NCS 362: Embedded Systems

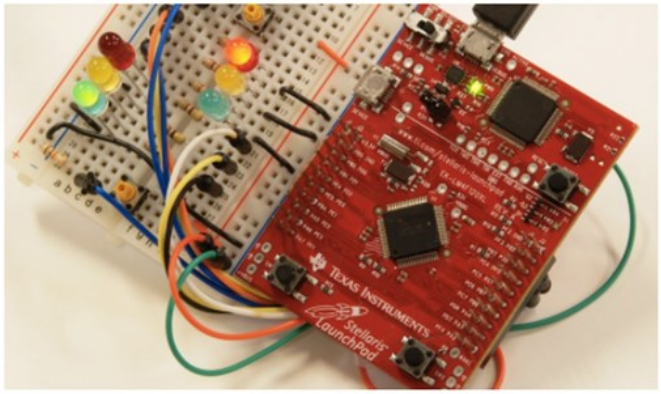
Instructor:
Dr. Ahmed Shalaby http://bu.edu.eg/staff/ahmedshalaby14#

How ? Course Book

Embedded Systems - Shape The World
http://users.ece.utexas.edu/~valvano/Volume1/E-Book/

users.ece.utexas.edu/~valvano/Volume1/E-Book/

Embedded Systems - Shape The World

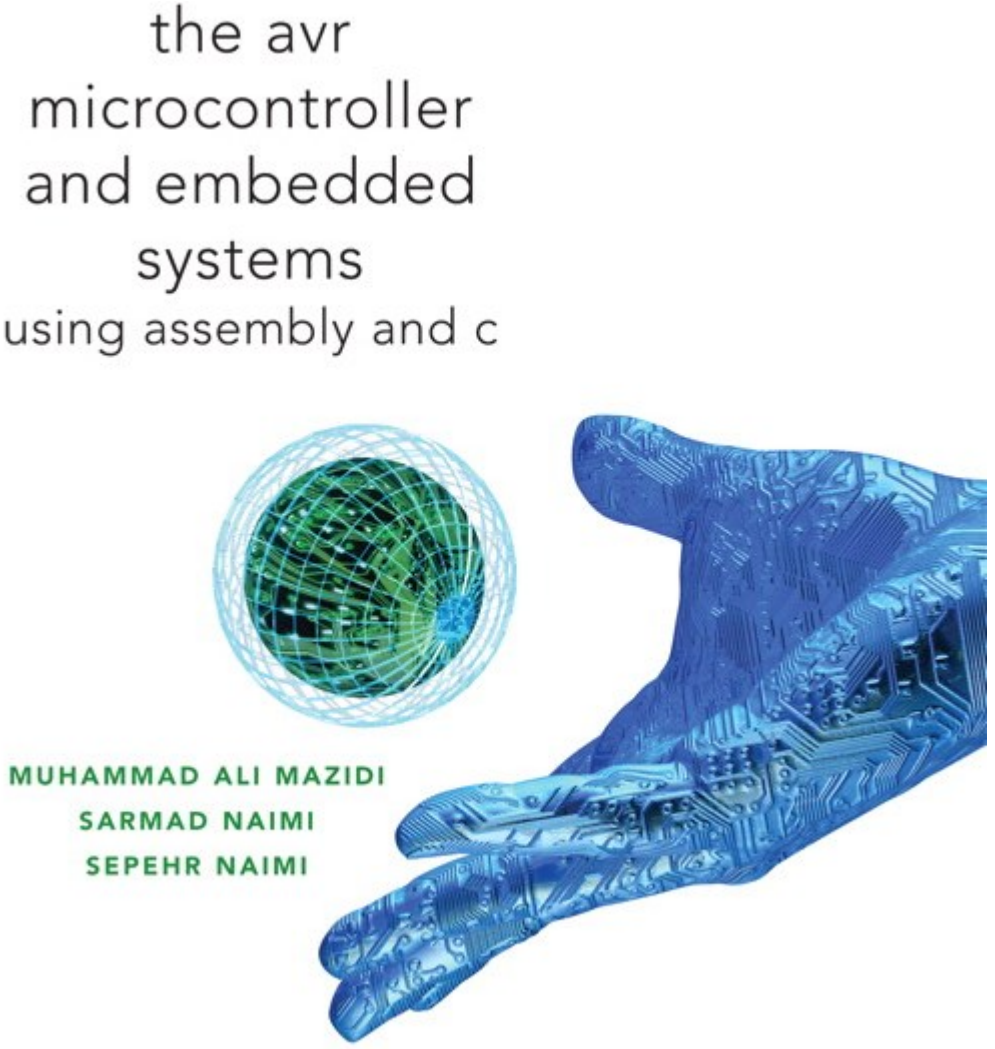


Jonathan Valvano and Ramesh Yerraballi

Table of Contents

Embedded Software in C
http://users.ece.utexas.edu/~valvano/embed/toc1.htm

How ? Course Book



Introduction to Computing
Chapter 0

The AVR microcontroller and embedded systems
using assembly and c

Topics

- Internal organization of computers
  - The different parts of a computer
    - I/O
    - Memory
    - CPU
  - Connecting the different parts
    - Connecting memory to CPU
    - Connecting I/Os to CPU
  - How computers work

Internal organization of computers

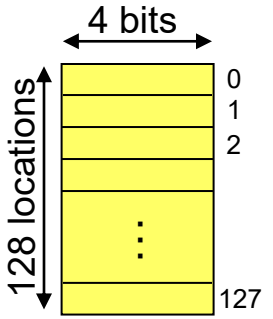
- CPU
- Memory
- I/O
  - Input
    - E.g. Keyboard, Mouse, Sensor
  - Output
    - E.g. LCD, printer, hands of a robot

Memory

- Everything that can store, retain, and recall information.
  - E.g. hard disk, a piece of paper, etc.

Memory characteristics

- Capacity
  - The number of bits that a memory can store.
    - E.g. 128 Kbits, 256 Mbits
- Organization
  - How the locations are organized
    - E.g. a 128 x 4 memory has 128 locations, 4 bits each
- Access time
  - How long it takes to get data from memory



Semiconductor memories

- ROM
  - Mask ROM
  - PROM (Programmable ROM)
  - EPROM (Erasable PROM)
  - EEPROM (Electronic Erasable PROM)
  - Flash EPROM
- RAM
  - SRAM (Static RAM)
  - DRAM (Dynamic RAM)
  - NV-RAM (Nonvolatile RAM)

Memory\ROM\Mask ROM

- Programmed by the IC manufacturer

Memory\ROM\PROM (Programmable ROM)

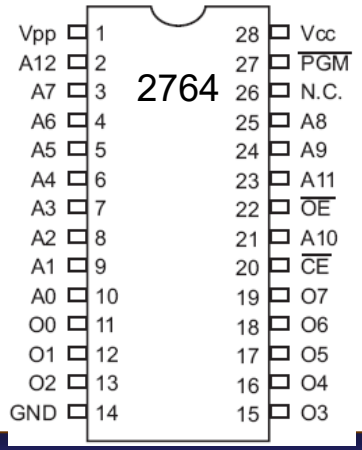
- OTP (One-Time Programmable)
  - You can program it only once

Memory\ROM\EPROM (Erasable Programmable ROM)

- UV-EPROM
  - You can shine ultraviolet (UV) radiation to erase it
  - Erasing takes up to 20 minutes
  - The entire contents of ROM are erased

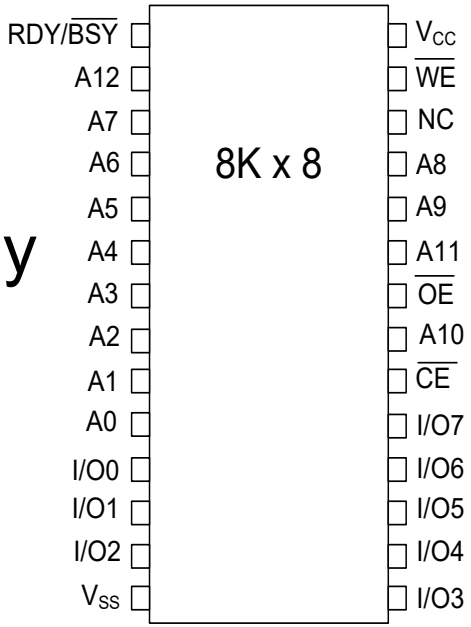
Table 0-5: Some UV-EPROM Chips

Part #	Capacity	Org.	Access	Pins	V_{PP}
2716	16K	2K × 8	450 ns	24	25 V
2732	32K	4K × 8	450 ns	24	25 V
2732A-20	32K	4K × 8	200 ns	24	21 V
27C32-1	32K	4K × 8	450 ns	24	12.5 V CMOS
2764-20	64K	8K × 8	200 ns	28	21 V
2764A-20	64K	8K × 8	200 ns	28	12.5 V
27C64-12	64K	8K × 8	120 ns	28	12.5 V CMOS



Memory\ROM\EEPROM (Electrically Erasable Programmable ROM)

- Erased Electrically
  - Erased instantly
  - Each byte can be erased separately



Part No.	Capacity	Org.	Speed	Pins	V_{PP}
2816A-25	16K	2K × 8	250 ns	24	5 V
2864A	64K	8K × 8	250 ns	28	5 V
28C64A-25	64K	8K × 8	250 ns	28	5 V CMOS
28C256-15	256K	32K × 8	150 ns	28	5 V
28C256-25	256K	32K × 8	250 ns	28	5 V CMOS

Memory\ROM\Flash ROM

- Erased in a Flash
- the entire device is erased at once

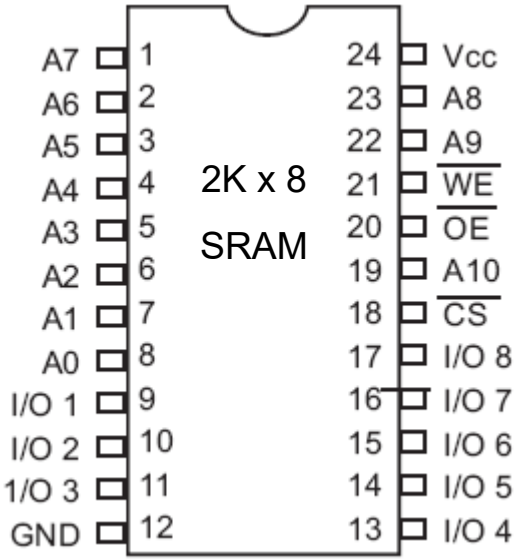
Part No.	Capacity	Org.	Speed	Pins	V_{PP}
28F256-20	256K	32K × 8	200 ns	32	12 V CMOS
28F010-15	1024K	128K × 8	150 ns	32	12 V CMOS
28F020-15	2048K	256K × 8	150 ns	32	12 V CMOS

Semiconductor memories

- ROM
  - Mask ROM
  - PROM (Programmable ROM)
  - EPROM (Erasable PROM)
  - EEPROM (Electronic Erasable PROM)
  - Flash EPROM
- RAM
  - SRAM (Static RAM)
  - DRAM (Dynamic RAM)
  - NV-RAM (Nonvolatile RAM)

Memory\RAM\SRAM (Static RAM)

- Made of flip-flops (Transistors)
- Advantages:
  - Faster
  - No need for refreshing
- Disadvantages:
  - High power consumption
  - Expensive



Memory\RAM\DRAM (Dynamic RAM)

- Made of capacitors
- Advantages:
  - Less power consumption
  - Cheaper
  - High capacity
- Disadvantages:
  - Slower
  - Refresh needed

Memory\RAM\NV-RAM (Nonvolatile RAM)

