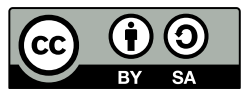
Machine code. Assembler

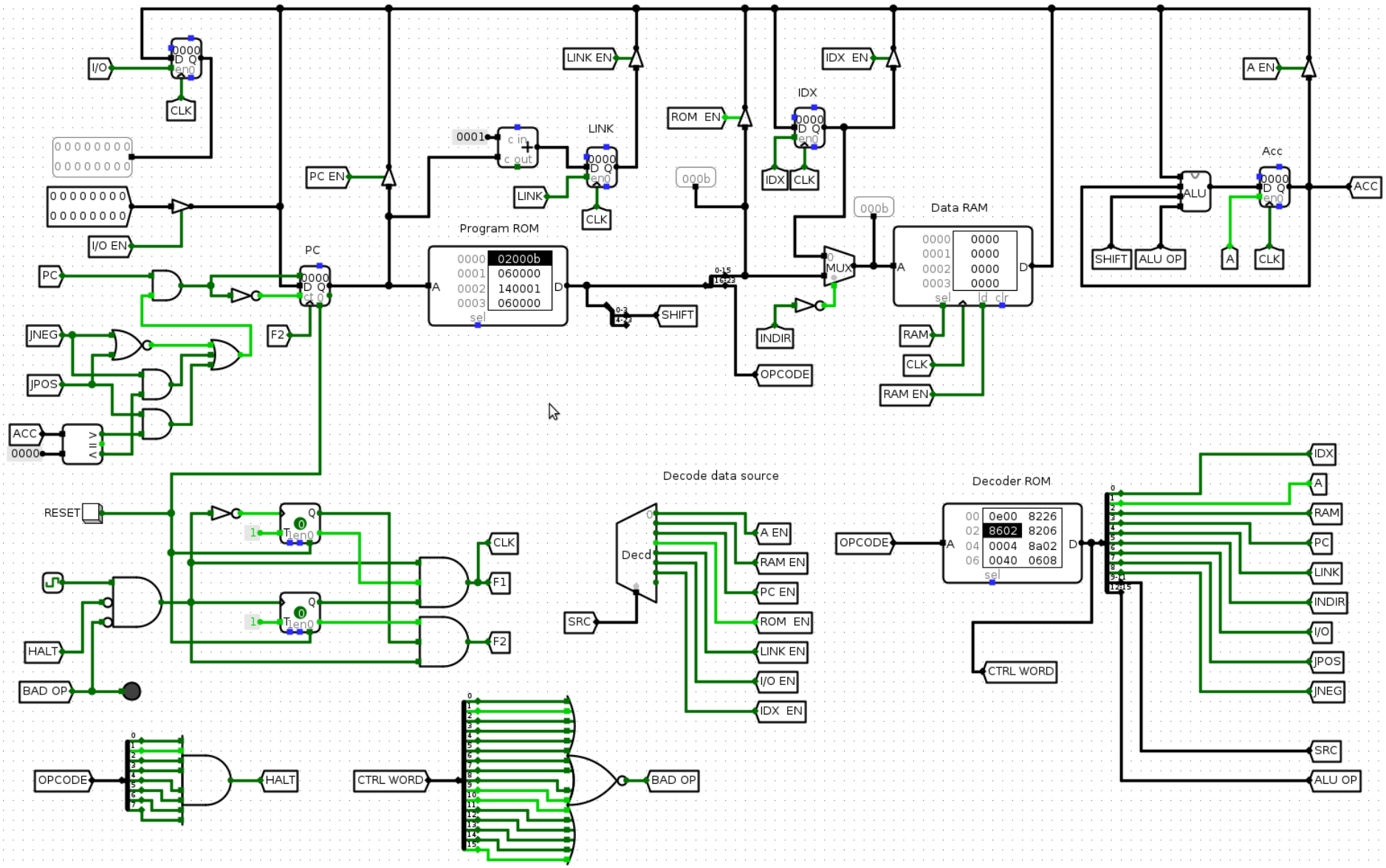
Saulius Gražulis

Vilnius, 2020

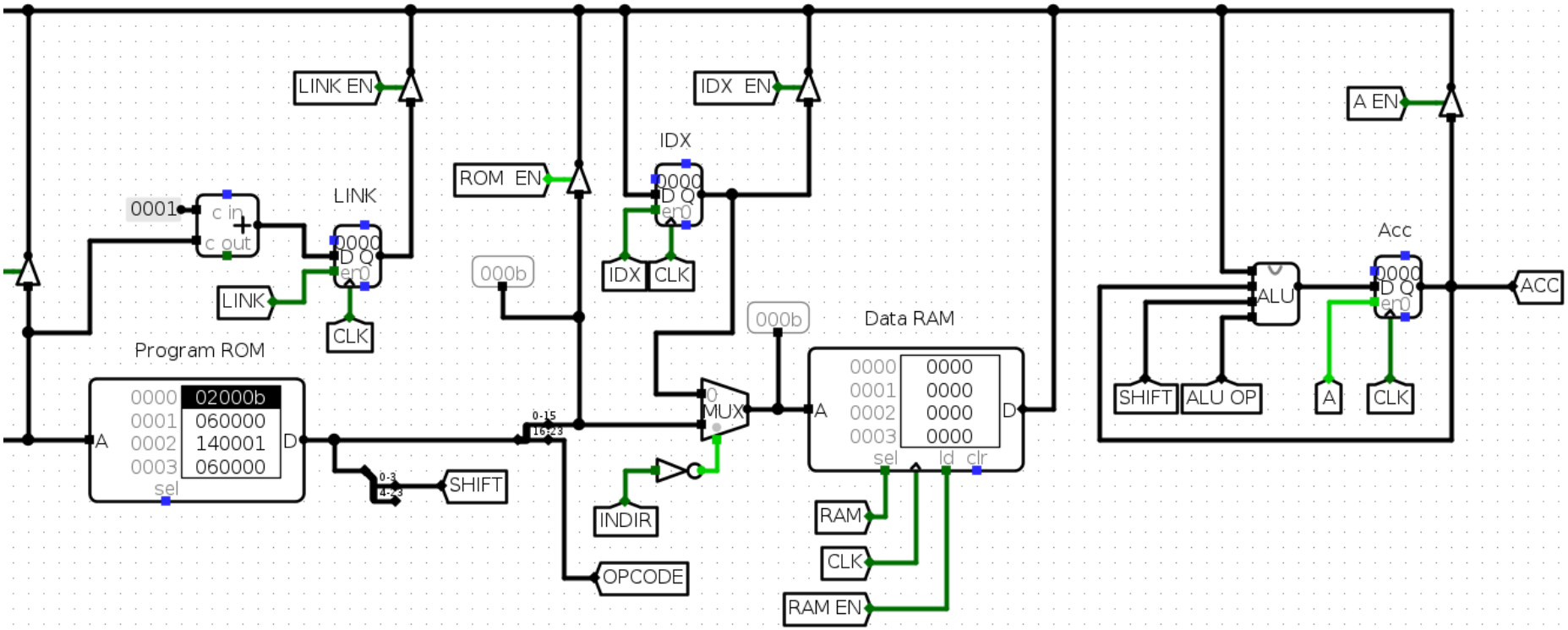
Vilnius University, Faculty of Mathematics and Informatics
Institute of Informatics



Simple Harvard architecture processor



Instruction memory



A simple algorithm

Fibonacci numbers

```c
int main()
{
    /* last two members of the
       sequence: */
    int a, b;
    int i;
    /* number of sequence members
       to produce: */
    int N = 21;

    a = b = 1;

    for( i = 1; i <= N; i++ ) {
        int t = b;
        b += a;
        printf( "%d\t%d\n", i, b );
        a = t;
    }

    return 0;
}
```

Step 0. Set $a \leftarrow 1$, $b \leftarrow 1$.
(a and b are two current sequence numbers)

Step i. for $1 \leq i \leq N$:

i. temporarily save b: $t \leftarrow b$;
ii. set $b \leftarrow a + b$;
iii. output b;
iv. set $a \leftarrow t$;

Step $N+1$. Terminate the program

Machine code

```c
int main()
{
    /* last two members of the
       sequence: */
    int a, b;
    int i;
    /* number of sequence members
       to produce: */
    int N = 21;

    a = b = 1;

    for( i = 1; i <= N; i++ ) {
        int t = b;
        b += a;
        printf( "%d\t%d\n", i, b );
        a = t;
    }

    return 0;
}
```

```
v2.0 raw
020001
040000
040001
020000
040002
030001
0B0000
060000
040001
0C0000
040000
030002
190001
040002
1A0015
0A0005
FFFFFF
```

