COS
Computersystemer

Lektion #4

Morten Hansen
moh@sdu.dk

Pointgivende aktivitet.

Den 1. oktober 2021 kl.: 12:15-13:00 afholder vi den 1. pointgivende aktivitet i jeres klasselokale. Det er vigtigt, at du er klar kl.: 12:15, da du ellers ikke kan deltage i prøven.

- Aktiviteten er *ikke* obligatorisk, men ved at deltage kan du score op til 5 point, som du kan lægge oven i din præstation til eksamen.
- Hvis du ikke deltager i den pointgivende aktivitet, kan du ***ikke*** få et andet forsøg. Dette gælder også i tilfælde af evt. Sygdom.
- Aktiviteten udføres som en *Multiple Choice Test* under *itslearning*.
- De udleverede pdf-dokumenter med *ASCII*-tabel, Boolsk Algebra, *Logic Gates* og *Tal repræsentationer, slides fra de fagets lektioner, lærebogen* samt egne noter, må benyttes under testen. Udover kladdepapir og blyant, er andre hjælpemidler ***ikke*** tilladt.
- Instruktorerne vil agere ”eksamens vagter”.

Morten Hansen
moh@sdu.dk

Semesterplanen

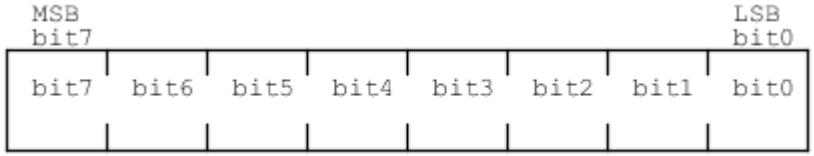
Husk at snakke om semesterplanen!

Morten Hansen
moh@sdu.dk

Bits, bytes og main memory

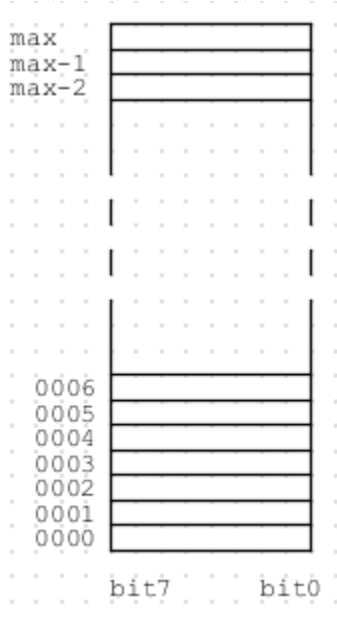
Bit:

- Mindst mulige mængde information.
- Kan antage 1 ud af 2 mulige værdier:
- 0 eller 1



Byte:

- Består af 8 bit.
- Kan antage 1 ud af 2^8 mulige værdier.
- =256 mulige, forskellige værdier, mønstre eller koder.

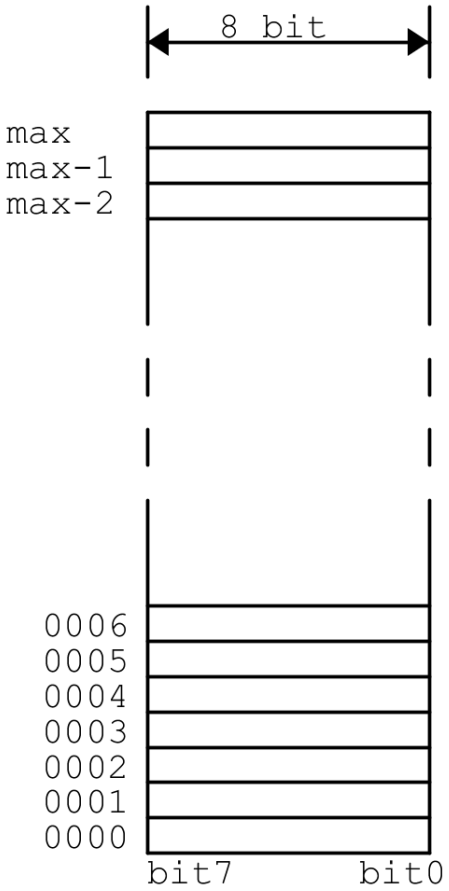


Main memory

- Består af en række celler
- Hver celle kan indeholde 8 bit information
- Hver celle har en adresse