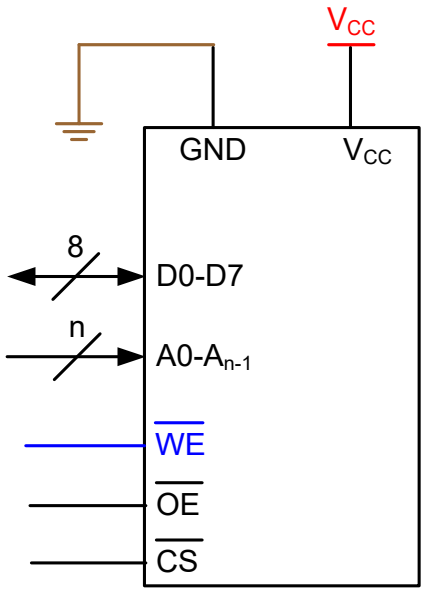
- Made of SRAM, Battery, control circuitry
- Advantages:
  - Very fast
  - Infinite program/erase cycle
  - Non-volatile
- Disadvantage:
  - Expensive

Internal parts of computers\CPU

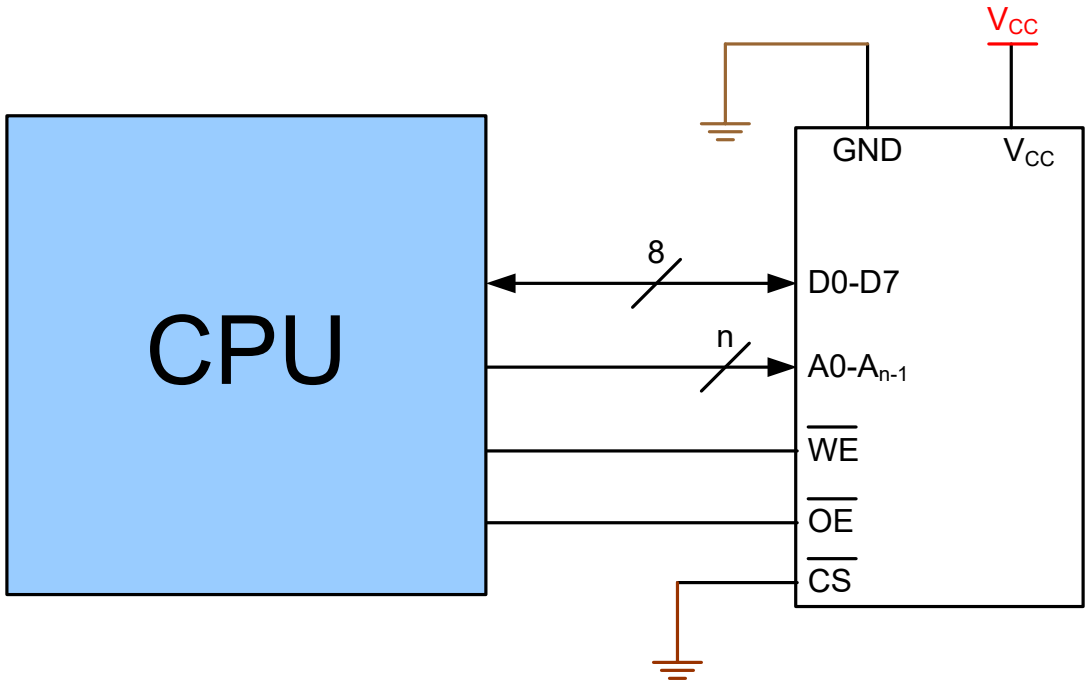
- Tasks:
  - It should execute instructions
    - It should recall the instructions one after another and execute them

