	`Y := imm/addr`
`STA $xxxx`	`($xxxx)^ := A`
`STX addr`	`($addr)^ := X`
`STY addr`	`($addr)^ := Y`

`TAX`	**`X := A`**
`TXA`	**`A := X`**
`TAY`	**`Y := A`**
`TYA`	**`A := Y`**
`TSX`	**`X := S`**
`TXS`	**`S := X`**