Morten Hansen
moh@sdu.dk

Main memory

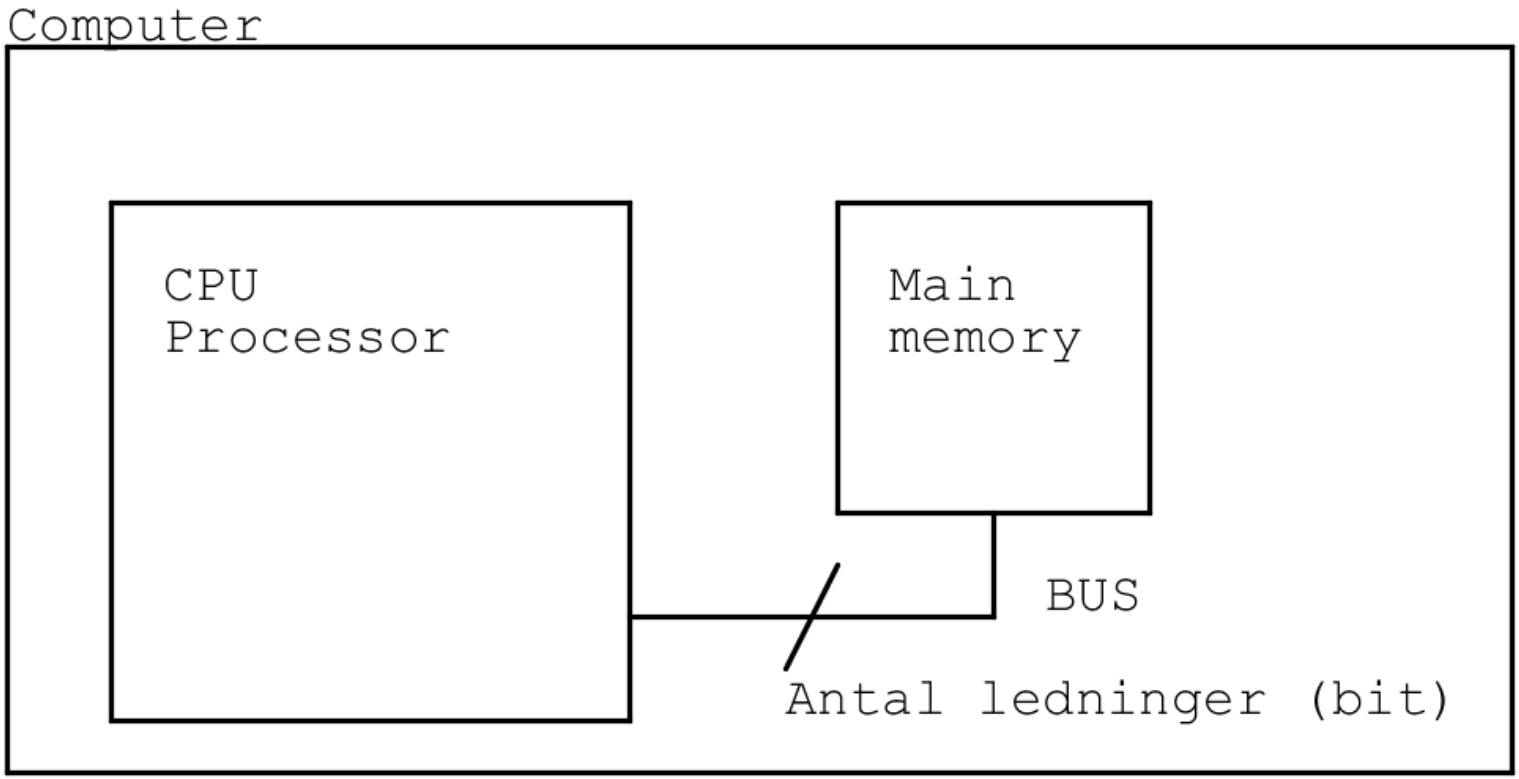


Data:
- Tal
- Karakterer
- Andet

Program:
- Maskinkoder

Morten Hansen
moh@sdu.dk

Computer arkitektur

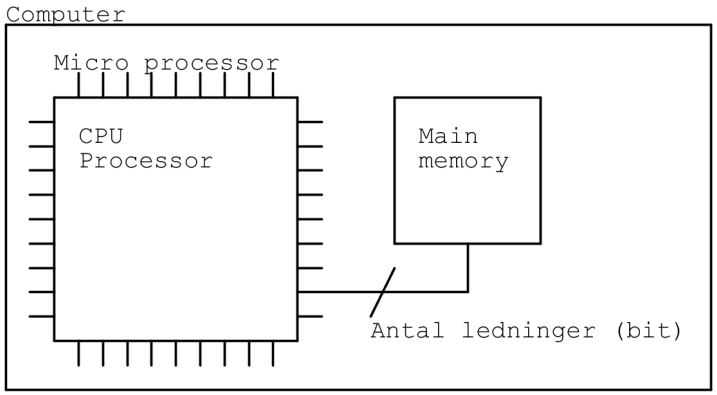


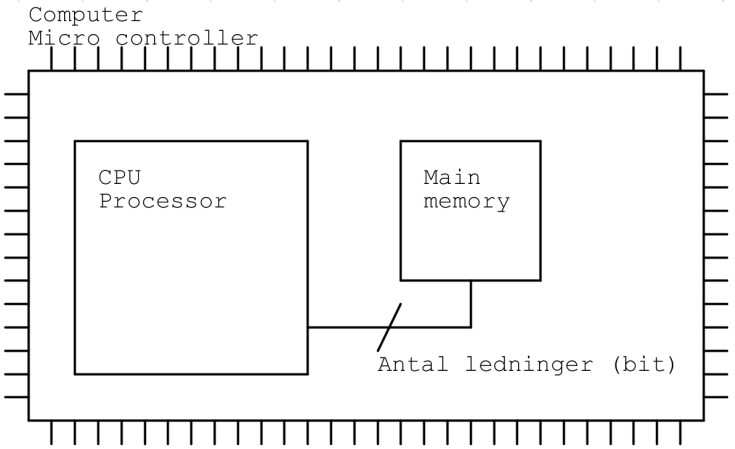
Morten Hansen
moh@sdu.dk

Micro processor og Micro controller.

Micro processor
CPU i en chip

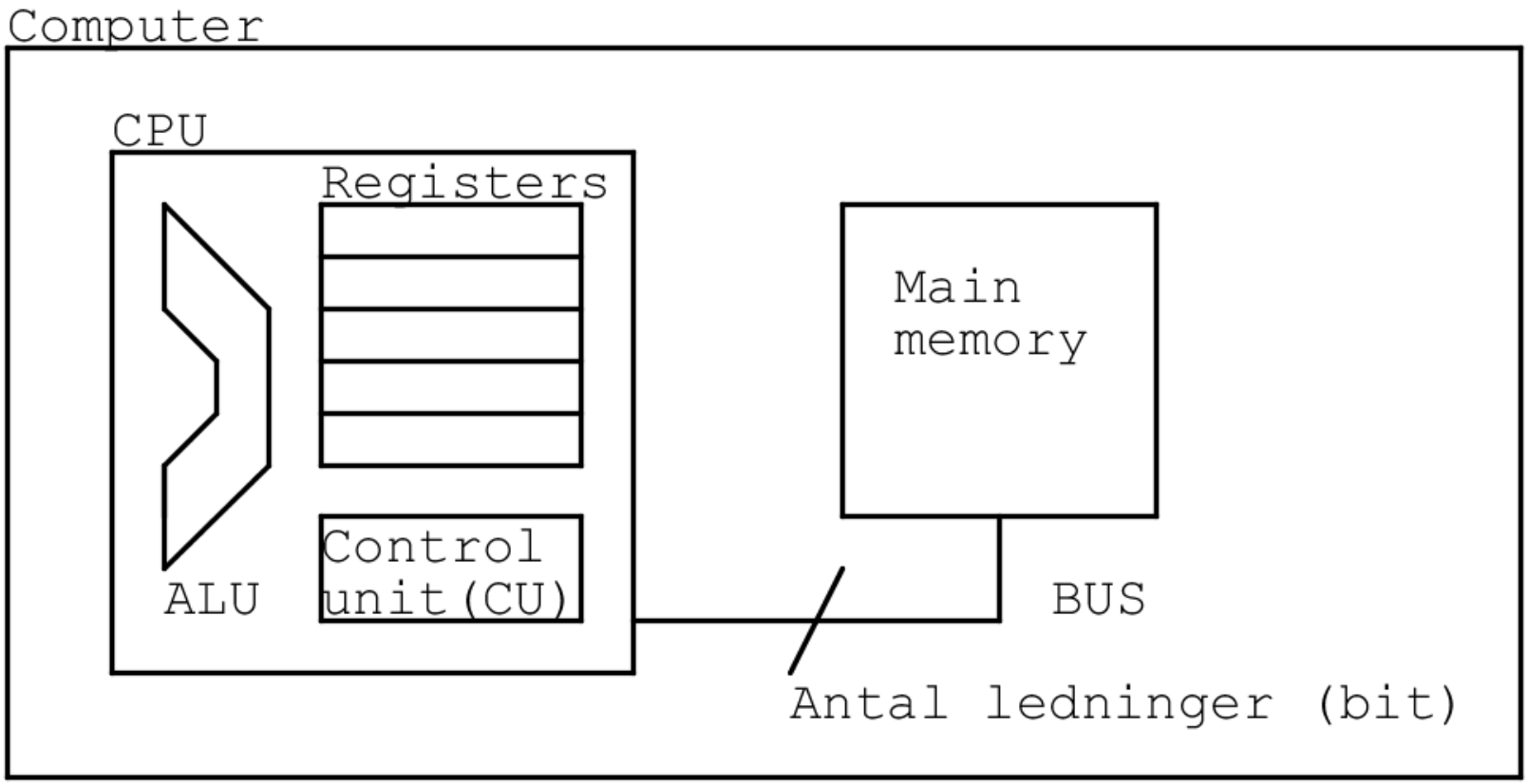
Micro controller
Computer i en chip





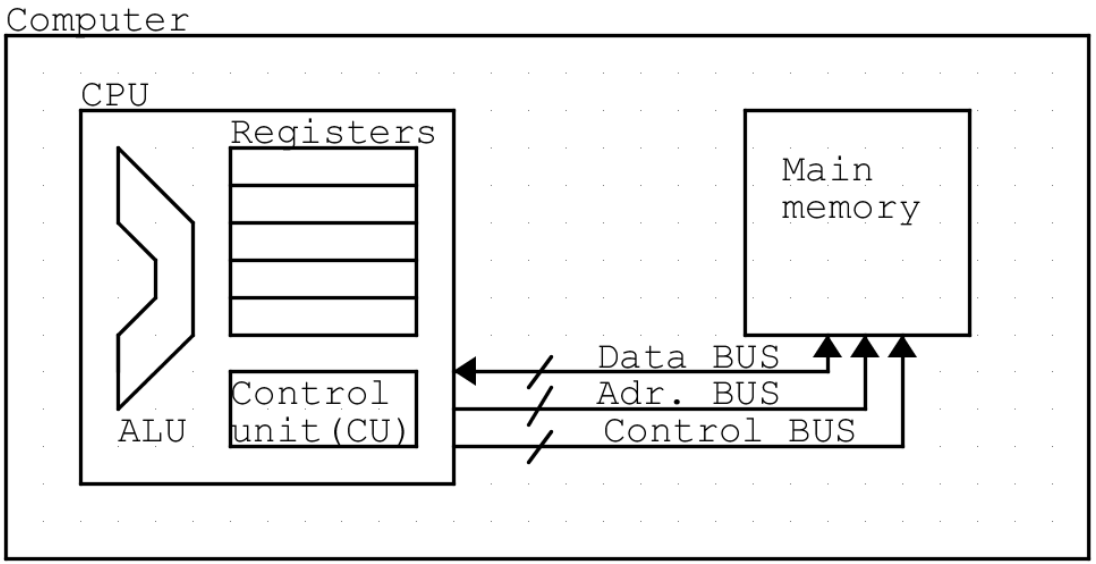
Morten Hansen
moh@sdu.dk

Inde i CPUen



Morten Hansen
moh@sdu.dk

De 3 "busser".