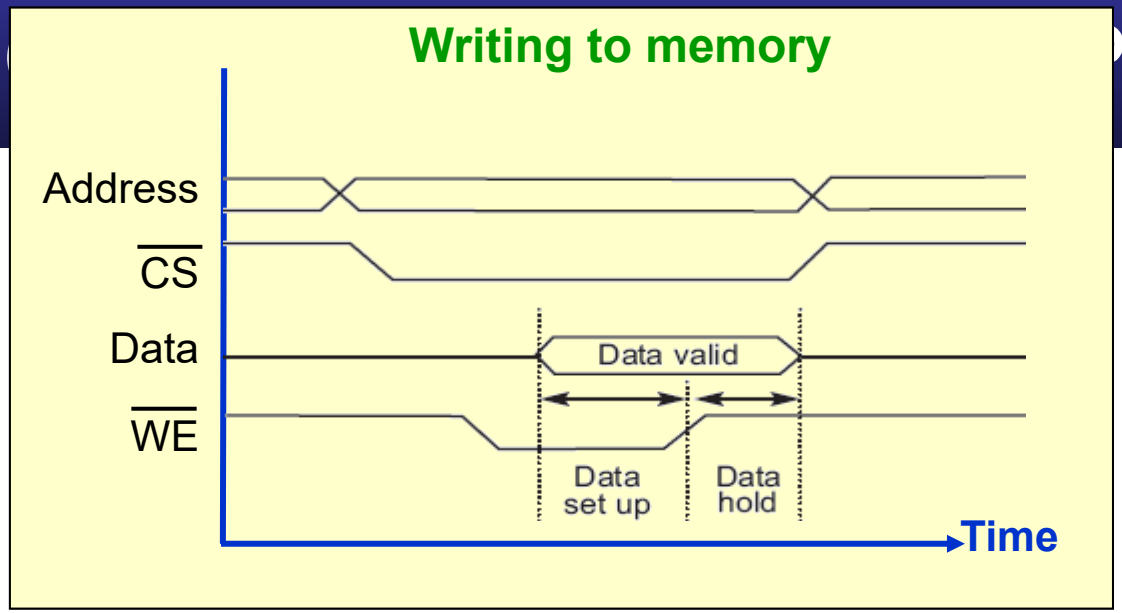
Connecting memory to CPU

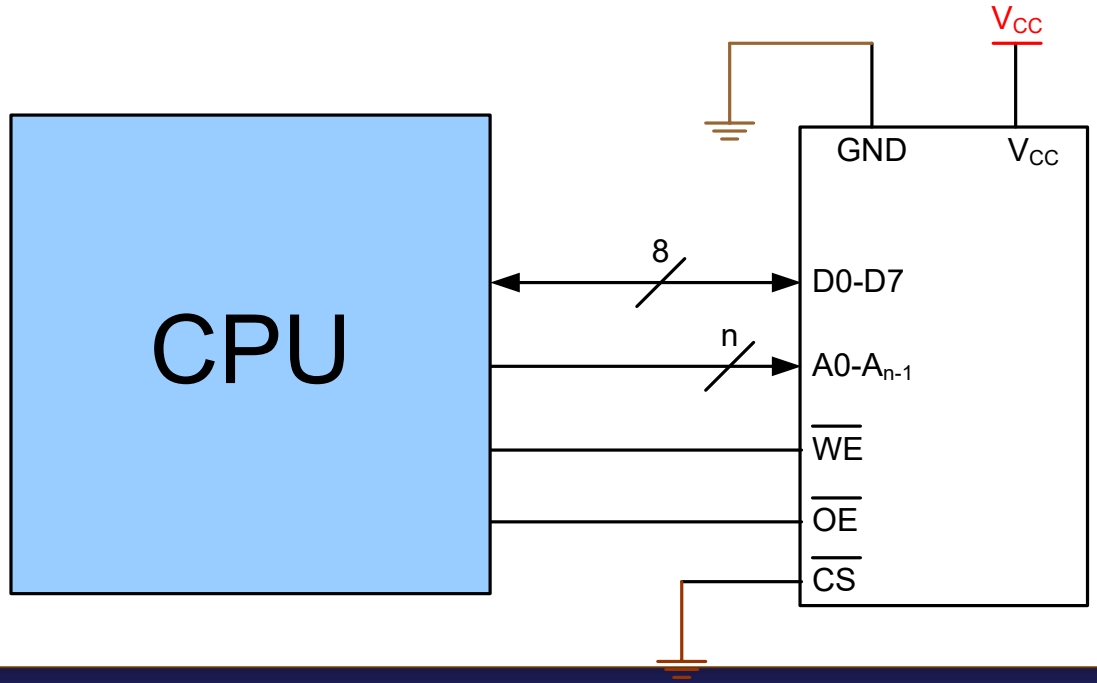
- Memory pin out

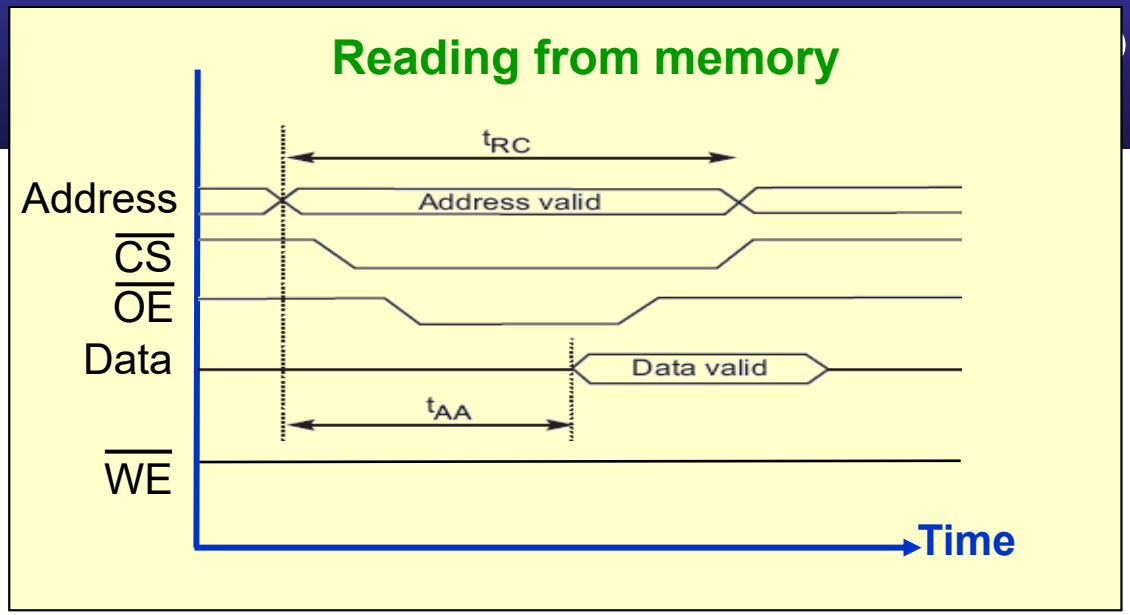


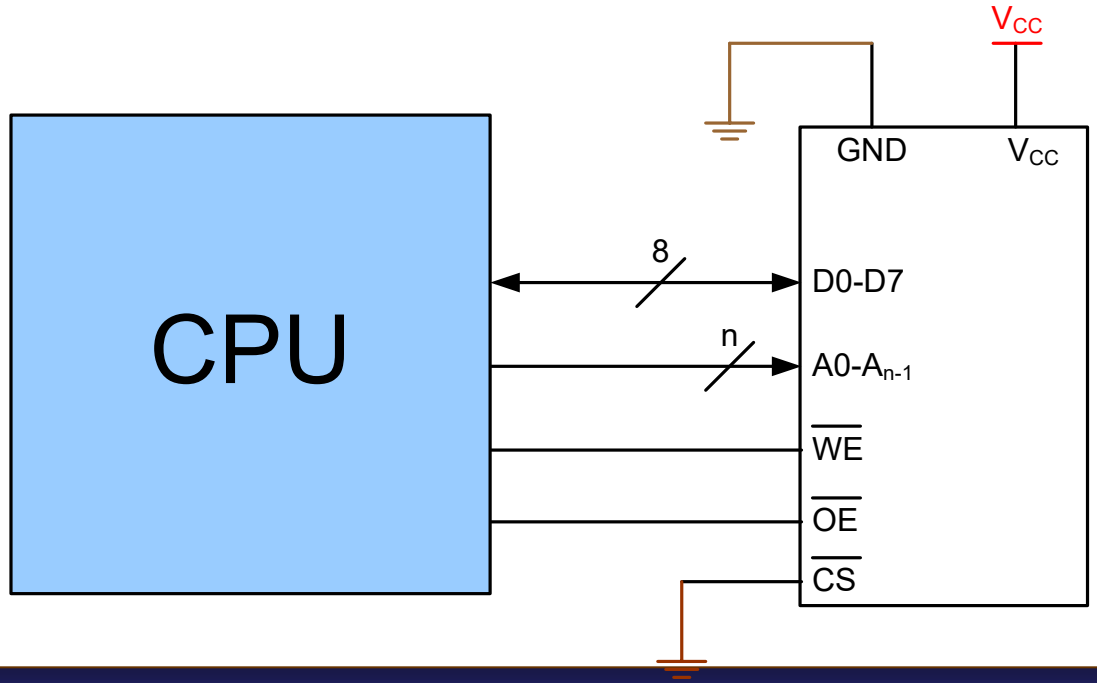
Connecting memory to CPU





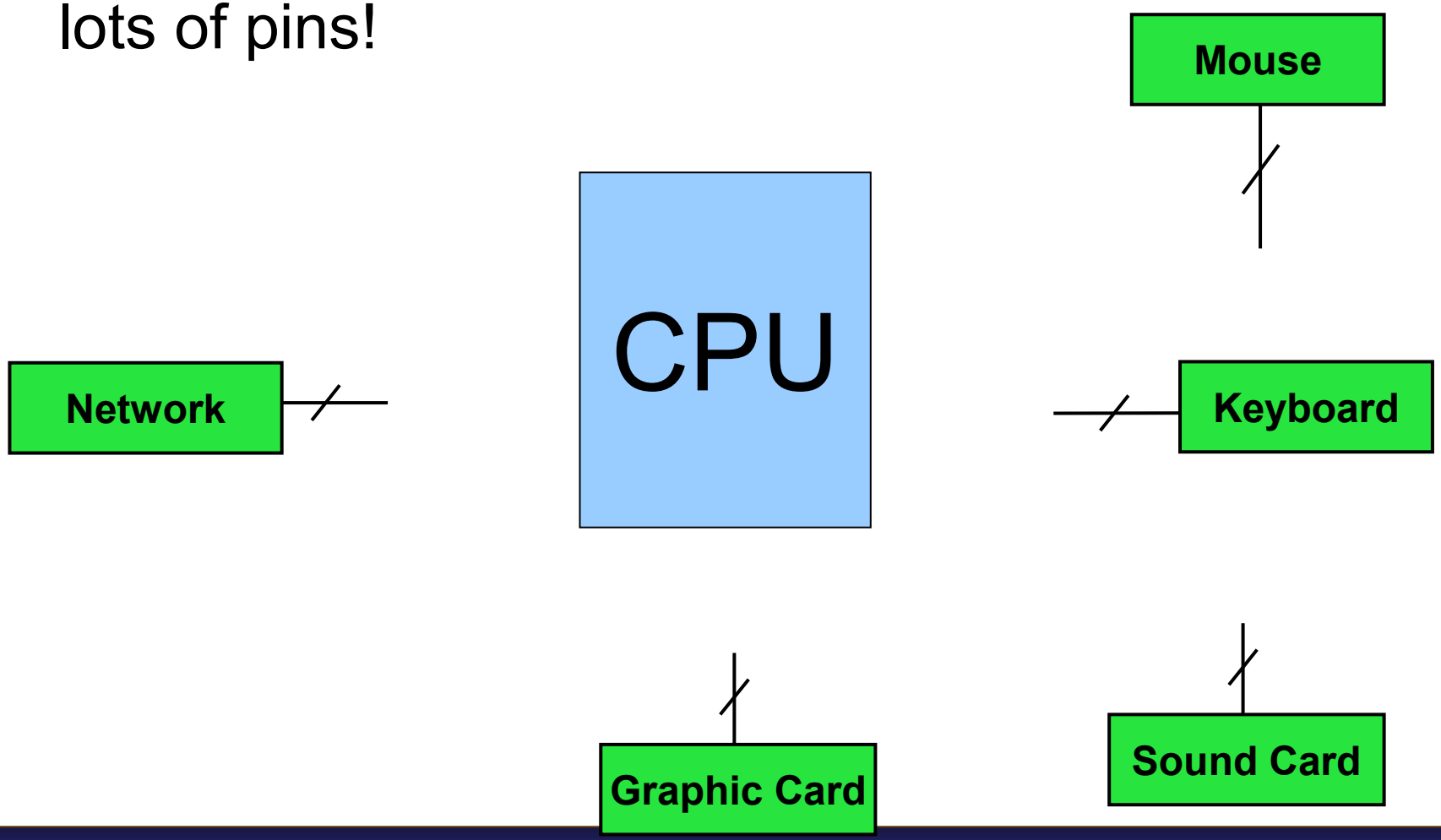




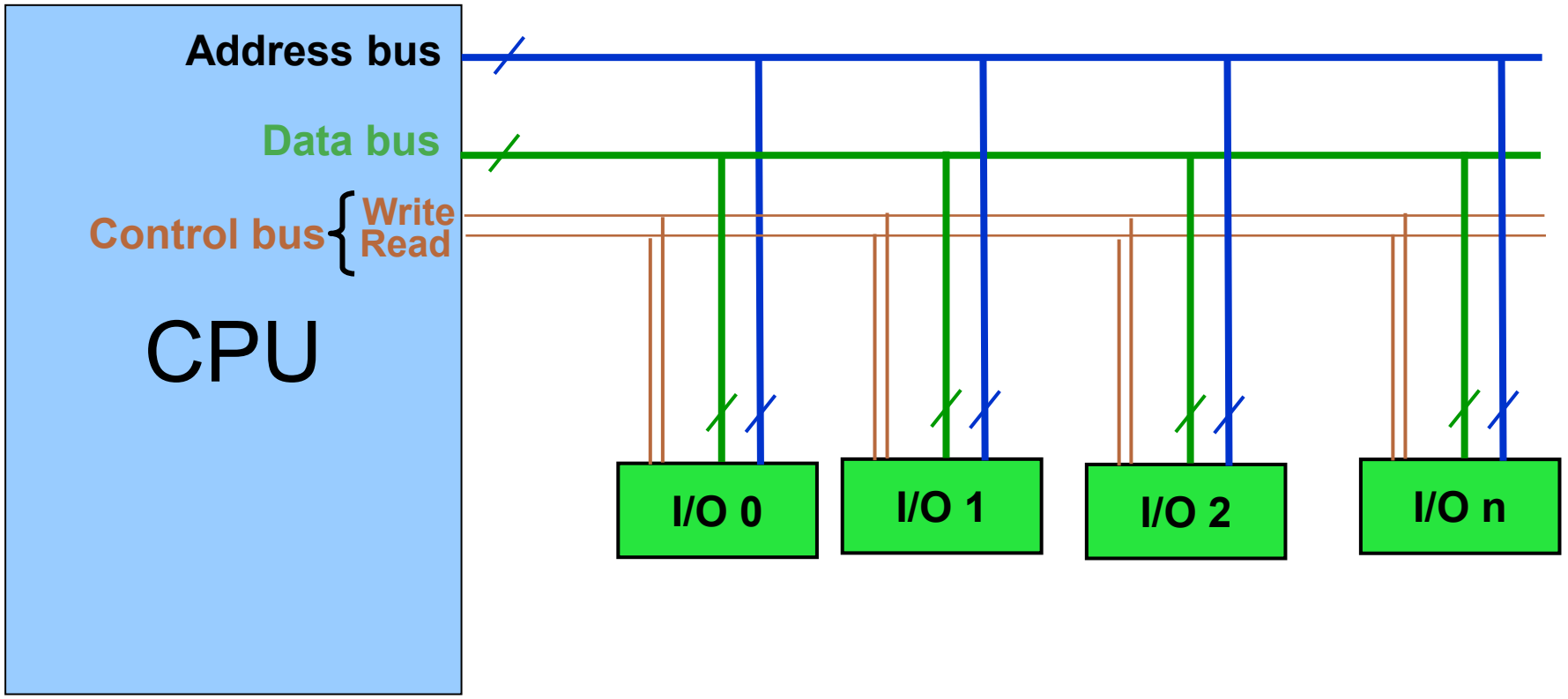


Connecting I/Os to CPU

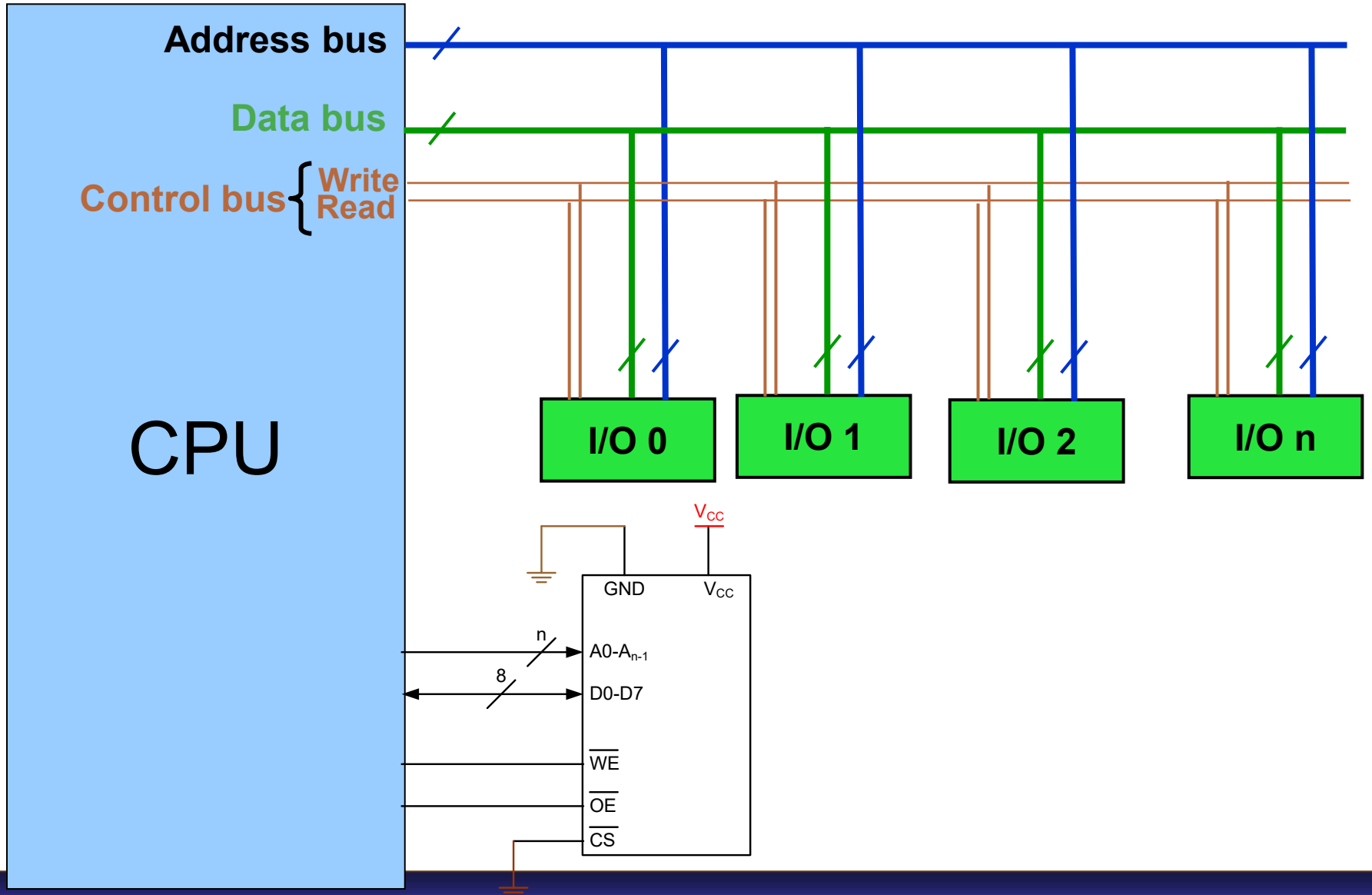
- CPU should have lots of pins!