Assembler example

```c
int main()
{
    /* last two members of the
       sequence: */
    int a, b;
    int i;
    /* number of sequence members
       to produce: */
    int N = 21;

    a = b = 1;

    for( i = 1; i <= N; i++ ) {
        int t = b;
        b += a;
        printf( "%d\t%d\n", i, b );
        a = t;
    }

    return 0;
}
```

```
        ;; Fibonacci numbers
A:      DS   1
B:      DS   1
I:      DS   1
N:      EQU  21
        ;;
        LDC  1
        ST   A
        ST   B
        LDC  0
        ST   I
LOOP:   LD   B
        ADD  A
        OUT  0
        ST   B
        SUB  A
        ST   A
        LD   I
        ADDC 1
        ST   I
        SUBC N
        JNEG LOOP
END:    HALT
```

Instruction mnemonics

Address	Machine code	Mnemonics		Meaning
0000	020001	LDC	1	Load const. 1 into the accumulator
0001	040000	ST	0	Store acc. to mem. location 0000
0002	040001	ST	1	
0003	020000	LDC	0	
0004	040002	ST	2	
0005	030001	LD	1	Load accumulator with [0001][1]
0006	0B0000	ADD	0	Add contents of location 0000 to acc.
...				
000D	1A0015	SUBC	15H	Subtract 21 from accumulator
000E	0A0005	JNEG	0005	Jump to prog. address 0005 if acc. < 0
000F	FFFFFF	HALT		Halt the CPU

[1] [0001] – contents of the data memory cell at the address 0001.

Instruction operands

```
N   EQU   21
A   EQU   0
B   EQU   1
I   EQU   2
```

Address	Machine code	Mnemonics		Meaning
0000	020001	LDC	1	Load const. 1 into the accumulator
0001	040000	ST	A	Store acc. to mem. location A
0002	040001	ST	B	
0003	020000	LDC	0	
0004	040002	ST	I	
0005	030001	LD	B	Load accumulator with [B][2]
0006	0B0000	ADD	A	Add contents of location A to acc.
...				
000D	1A0015	SUBC	N	Subtract N (= 21) from accumulator
000E	0A0005	JNEG	0005	Jump if accumulator is negative

[2] [B] – contents of the data memory cell that holds value of B.

Instruction operands

```
N   EQU   21
A   DS    1
B   DS    1
I   DS    1
```

Address	Machine code	Mnemonics		Meaning
0000	020001	LDC	1	Load const. 1 into the accumulator
0001	040000	ST	A	Store acc. to mem. location A
0002	040001	ST	B	
0003	020000	LDC	0	
0004	040002	ST	I	
0005	030001	LD	B	Load accumulator with [B][2]
0006	0B0000	ADD	A	Add contents of location A to acc.
...				
000D	1A0015	SUBC	N	Subtract N (= 21) from accumulator
000E	0A0005	JNEG	0005	Jump if accumulator is negative

[2] [B] – contents of the data memory cell that holds value of B.

Labels

```
        ORG   100
N   :   EQU   21
A   :   DS    1
B   :   DS    1
I   :   DS    1
```

Address	Machine code	Mnemonics		Meaning
	020001	LDC	1	Load const. 1 into the accumulator
	040000	ST	A	Store acc. to mem. location A
	040001	ST	B	
	020000	LDC	0	
	040002	ST	I	
LOOP:	030001	LD	B	Load acc. from mem. location B
	0B0000	ADD	A	Add contents of location A to acc.
...				
	1A0015	SUBC	N	Subtract N (= 21) from accumulator
	0A0005	JNEG	LOOP	Jump if accumulator is negative

Assembler program

Label	Operation	Operand(s)	Comment
	ORG	100	
N:	EQU	21	; Number of sequence members to compute
A:	DS	1	
B:	DS	1	
I:	DS	1	
	LDC	1	; Load const. 1 into accumulator
	ST	A	; Store acc. to mem. location A
	ST	B	
	LDC	0	
	ST	I	
LOOP:	LD	B	; Load acc. from mem. location B
	ADD	A	; Add contents of location A to acc.
...			
	SUBC	N	; Subtract 21 from accumulator
	JNEG	LOOP	; Repeat the loop if acc. is negative

Addressing modes

Mode	**Action**	**Description**
Immediate	A $\leftarrow$ operand	Use value that is in the command itself
Direct	A $\leftarrow$ [operand]	Use value using *address* in the command
Register	A $\leftarrow$ IDX	The value used is in a register
Indirect	A $\leftarrow$ [IDX]	*Address* of the value is in a register
Indexed	A $\leftarrow$ operand[IDX]	Address of the value is the register content + the operand