- Data BUS: Antal af bit, der kan overføres mellem CPU og main memory ad gangen.
- Adress BUS: Antal bit (b), der skal bruges til at vælge den rigtige celle i main memory.
  2^a = den maximale størrelse an main memory.
- Control BUS.
  1 bit til angivelse af READ eller WRITE.
  1 bit til at angive timing i dataoverførslen.
  …måske mere næste gang.

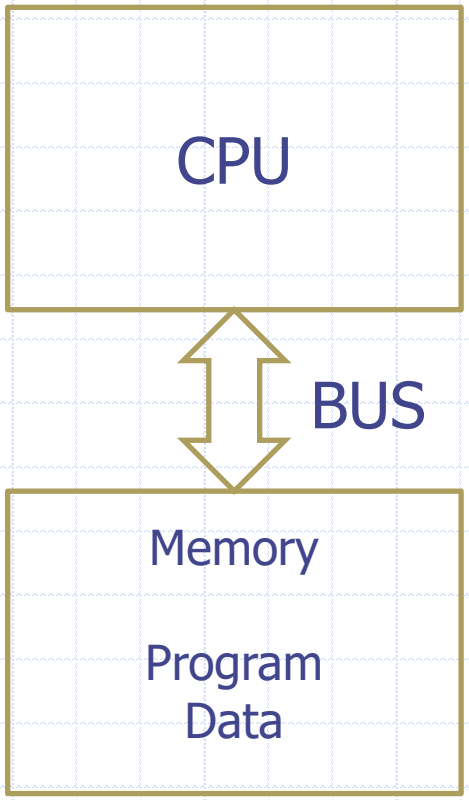
Morten Hansen
moh@sdu.dk

John Von Neuman

John von Neumann
(1903-1957)

Morten Hansen
moh@sdu.dk


John von Neumann
(1903-1957)



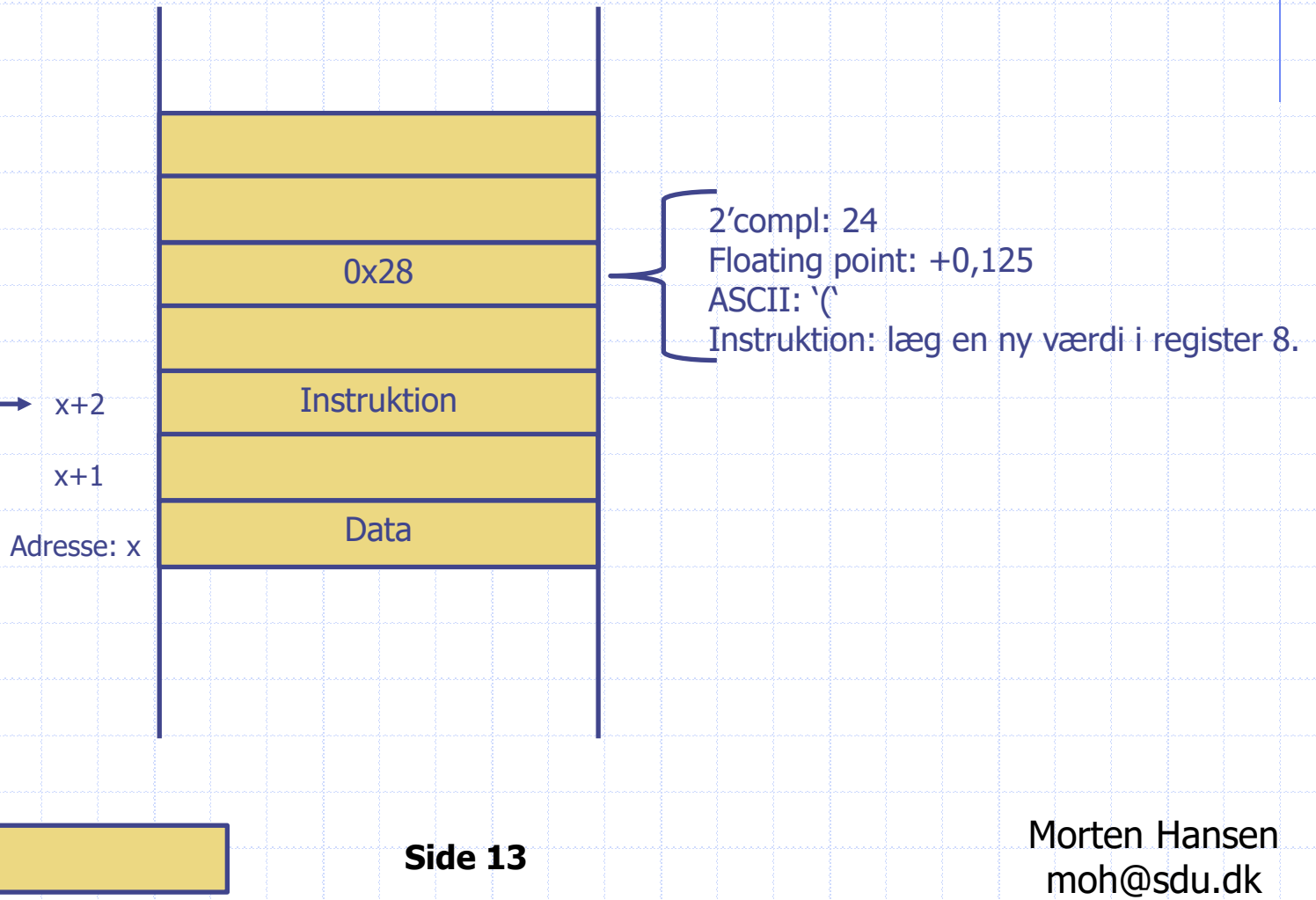
Morten Hansen
moh@sdu.dk

Stored Program Concept

A program can be encoded as bit patterns and stored in Main Memory. From there, the Control Unit can extract, decode, and execute instructions.

Instead of rewiring the CPU, the program can be altered by changing the contents of Main Memory.

Memory indhold



Morten Hansen
moh@sdu.dk

Maskinsprog.

- **Maskininstruktion (Machine instruction):** er en instruction til CPUen, der er kodet som et binært mønster.
- **Maskinsprog (Machine language):** er en samling af maskininstruktioner, der kan afkodes af en specific CPU.

Morten Hansen
moh@sdu.dk

Machine Language Philosophies

- Reduced Instruction Set Computing (RISC)
  - Few, simple, efficient, and fast instructions
  - Examples: PowerPC from Apple/IBM/Motorola and ARM
- Complex Instruction Set Computing (CISC)
  - Many, convenient, and powerful instructions
  - Example: Intel

Machine Instruction Types

- Data Transfer: copy data from one location to another (e.g. LOAD, STORE)
- Arithmetic/Logic: operations on bit patterns (e.g. +, -, *, /, AND, OR, SHIFT, ROTATE)
- Control: direct the execution of the program (e.g. JUMP, BRANCH)

Figure 2.2 Adding values stored in memory

Step 1. Get one of the values to be added from memory and place it in a register.