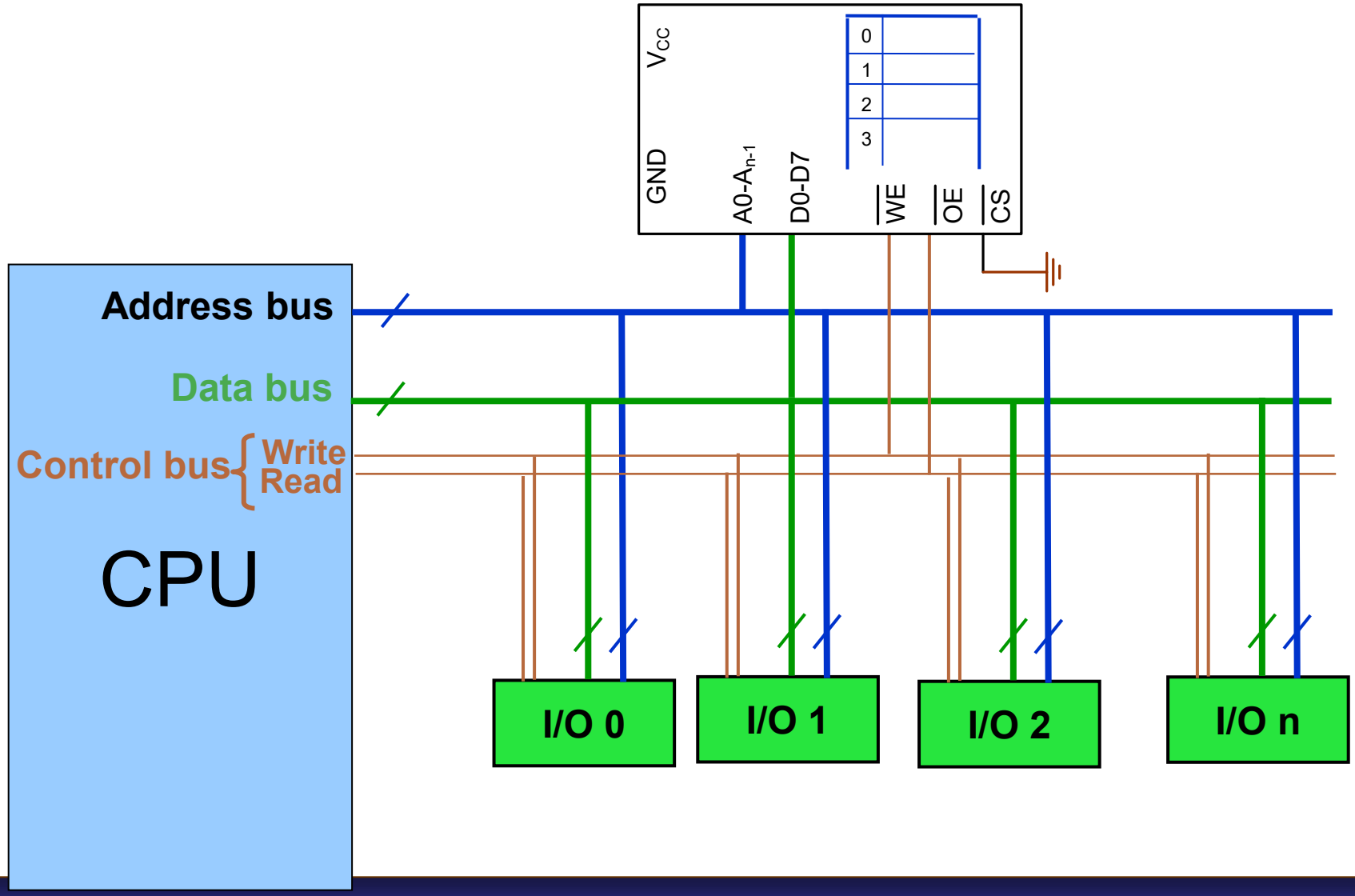
Connecting I/Os to CPU using bus



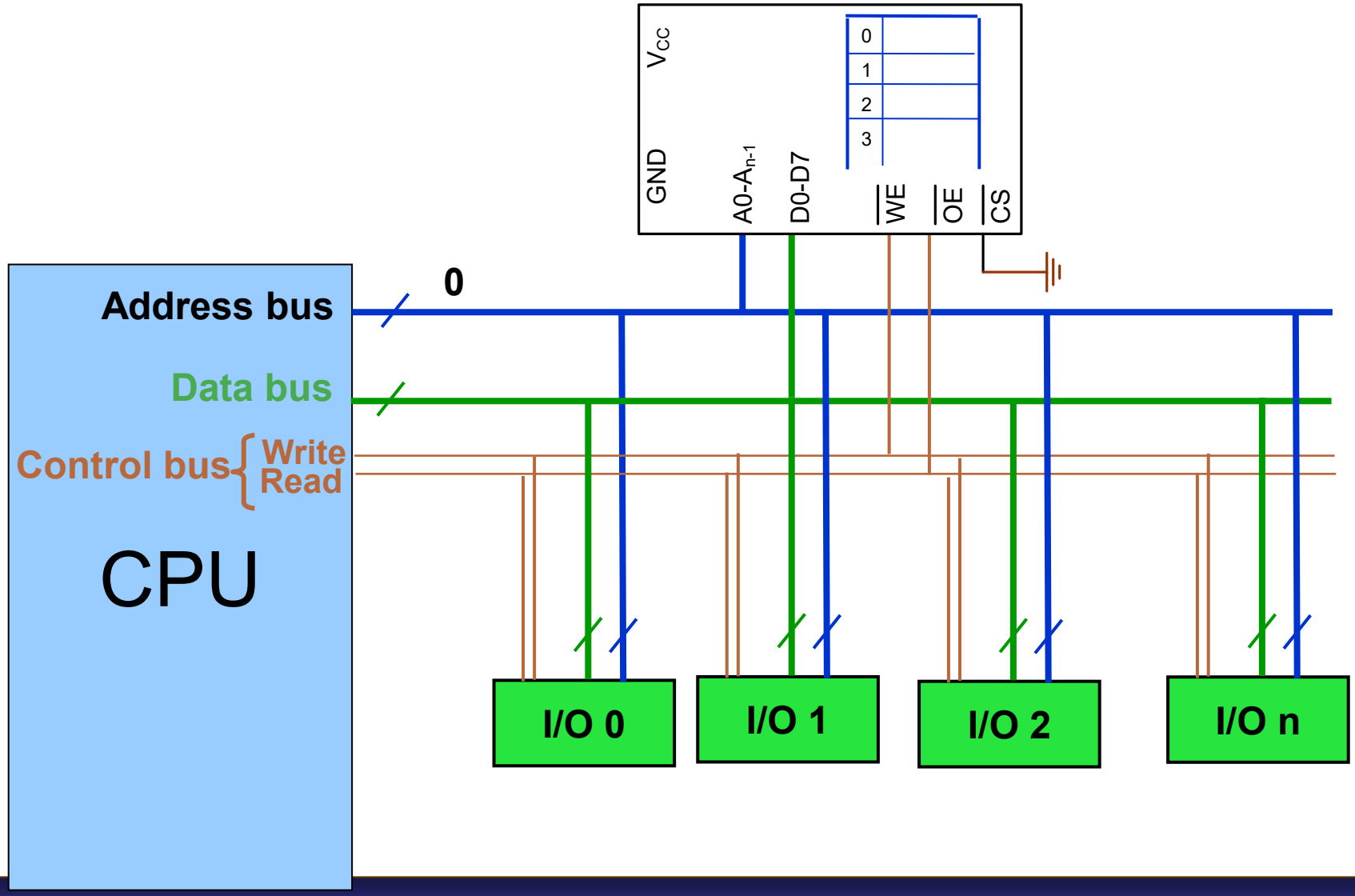
Connecting I/Os and Memory to CPU



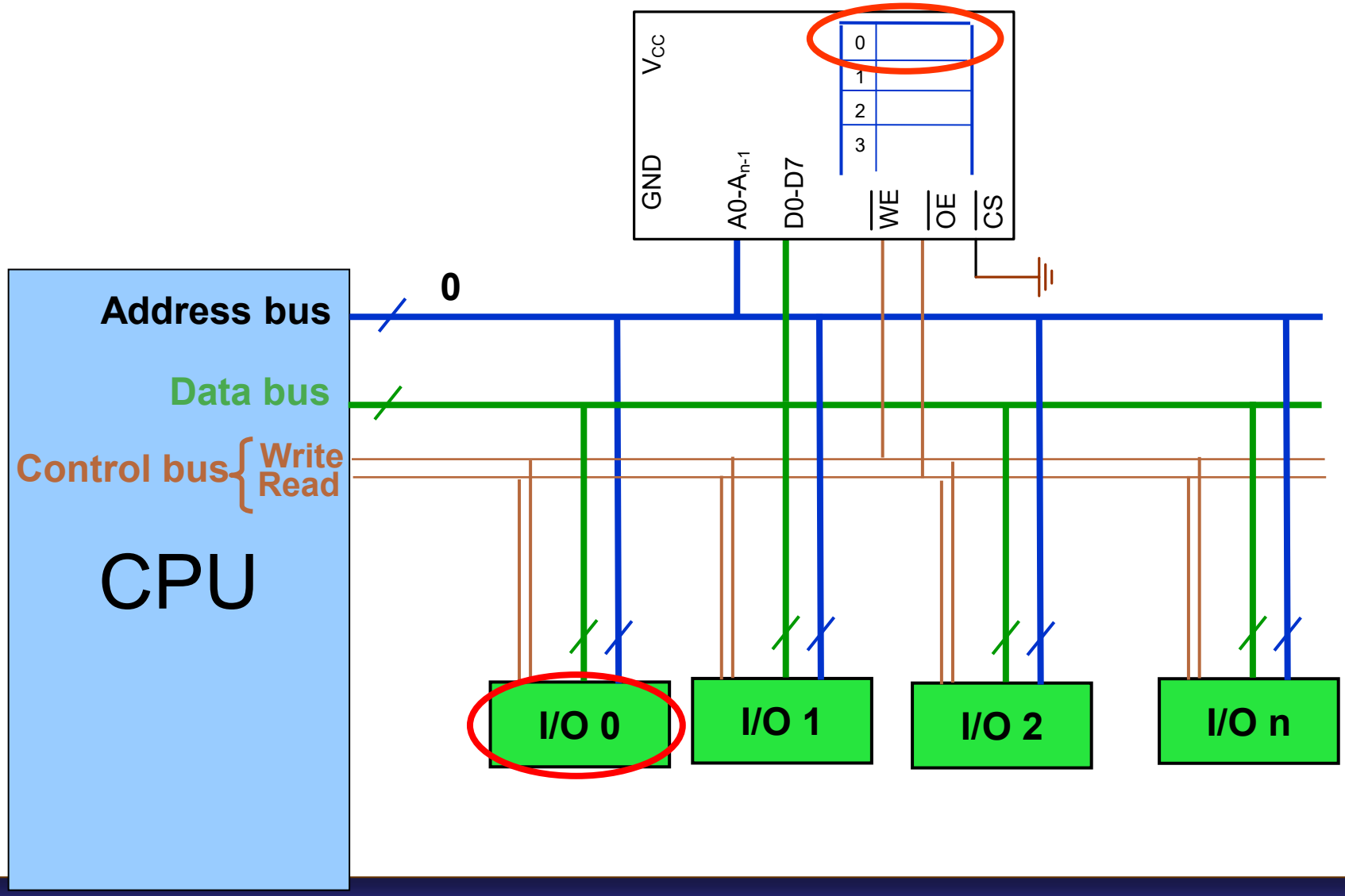
Connecting I/Os and memory to CPU using bus