CPU model visible to programmer

From the point of view of a programmer, to describe a processor we need to specify:

- The **CPU registers** that are visible to the programmer
- The **Instruction set architecture (ISA)** of the CPU
- The **addressing modes** of the CPU
- The **memory organisation** of the CPU
- The **input/output** of the CPU

(Gutierrez-Osuna 2000)

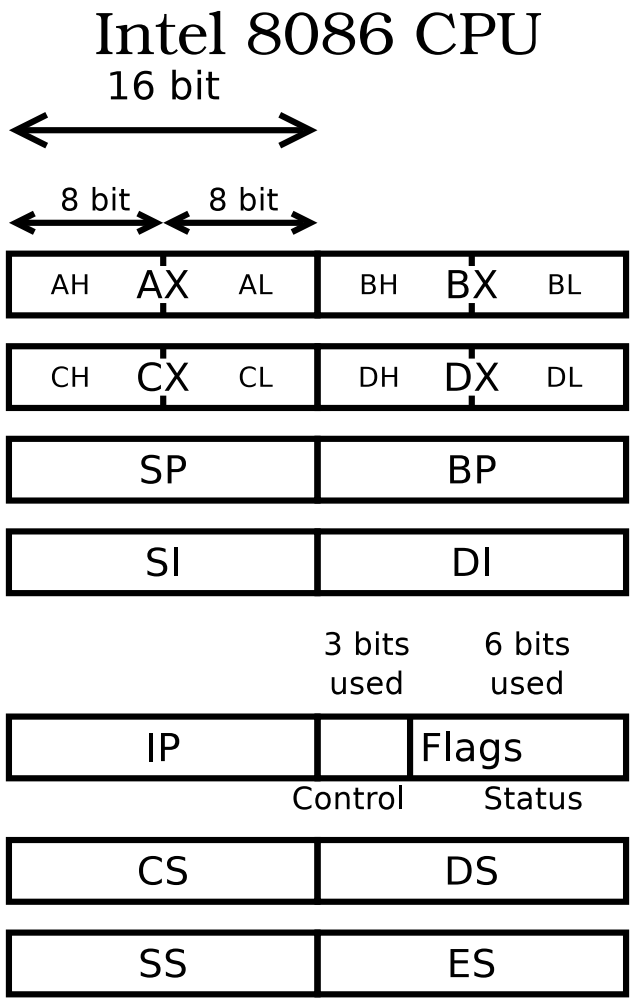
16-bit Harvard RISC-like CPU

- Registers: `A`, `PC`, `IDX`
- Instruction format:

OPC (8 bit)	ADDR/IMM (16 bit)

- Word size:
  - data word – 16 bits
  - instruction word – 24 (8 + 16) bits
- Address width: 16 bits
- Addressable unit: word
- Address space: flat
- ISA: accumulator architecture; `LD`, `LDC`, `ST`, `ADD`, `JNZ`, `JNEG`, `...`
- Input/output: ports in separate address space; no interrupts; one input and one output port

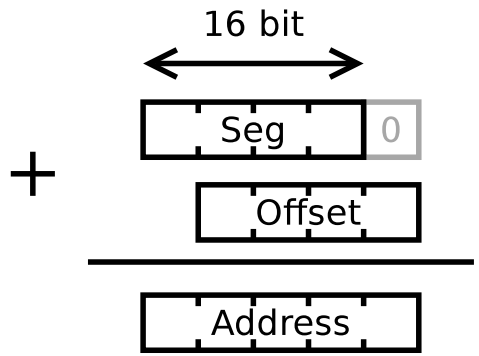
Registers



(Intel 1979)

Segmented addressing

Segment address is shifted by 4 bits and added to the offset:



Take home messages

- Machine code is binary (for binary machines), usually expressed in HEX
- Command mnemonics are used to facilitate instruction description and memorisation
- Assemblers are used to facilitate programming in machine language
- Main notions of assembler: labels, operations, operands, pseudoinstructions, comments
- Computer/CPU architecture can be described by specifying several programmer-visible parameters

References

Gutierrez-Osuna, Ricardo (2000). *Lecture 2: MC68000 architecture*. URL: http://courses.cs.tamu.edu/rgutier/ceg411_f01/l2.pdf.

Intel (Oct. 1979). *Intel 8086 Family User's Manual*. Intel Corporation. URL: https://edge.edx.org/c4x/BITSPilani/EEE231/asset/8086_family_Users_Manual_1_.pdf.