Step 2. Get the other value to be added from memory and place it in another register.

Step 3. Activate the addition circuitry with the registers used in Steps 1 and 2 as inputs and another register designated to hold the result.

Step 4. Store the result in memory.

Step 5. Stop.

Figure 2.3 Dividing values stored in memory

Step 1. LOAD a register with a value from memory.

Step 2. LOAD another register with another value from memory.

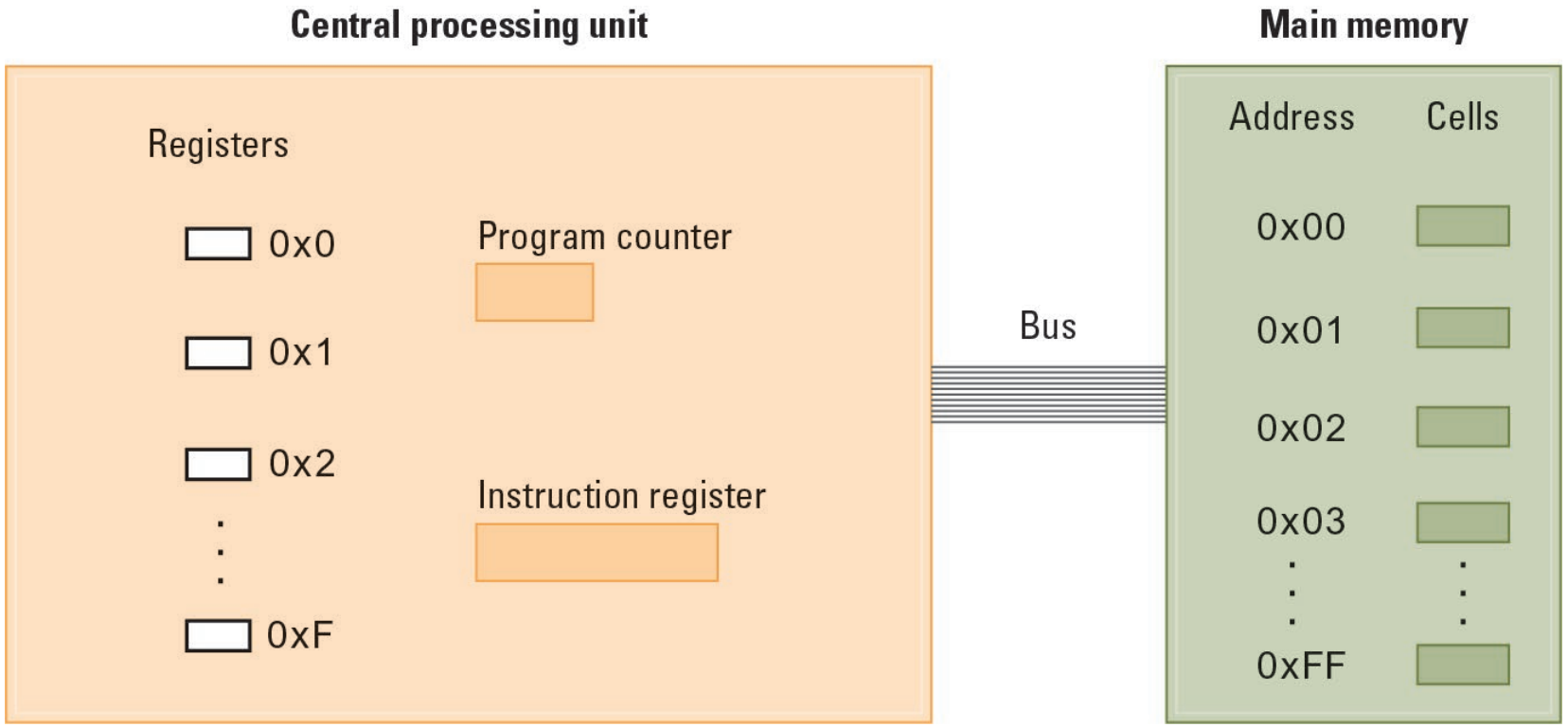
Step 3. If this second value is zero, JUMP to Step 6.

Step 4. Divide the contents of the first register by the second register and leave the result in a third register.

Step 5. STORE the contents of the third register in memory.

Step 6. STOP.

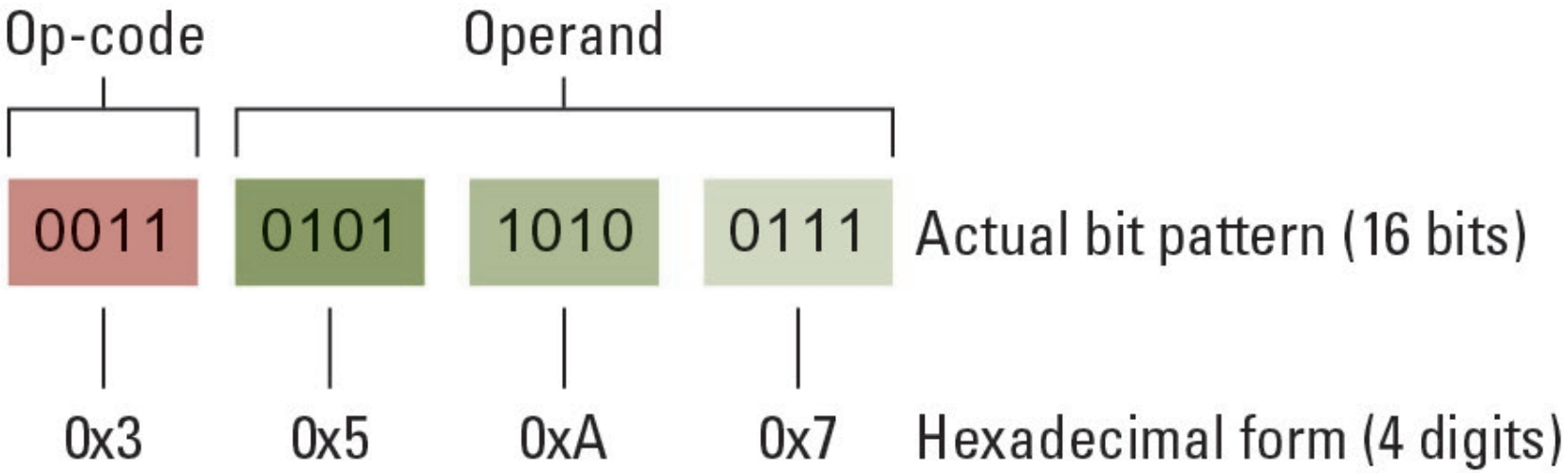
Figure 2.4 The architecture of the Vole, as described in Appendix C