Connecting I/Os and memory to CPU using bus

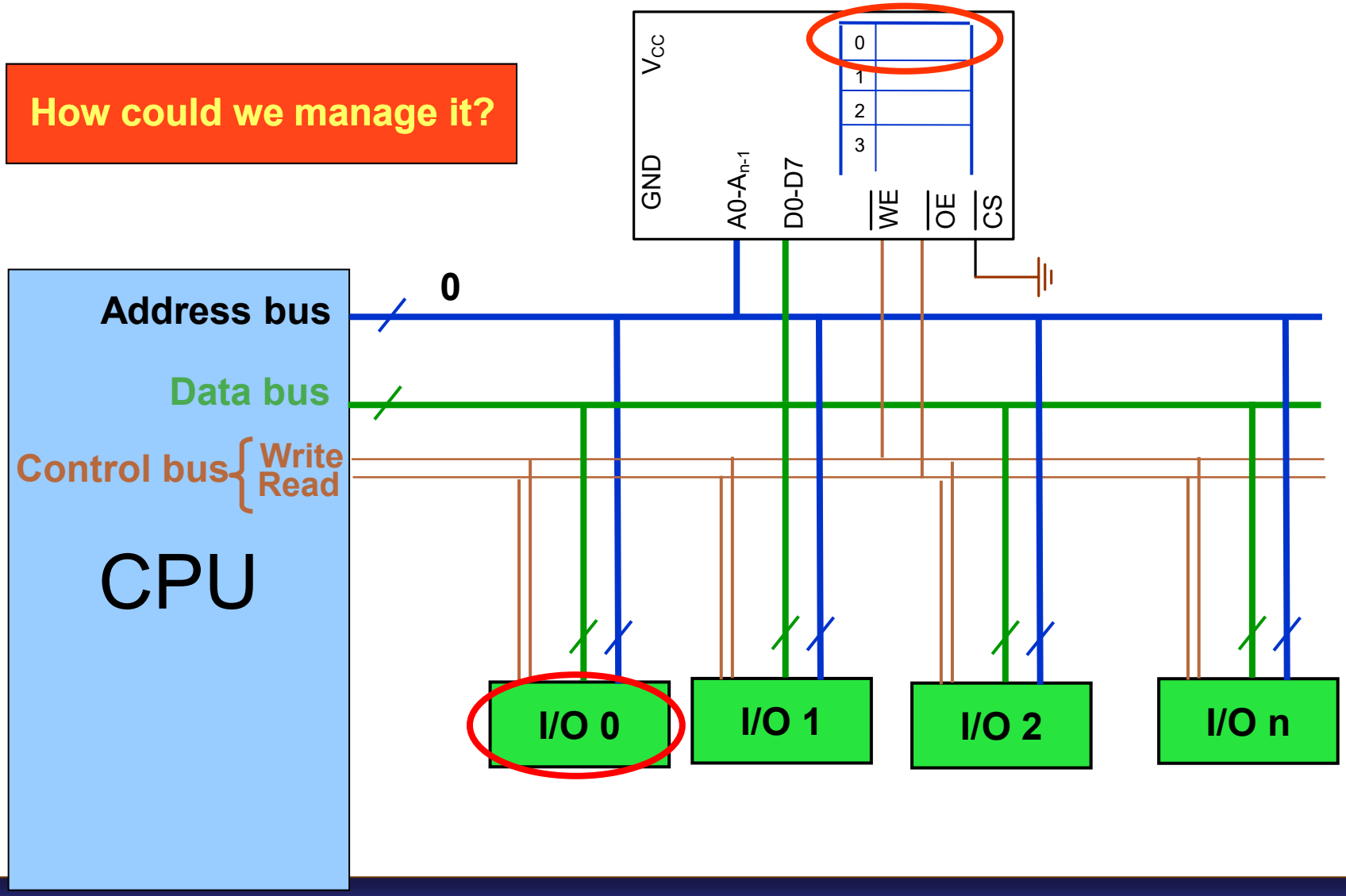


Connecting I/Os and memory to CPU using bus

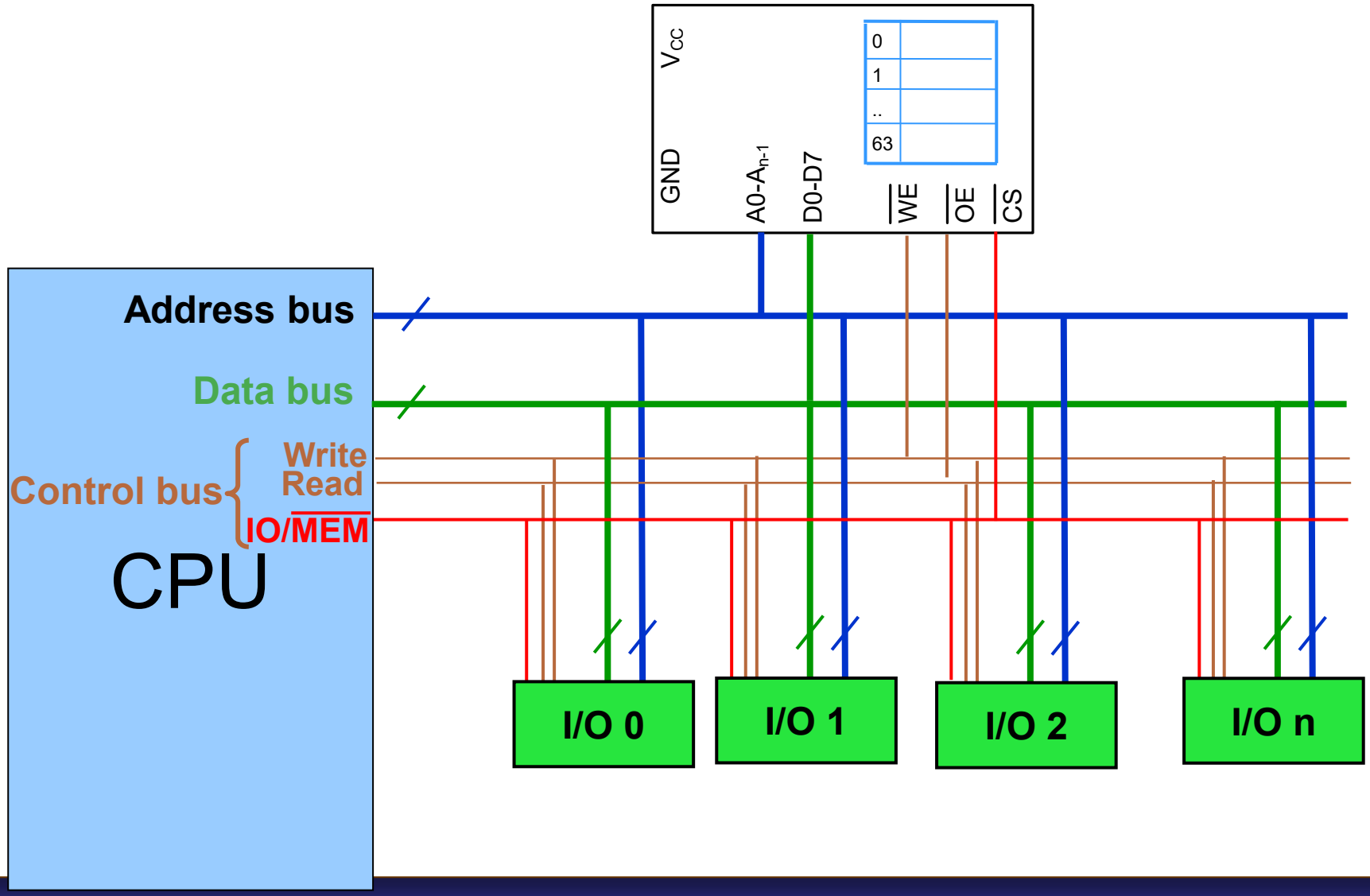


Connecting I/Os and memory to CPU using bus

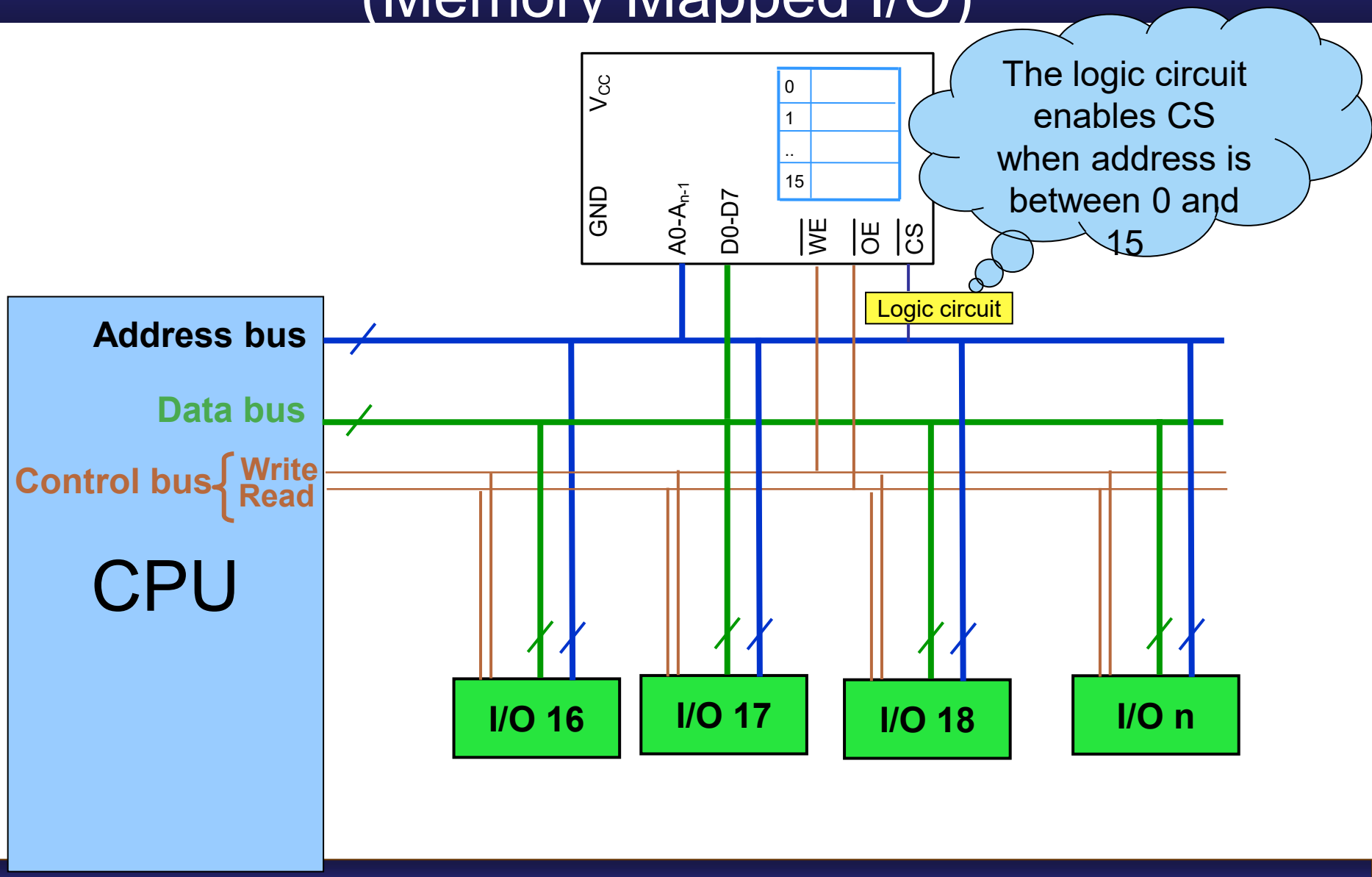
How could we manage it?



Connecting I/Os and Memory to CPU using bus (Peripheral I/O)

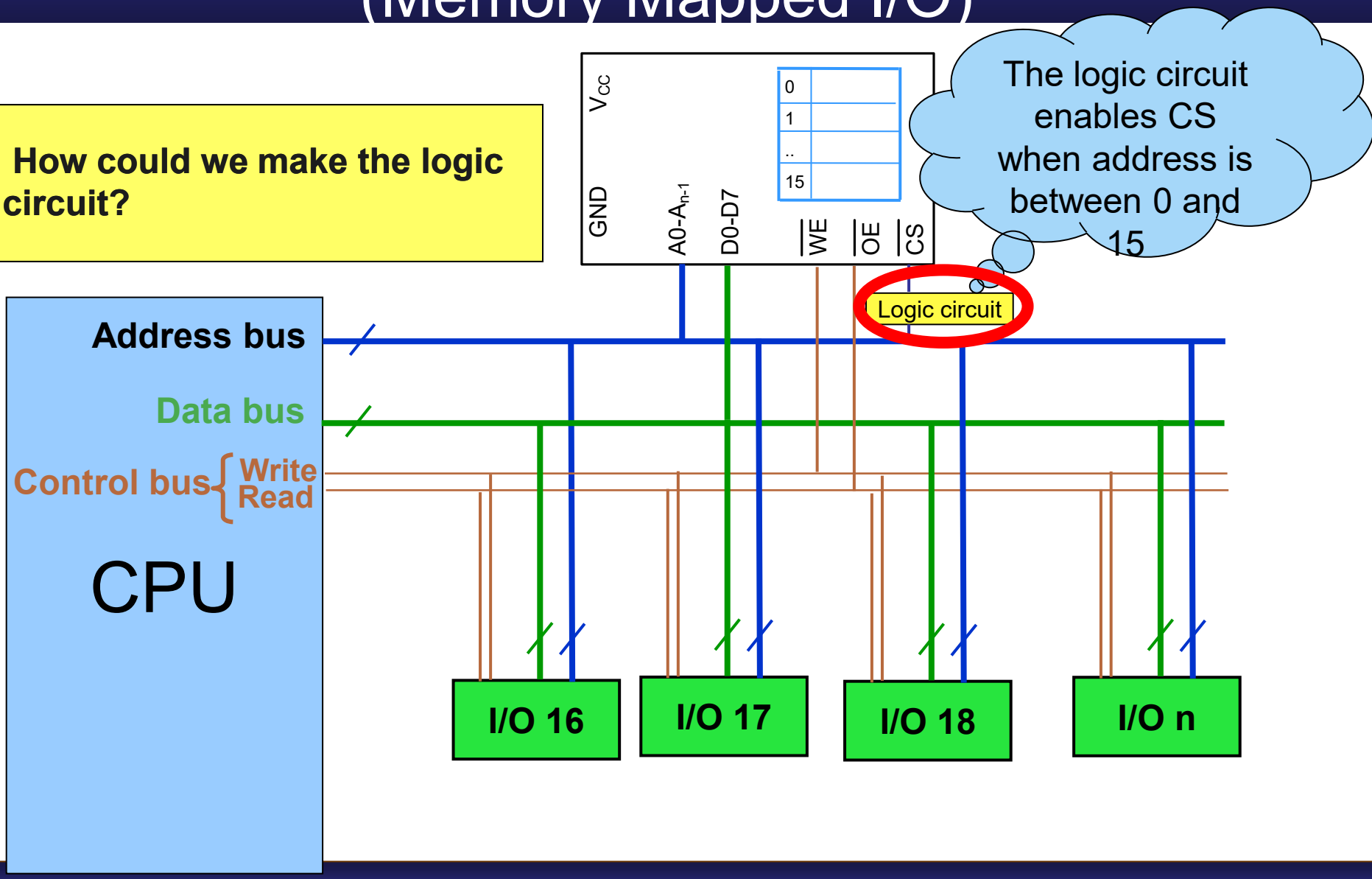


Connecting I/Os and Memory to CPU using bus (Memory Mapped I/O)



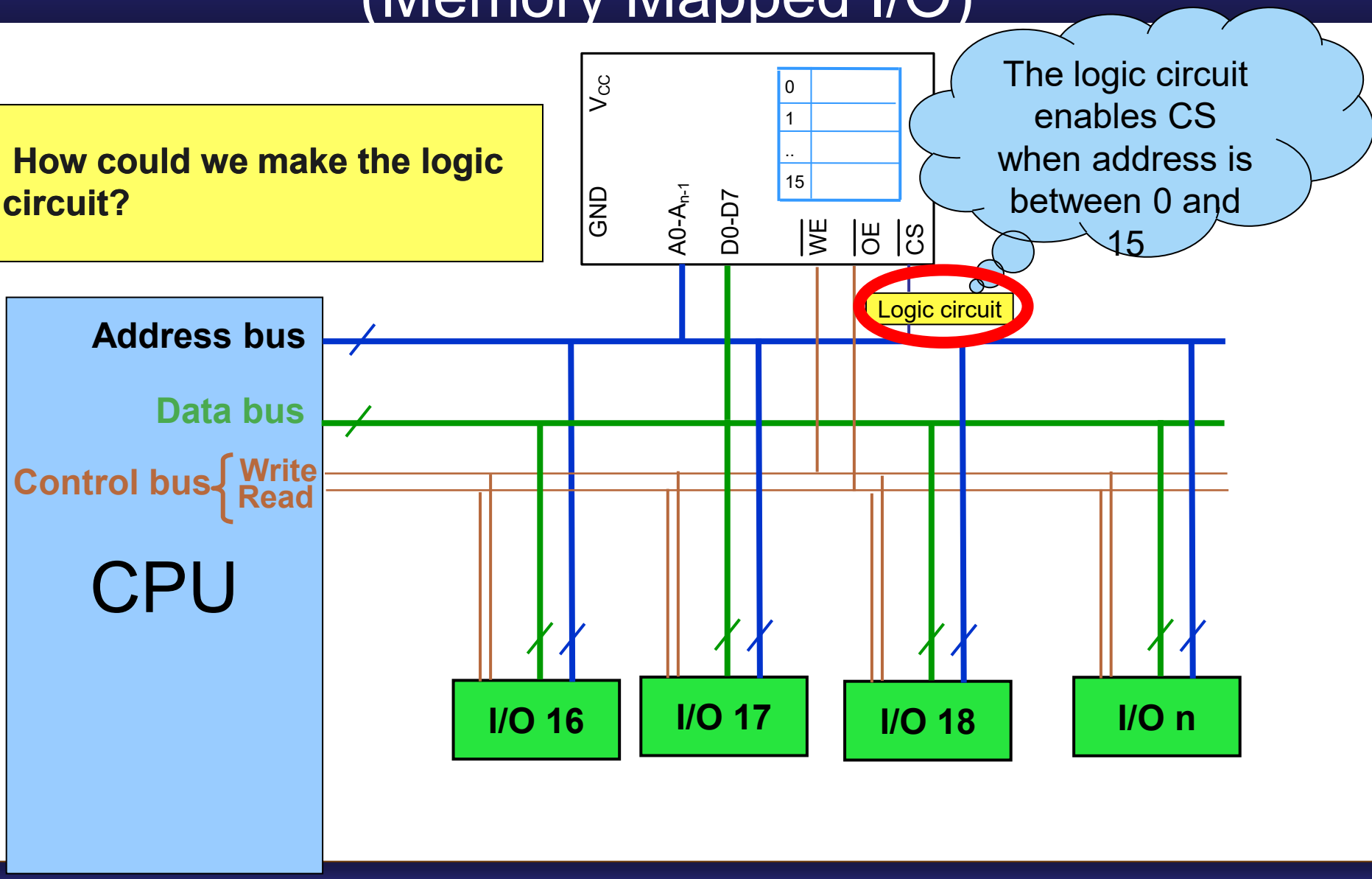
Connecting I/Os and Memory to CPU using bus (Memory Mapped I/O)

How could we make the logic circuit?



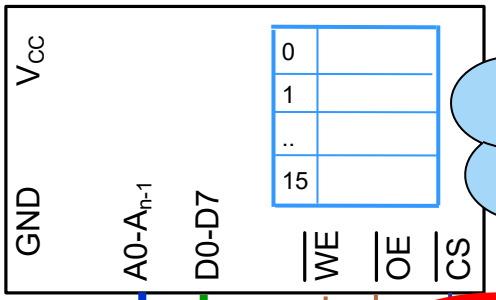
Connecting I/Os and Memory to CPU using bus (Memory Mapped I/O)

How could we make the logic circuit?



Connecting I/Os and Memory to CPU using bus (Memory Mapped I/O)

How could we make the logic circuit?



The logic circuit enables CS when address is between 0 and 15

Logic circuit

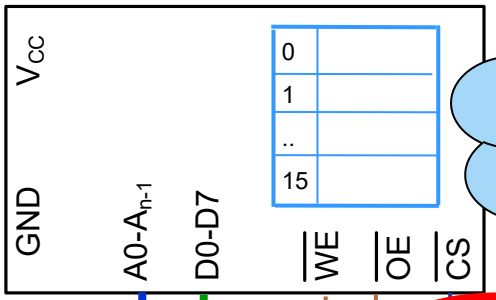
Address bus

Solution

1. Write the address range in binary

Connecting I/Os and Memory to CPU using bus (Memory Mapped I/O)

How could we make the logic circuit?



The logic circuit enables CS when address is between 0 and 15

Logic circuit

Address bus

Solution

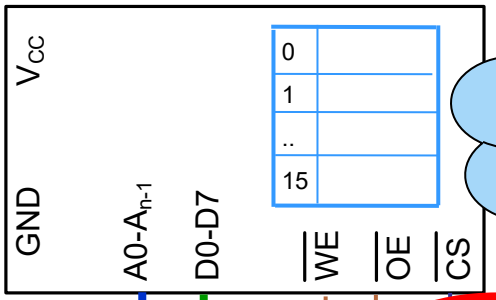
1. Write the address range in binary

From address 0 ➔ 0 0 0 0 0 0 0 0

To address15 ➔ 0 0 0 0 1 1 1 1

Connecting I/Os and Memory to CPU using bus (Memory Mapped I/O)

How could we make the logic circuit?



The logic circuit enables CS when address is between 0 and 15

Logic circuit

Address bus

Co

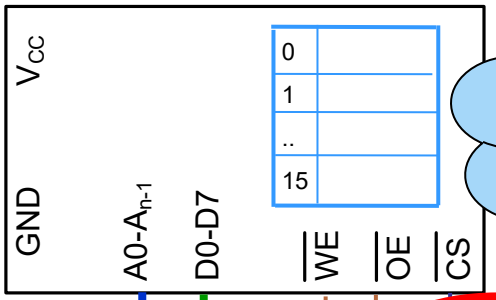
Solution

1. Write the address range in binary
2. Separate the fixed part of address

	a7	a6	a5	a4	a3	a2	a1	a0
From address 0 ➔	0	0	0	0	0	0	0	0
To address15 ➔	0	0	0	0	1	1	1	1

Connecting I/Os and Memory to CPU using bus (Memory Mapped I/O)

How could we make the logic circuit?



The logic circuit enables CS when address is between 0 and 15

Logic circuit

Address bus

Co

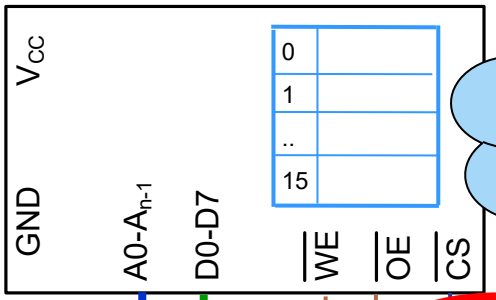
Solution

1. Write the address range in binary
2. Separate the fixed part of address
3. Using a NAND, design a logic circuit whose output activates when the fixed address is given to it.

	a7	a6	a5	a4	a3	a2	a1	a0
From address 0 ➔	0	0	0	0	0	0	0	0
To address15 ➔	0	0	0	0	1	1	1	1

Connecting I/Os and Memory to CPU using bus (Memory Mapped I/O)

How could we make the logic circuit?