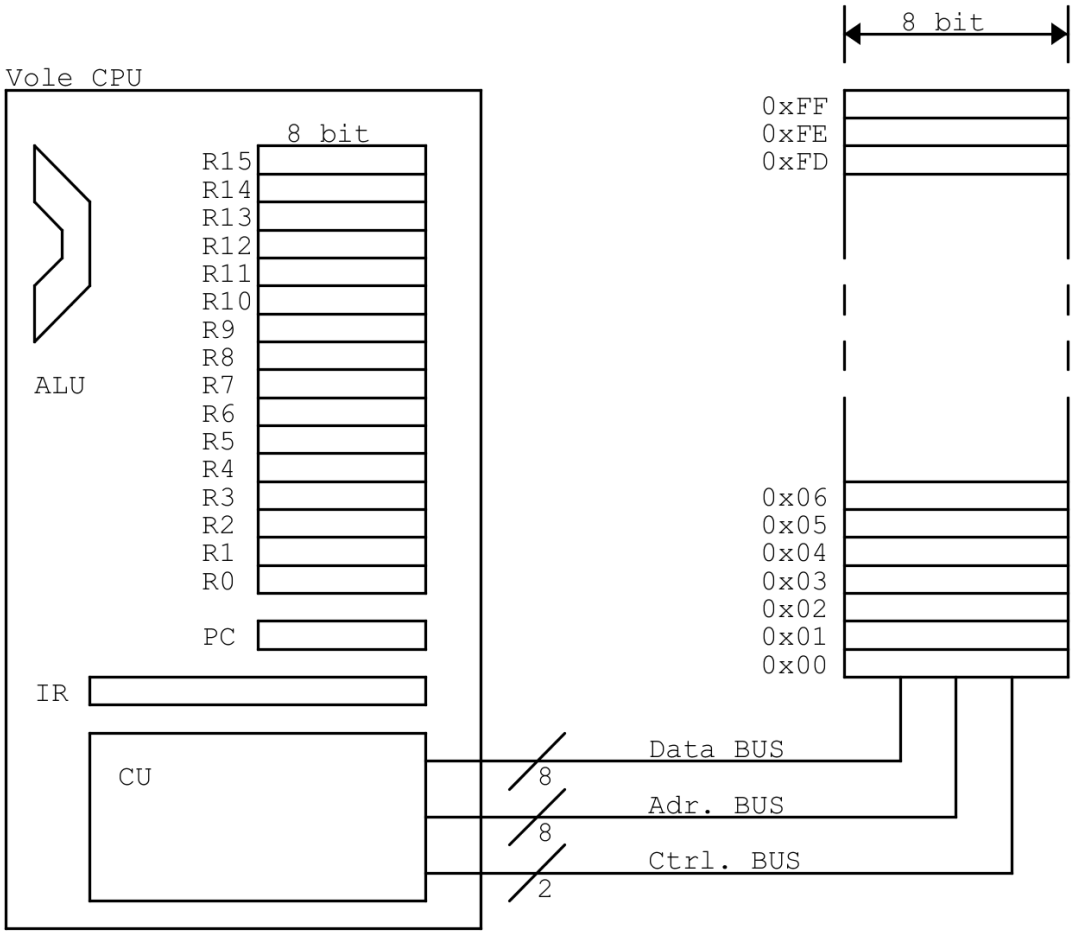
Parts of a Machine Instruction

- **Op-code:** Specifies which operation to execute
- **Operand:** Gives more detailed information about the operation
  - Interpretation of operand varies depending on op-code

Figure 2.5 The composition of a Vole instruction



VOLE i Appendix C



Data BUS: 8 bit data.
Adr. BUS: Antal memory celler = 2^8 = 256.
Ctrl. BUS:

- $\overline{RD}/WR$
- *Enable*

Instruktioner:

- LOAD
- STORE
- MOVE
- ADD
- OR
- AND
- XOR
- ROTATE
- JUMP
- HALT

Morten Hansen
moh@sdu.dk

Figure 2.7 An encoded version of the instructions in Figure 2.2

Encoded instructions	Translation
0x156C	Load register 0x5 with the bit pattern found in the memory cell at address 0x6C.
0x166D	Load register 0x6 with the bit pattern found in the memory cell at address 0x6D.
0x5056	Add the contents of register 0x5 and 0x6 as though they were two's complement representation and leave the result in register 0x0.
0x306E	Store the contents of register 0x0 in the memory cell at address 0x6E.
0xC000	Halt.

2.3 Program Execution

- Controlled by two special purpose registers
  - Instruction register
    - holds current instruction
  - Program counter
    - holds address of next instruction