The logic circuit enables CS when address is between 0 and 15

Logic circuit

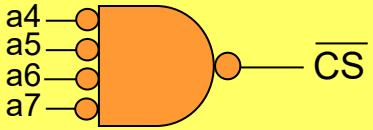
Address bus

Co

Solution

1. Write the address range in binary
2. Separate the fixed part of address
3. Using a NAND, design a logic circuit whose output activates when the fixed address is given to it.

	a7	a6	a5	a4	a3	a2	a1	a0
From address 0 ➔	0	0	0	0	0	0	0	0
To address15 ➔	0	0	0	0	1	1	1	1



Another example for address decoder

- Design an address decoder for address of 300H to 3FFH.

Another example for address decoder

- Design an address decoder for address of 300H to 3FFH.

Solution

1. Write the address range in binary

Another example for address decoder

- Design an address decoder for address of 300H to 3FFH.

Solution

1. Write the address range in binary
2. Separate the fixed part of address

	a11	a10	a9	a8	a7	a6	a5	a4	a3	a2	a1	a0
From address 300H ➔	0	0	1	1	0	0	0	0	0	0	0	0
To address 3FFH ➔	0	0	1	1	1	1	1	1	1	1	1	1

Another example for address decoder

- Design an address decoder for address of 300H to 3FFH.

Solution

1. Write the address range in binary
2. Separate the fixed part of address

	a11	a10	a9	a8	a7	a6	a5	a4	a3	a2	a1	a0
From address 300H ➔	0	0	1	1	0	0	0	0	0	0	0	0
To address 3FFH ➔	0	0	1	1	1	1	1	1	1	1	1	1

Another example for address decoder

- Design an address decoder for address of 300H to 3FFH.

Solution

1. Write the address range in binary
2. Separate the fixed part of address
3. Design the logic circuit.

	a11	a10	a9	a8	a7	a6	a5	a4	a3	a2	a1	a0
From address 300H ➔	0	0	1	1	0	0	0	0	0	0	0	0
To address 3FFH ➔	0	0	1	1	1	1	1	1	1	1	1	1

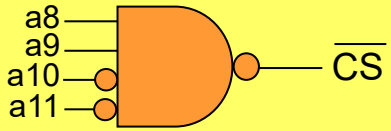
Another example for address decoder

- Design an address decoder for address of 300H to 3FFH.

Solution

1. Write the address range in binary
2. Separate the fixed part of address
3. Design the logic circuit.

	a11	a10	a9	a8	a7	a6	a5	a4	a3	a2	a1	a0
From address 300H ➔	0	0	1	1	0	0	0	0	0	0	0	0
To address 3FFH ➔	0	0	1	1	1	1	1	1	1	1	1	1



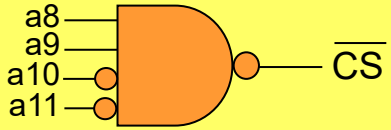
Another example for address decoder

- Design an address decoder for address of 300H to 3FFH.

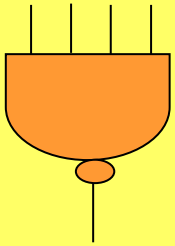
Solution

1. Write the address range in binary
2. Separate the fixed part of address
3. Design the logic circuit.

	a11	a10	a9	a8	a7	a6	a5	a4	a3	a2	a1	a0
From address 300H ➔	0	0	1	1	0	0	0	0	0	0	0	0
To address 3FFH ➔	0	0	1	1	1	1	1	1	1	1	1	1



An easy way of designing



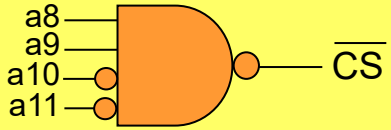
Another example for address decoder

- Design an address decoder for address of 300H to 3FFH.

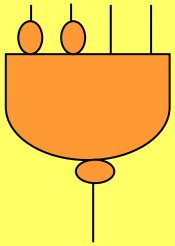
Solution

1. Write the address range in binary
2. Separate the fixed part of address
3. Design the logic circuit.

	a11	a10	a9	a8	a7	a6	a5	a4	a3	a2	a1	a0
From address 300H ➔	0	0	1	1	0	0	0	0	0	0	0	0
To address 3FFH ➔	0	0	1	1	1	1	1	1	1	1	1	1

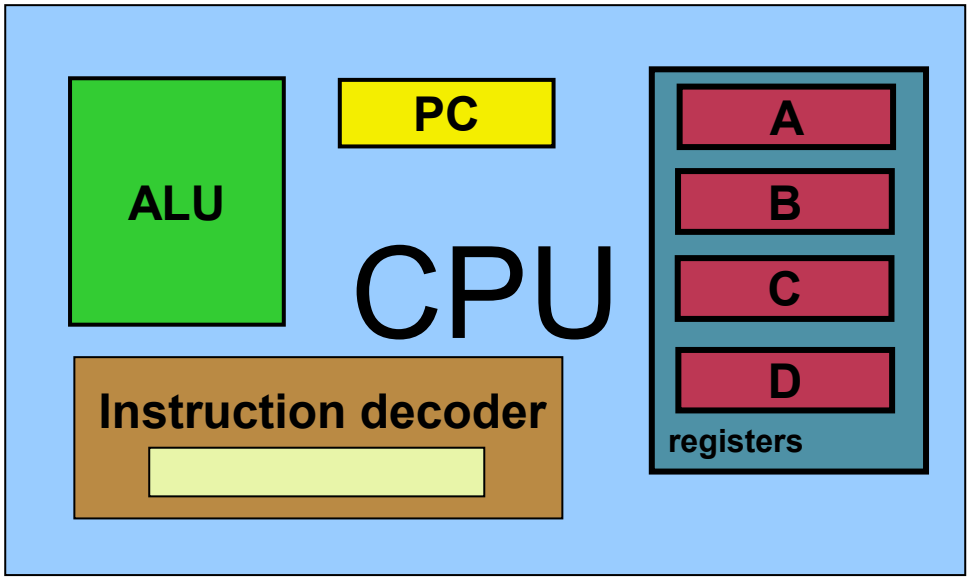


An easy way of designing

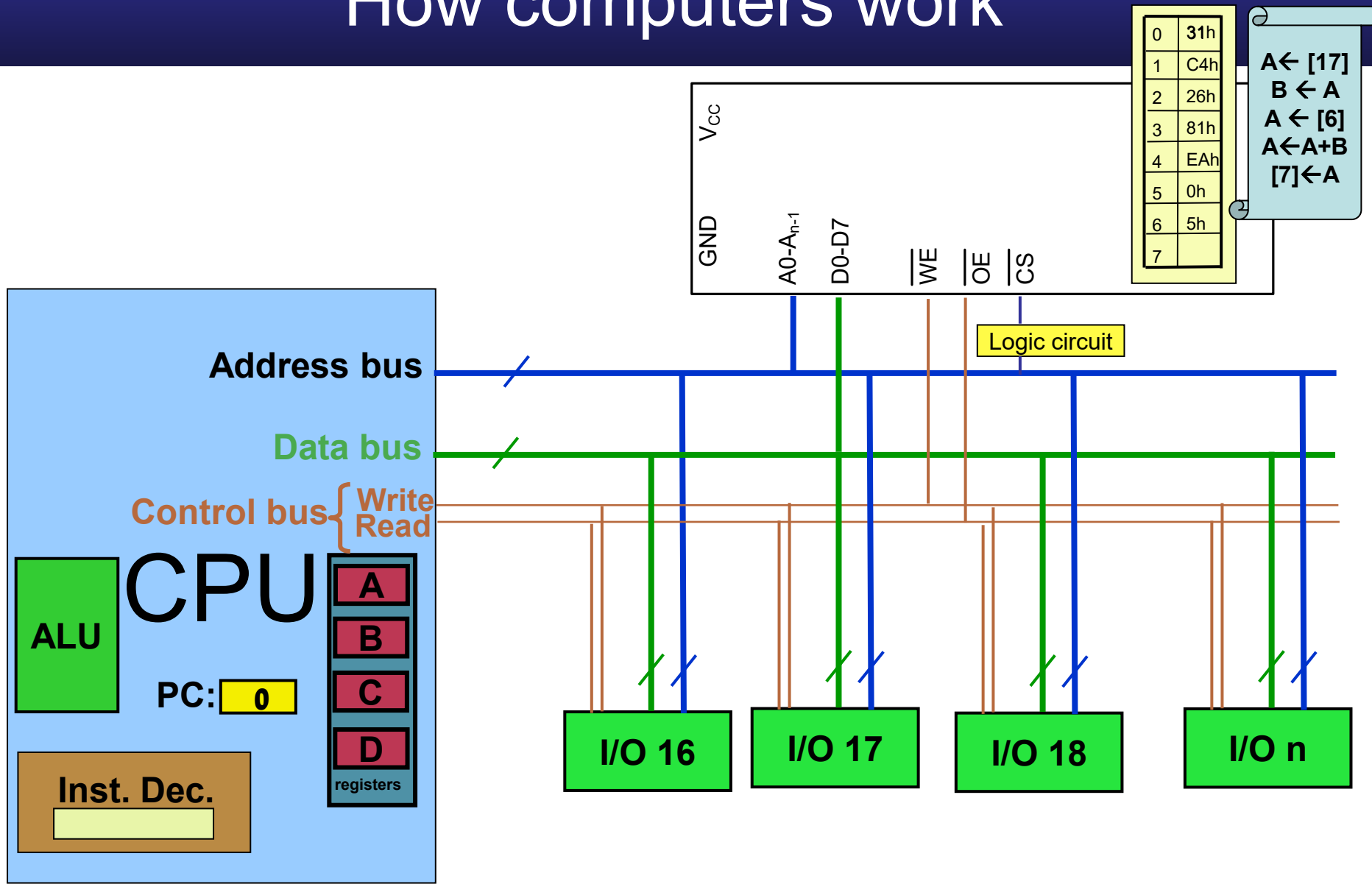


Inside the CPU

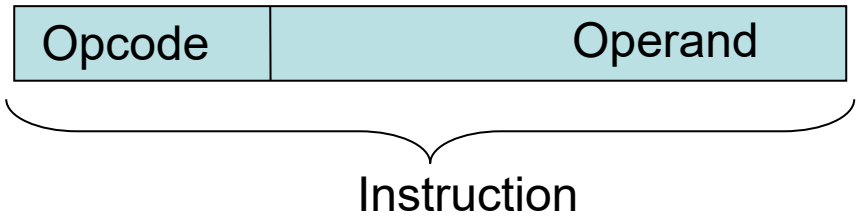
- PC (Program Counter)
- Instruction decoder
- ALU (Arithmetic Logic Unit)
- Registers