- Machine Cycle: (repeat these 3 steps)
  - Fetch, Decode, Execute

Figure 2.8 The machine cycle

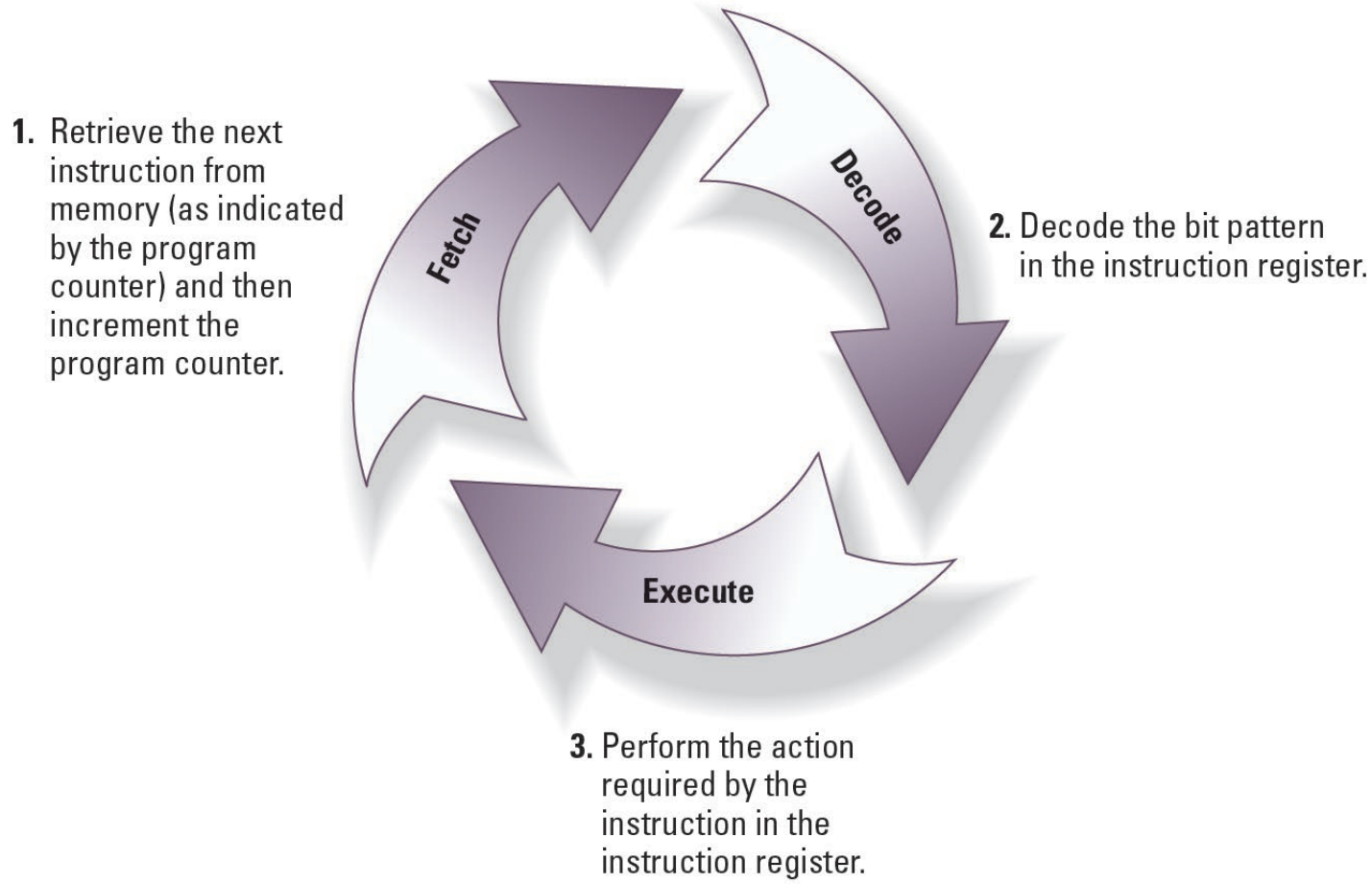
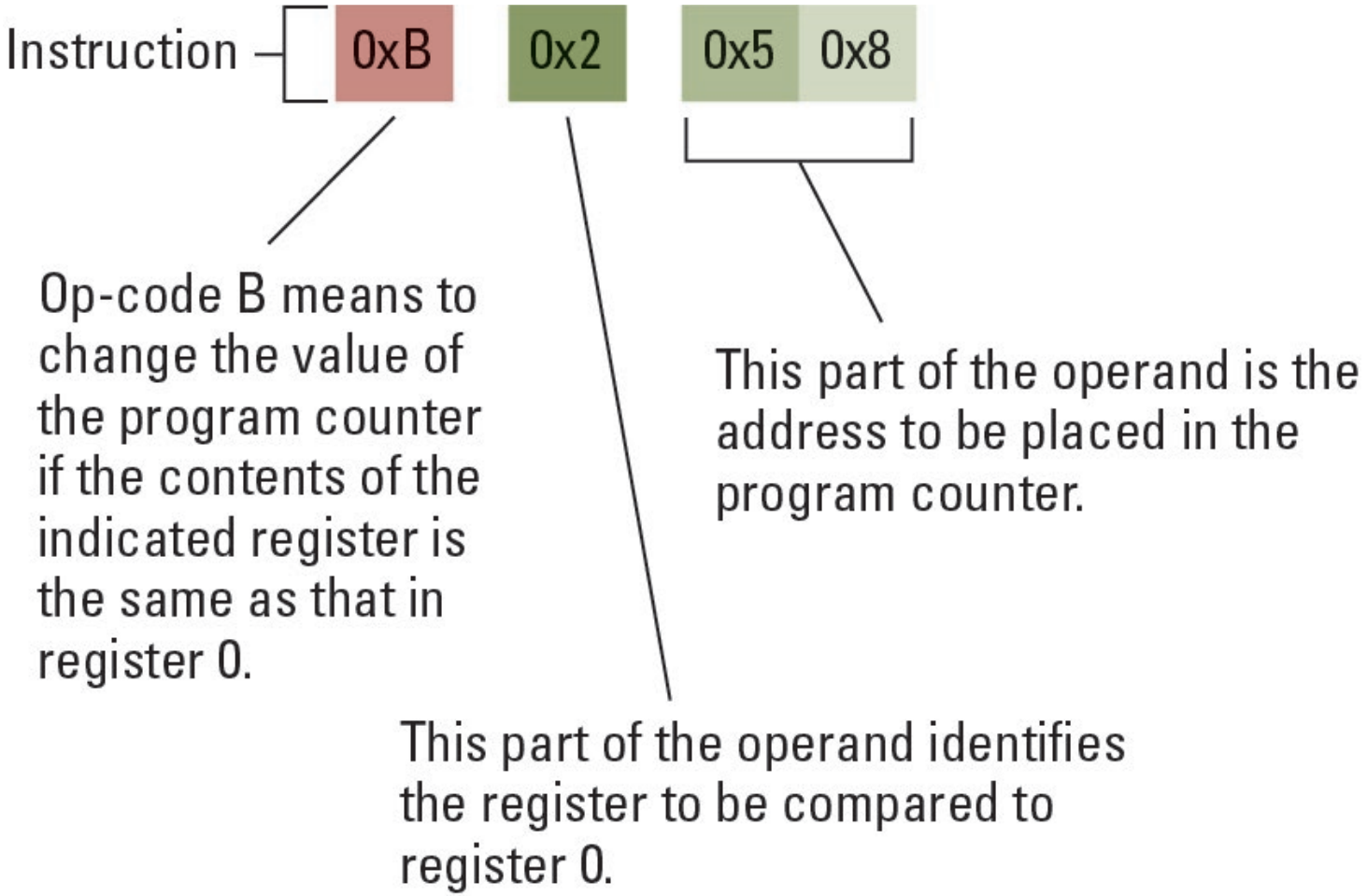


Figure 2.9 Decoding the instruction B258



VOLEs instruktionssæt.

Op-code	Operand	Description
1	RXY	LOAD the register R with the bit pattern found in the memory cell whose address is XY. *Example:* 14A3 would cause the contents of the memory cell located at address A3 to be placed in register 4.
2	RXY	LOAD the register R with the bit pattern XY. *Example:* 20A3 would cause the value A3 to be placed in register 0.
3	RXY	STORE the bit pattern found in register R in the memory cell whose address is XY. *Example:* 35B1 would cause the contents of register 5 to be placed in the memory cell whose address is B1.
4	0RS	MOVE the bit pattern found in register R to register S. *Example:* 40A4 would cause the contents of register A to be copied into register 4.
5	RST	ADD the bit patterns in registers S and T as though they were two's complement representations and leave the result in register R. *Example:* 5726 would cause the binary values in registers 2 and 6 to be added and the sum placed in register 7.
6	RST	ADD the bit patterns in registers S and T as though they represented values in floating-point notation and leave the floating-point result in register R. *Example:* 634E would cause the values in registers 4 and E to be added as floating-point values and the result to be placed in register 3.
7	RST	OR the bit patterns in registers S and T and place the result in register R. *Example:* 7CB4 would cause the result of ORing the contents of registers B and 4 to be placed in register C.
8	RST	AND the bit patterns in register S and T and place the result in register R. *Example:* 8045 would cause the result of ANDing the contents of registers 4 and 5 to be placed in register 0.
9	RST	EXCLUSIVE OR the bit patterns in registers S and T and place the result in register R. *Example:* 95F3 would cause the result of EXCLUSIVE ORing the contents of registers F and 3 to be placed in register 5.
A	R0X	ROTATE the bit pattern in register R one bit to the right X times. Each time place the bit that started at the low-order end at the high-order end. *Example:* A403 would cause the contents of register 4 to be rotated 3 bits to the right in a circular fashion.
B	RXY	JUMP to the instruction located in the memory cell at address XY if the bit pattern in register R is equal to the bit pattern in register number 0. Otherwise, continue with the normal sequence of execution. (The jump is implemented by copying XY into the program counter during the execute phase.) *Example:* B43C would first compare the contents of register 4 with the contents of register 0. If the two were equal, the pattern 3C would be placed in the program counter so that the next instruction executed would be the one located at that memory address. Otherwise, nothing would be done and program execution would continue in its normal sequence.
C	000	HALT execution. *Example:* C000 would cause program execution to stop.

Se: itslearning/resourcer/værktøjer/
VOLE cpu Simulator

Figure 2.10 The program from Figure 2.7 stored in main memory ready for execution

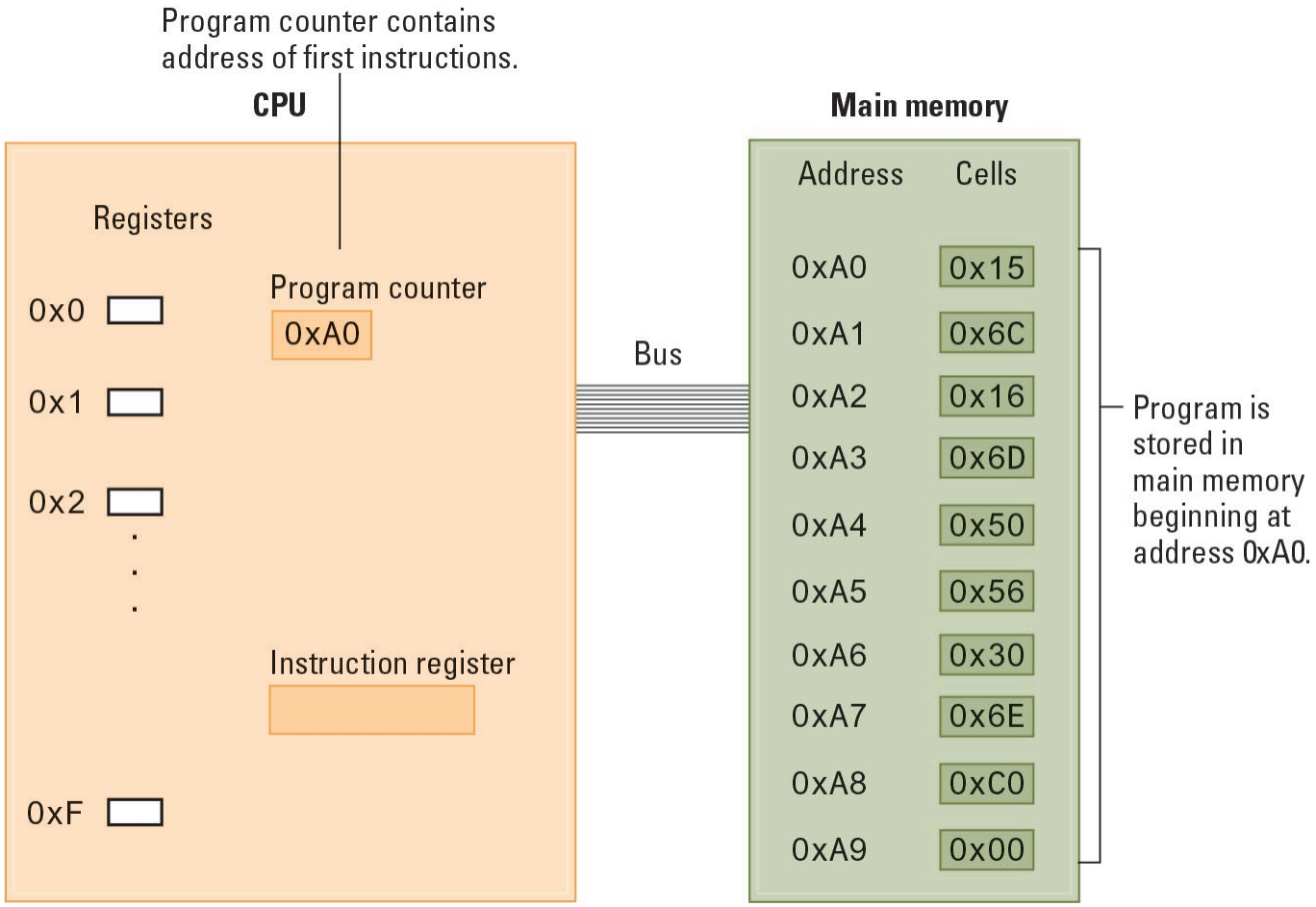
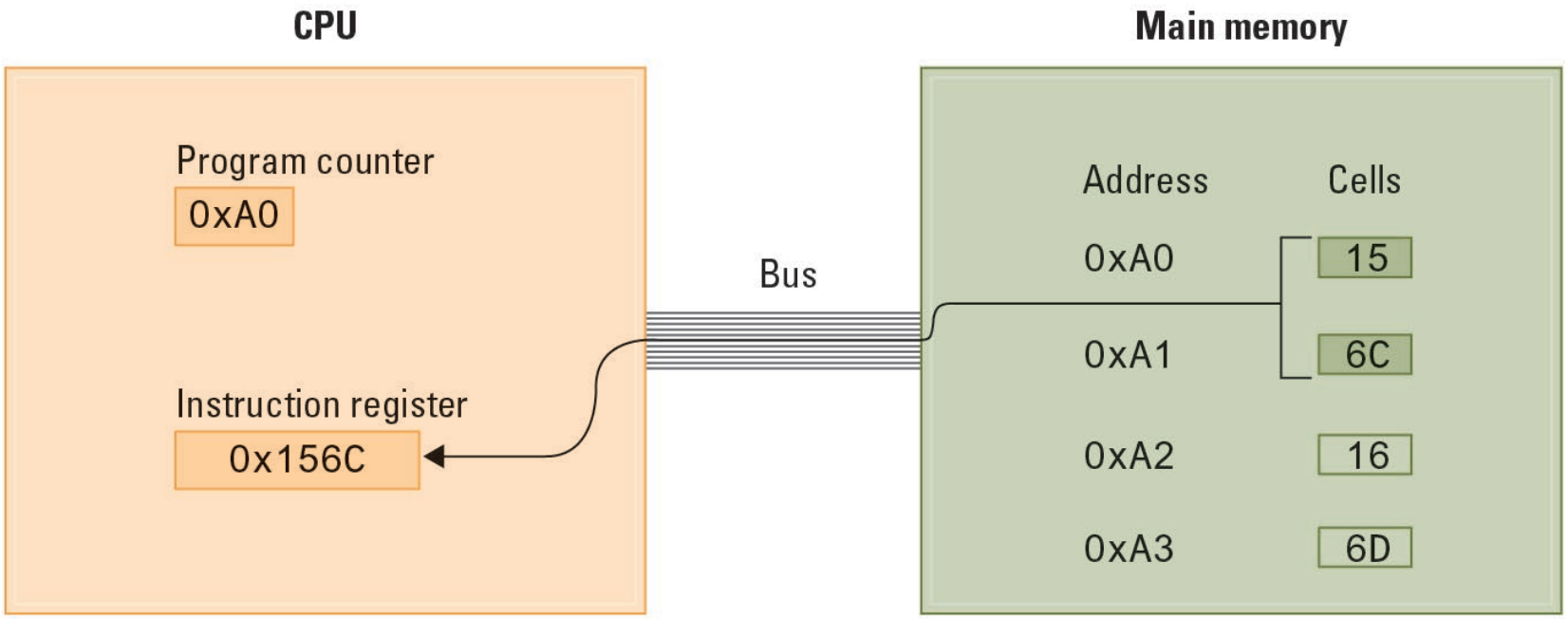
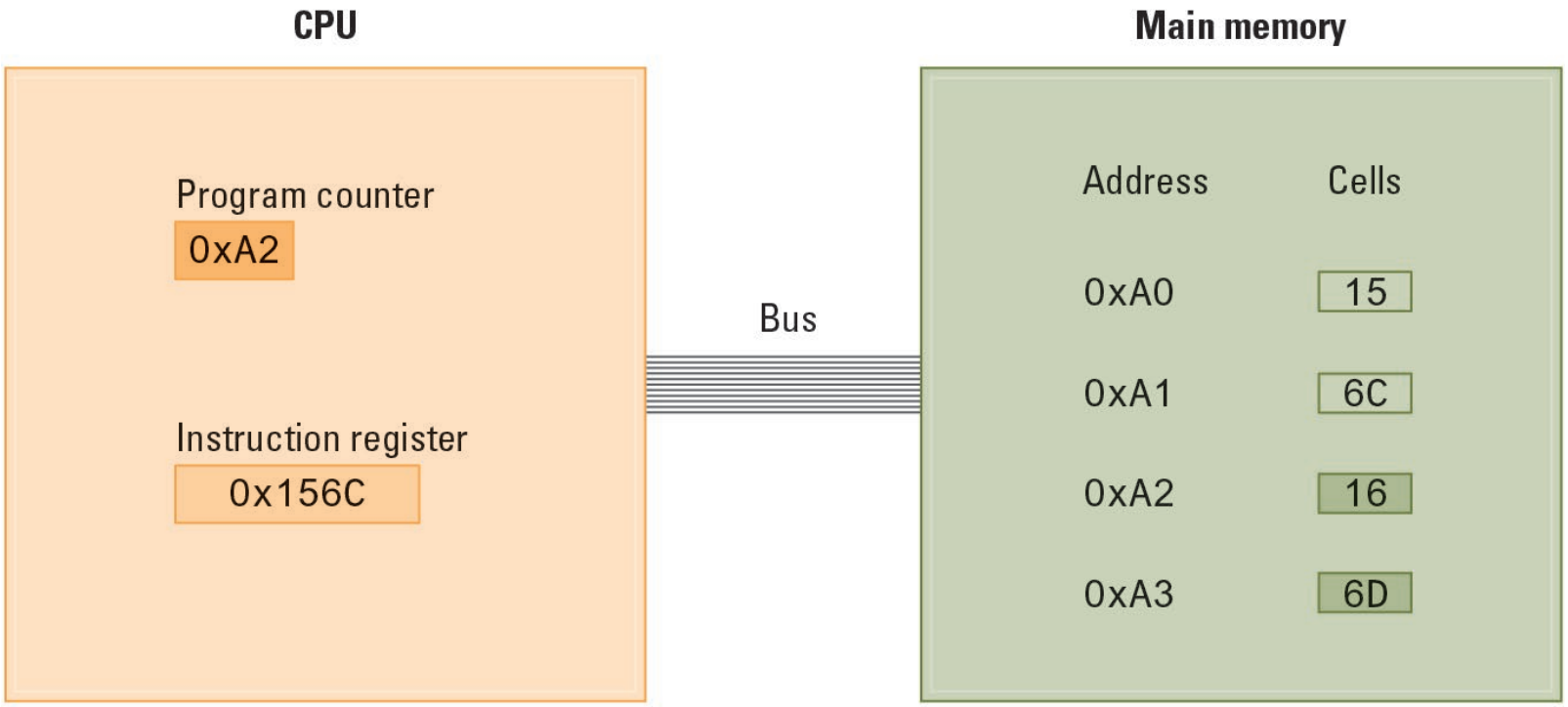


Figure 2.11 Performing the fetch step of the machine cycle



a. At the beginning of the fetch step, the instruction starting at address 0xA0 is retrieved from memory and placed in the instruction register.

Figure 2.11 Performing the fetch step of the machine cycle (continued)



b. Then the program counter is incremented so that it points to the next instruction.

Programmering

Adder de binære værdier i adr.: 0x40 og 0x41. Aflever resultatet i adr.: 0x42

```
Maskin-          Assembler                  Høj niveau kode
instruktioner    Kode                       3. generations
                                            programmeringssprog
                                            (her: C)
-------------------------------------------------------------------------
0x1140           LOAD  R1,[0x40]            Resultat = Addent1 + Addent2;
0x1241           LOAD  R2,[0x41]
0x5012           ADD   R0, R1, R2
0x3042           STORE R0,[0x42]
```


Morten Hansen
moh@sdu.dk

2.4 Arithmetic/Logic Instructions

- Logic Operations:
  - AND, OR, XOR
  - often used to mask an operand
- Rotation and Shift Operations:
  - circular shift, logical shift, arithmetic shift
- Arithmetic Operations:
  - add, subtract, multiply, divide
  - two's complement versus floating-point

De logiske operationer er ”bit-wise”!

```
    01010101
AND 11110000
    --------
    01010000


    01010101
OR  11110000
    --------
    11110101


    01010101
XOR 11110000
    --------
    10100101
```


Morten Hansen
moh@sdu.dk

Masking.

```
Eks.: Set bit 2:

    10101010
OR  00000100
    --------
    10101110


    01010101
OR  00000100
    --------
    01010101
```

```
Eks.: Clear bit 2:

    01010101
AND 11111011
    --------
    01010001


    10101010
AND 11111011
    --------
    10101010
```

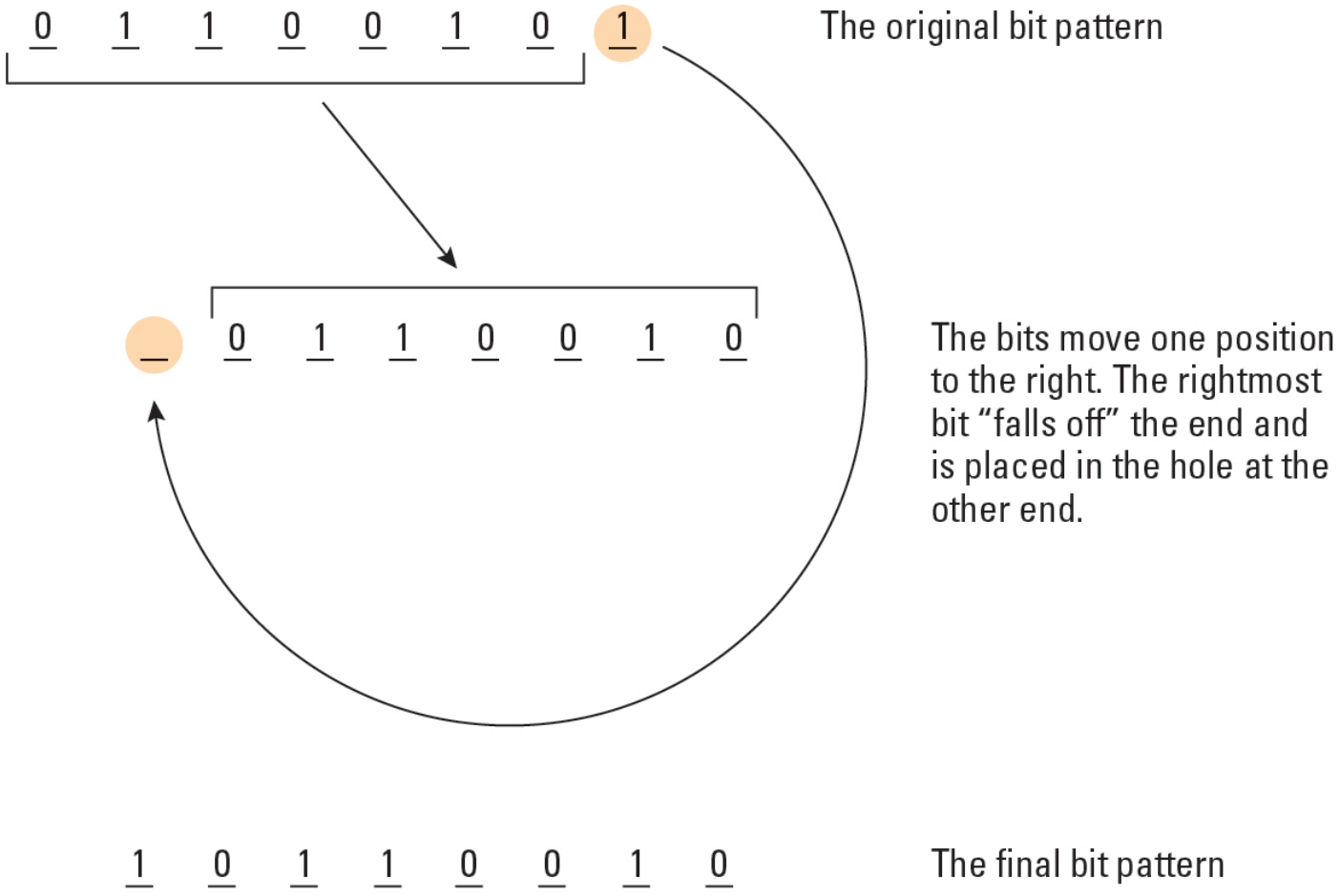
```
Eks.: Test bit 2:

    10101010
AND 00000100
    --------
    00000000  = 0 => bit2 = 0


    01010101
AND 00000100
    --------
    00000100  ≠ 0  => bit2 = 0
```


Morten Hansen
moh@sdu.dk

Figure 2.12 Rotating the bit pattern 0x65 one bit to the right



Spørgsmål?

Morten Hansen
moh@sdu.dk