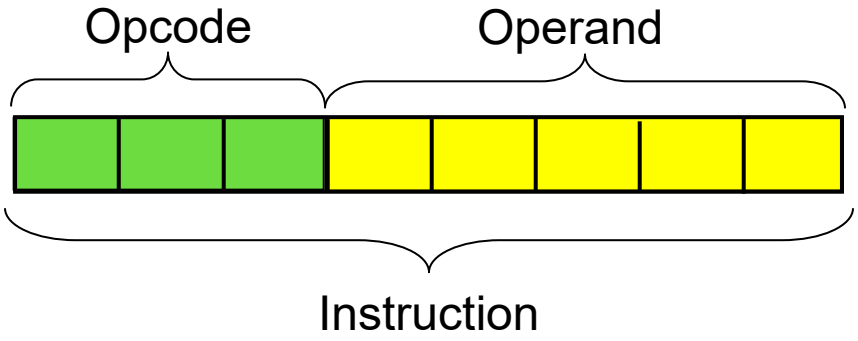


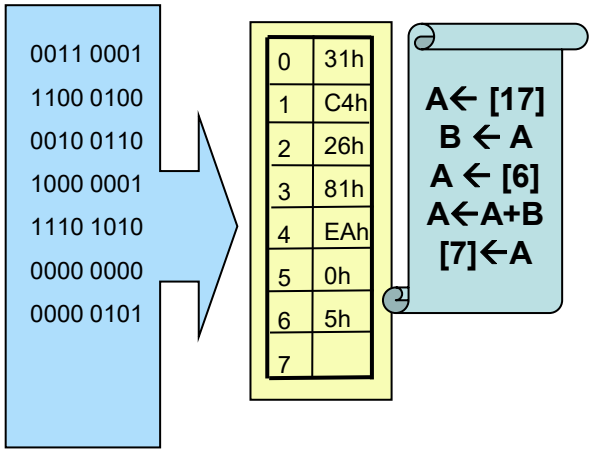
How computers work



How Instruction decoder works





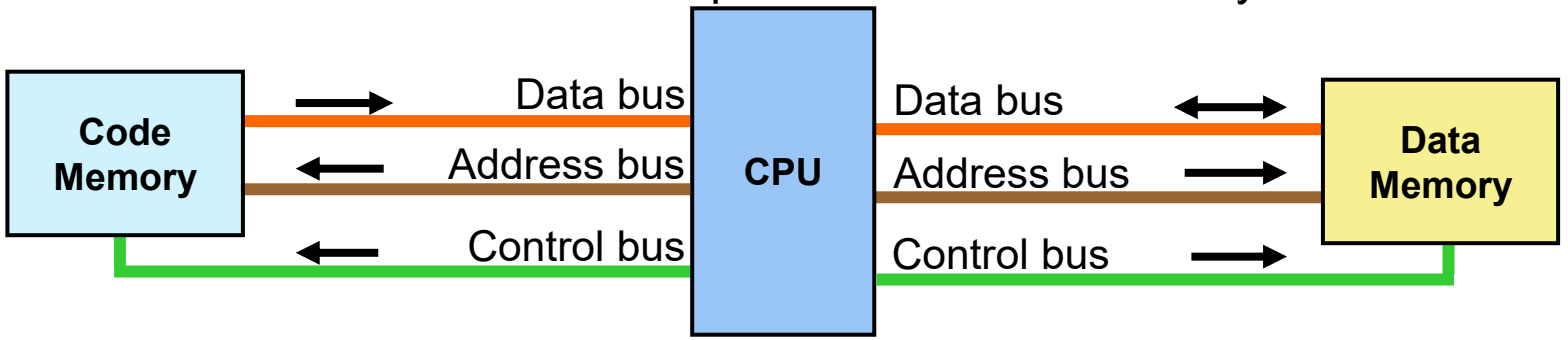


Operation Code	Meaning
000	A ← x
001	A ← [x]
010	A ← A – register (x)
011	A ← A + x
100	A ← A + register (x)
101	A ← A – x
110	Register (x_H) ← Register (x_L)
111	[x] ← A

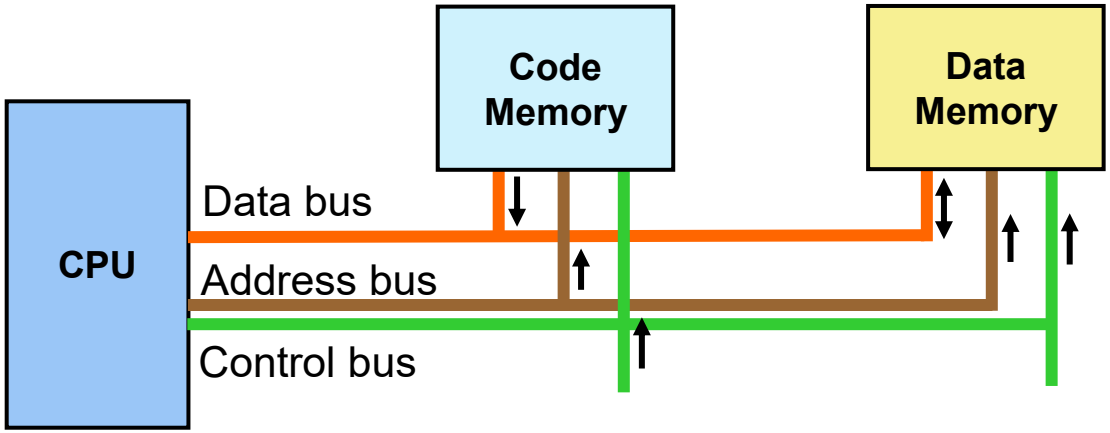
Von Neumann vs. Harvard architecture

- Harvard architecture

Harvard architecture has separate data and instruction busses, allowing transfers to be performed simultaneously on both busses.



- Von Neumann architecture



A von Neumann architecture has only one bus which is used for both data transfers and instruction fetches, and therefore data transfers and instruction fetches must be scheduled

NCS 362: Embedded Systems

Instructor:
Dr. Ahmed Shalaby http://bu.edu.eg/staff/ahmedshalaby14#

Introduction to AVR
Chapter 1

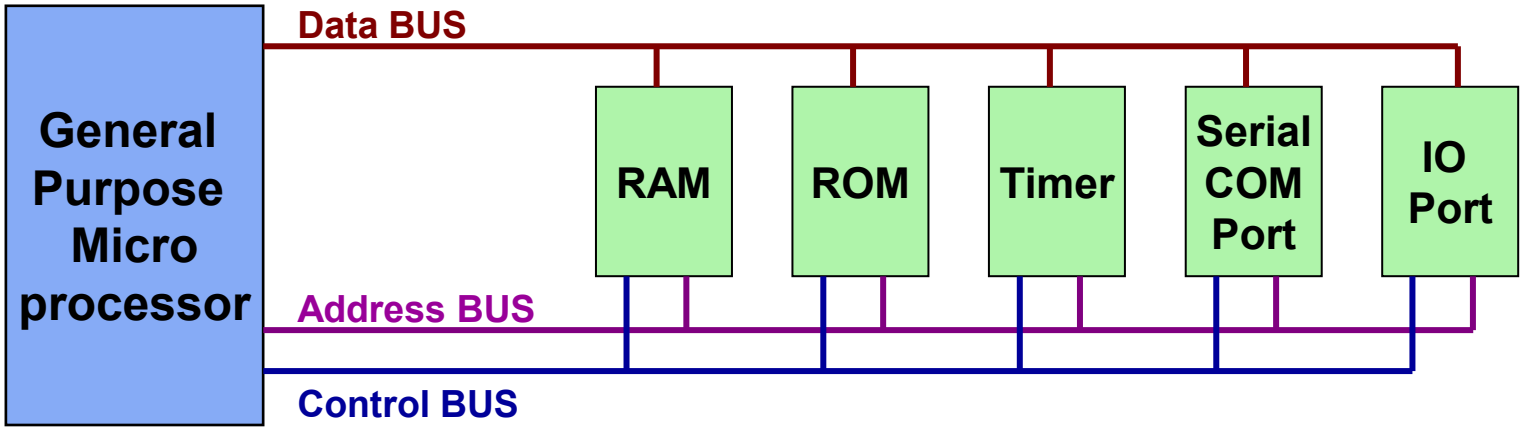
The AVR microcontroller and embedded systems
using assembly and c

Topics

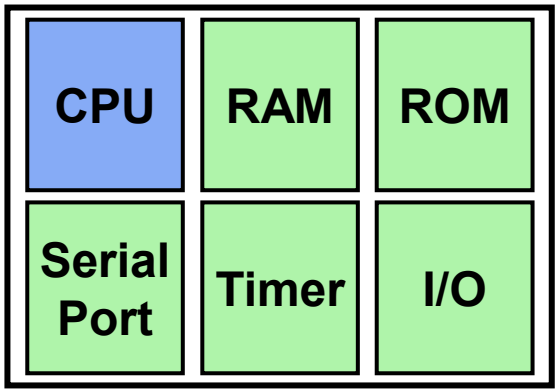
- Microcontrollers vs. Microprocessors
- Most common microcontrollers
- AVR Features
- AVR members

General Purpose Microprocessors vs. Microcontrollers

- General Purpose Microprocessors



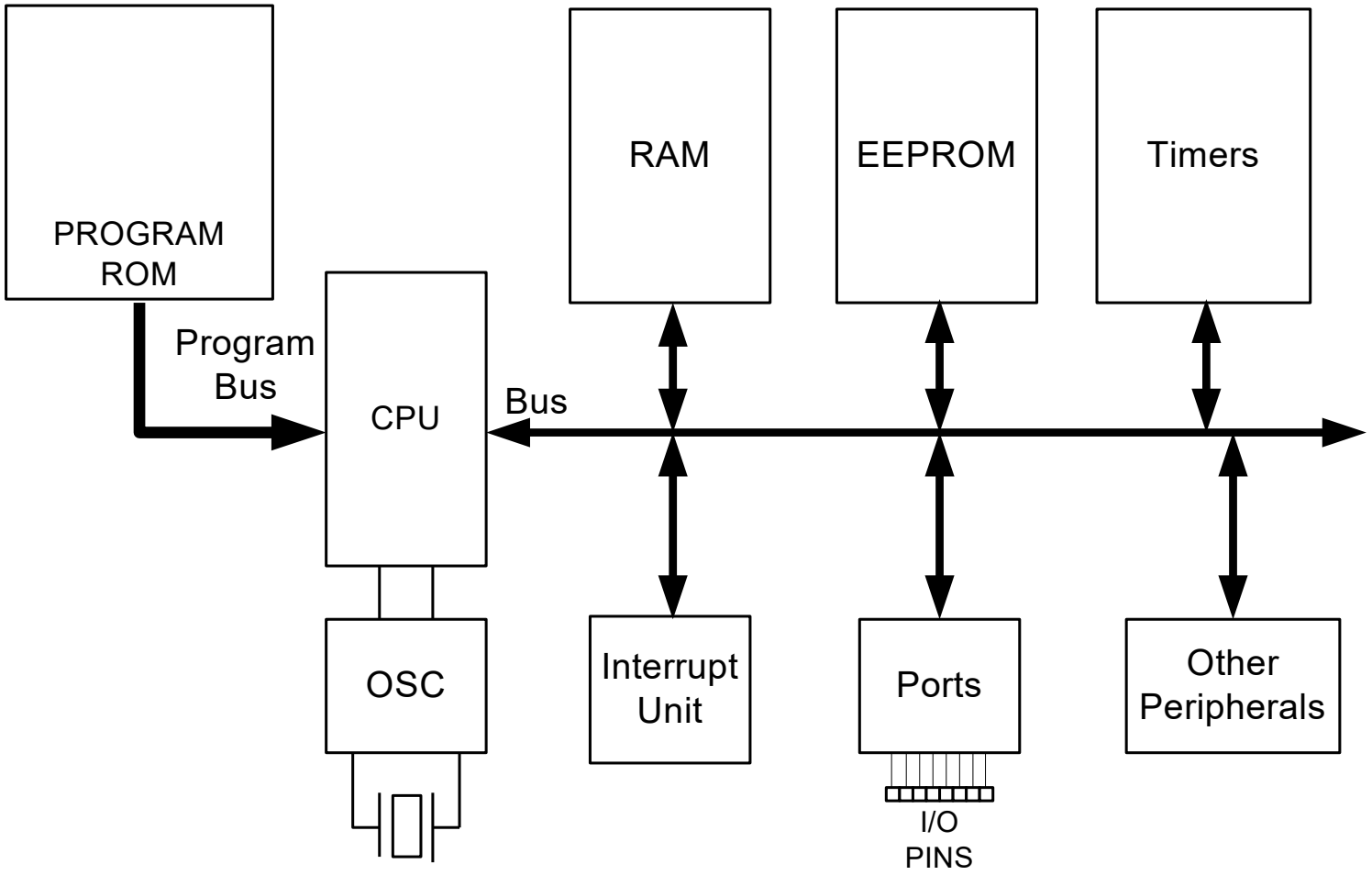
- Microcontrollers



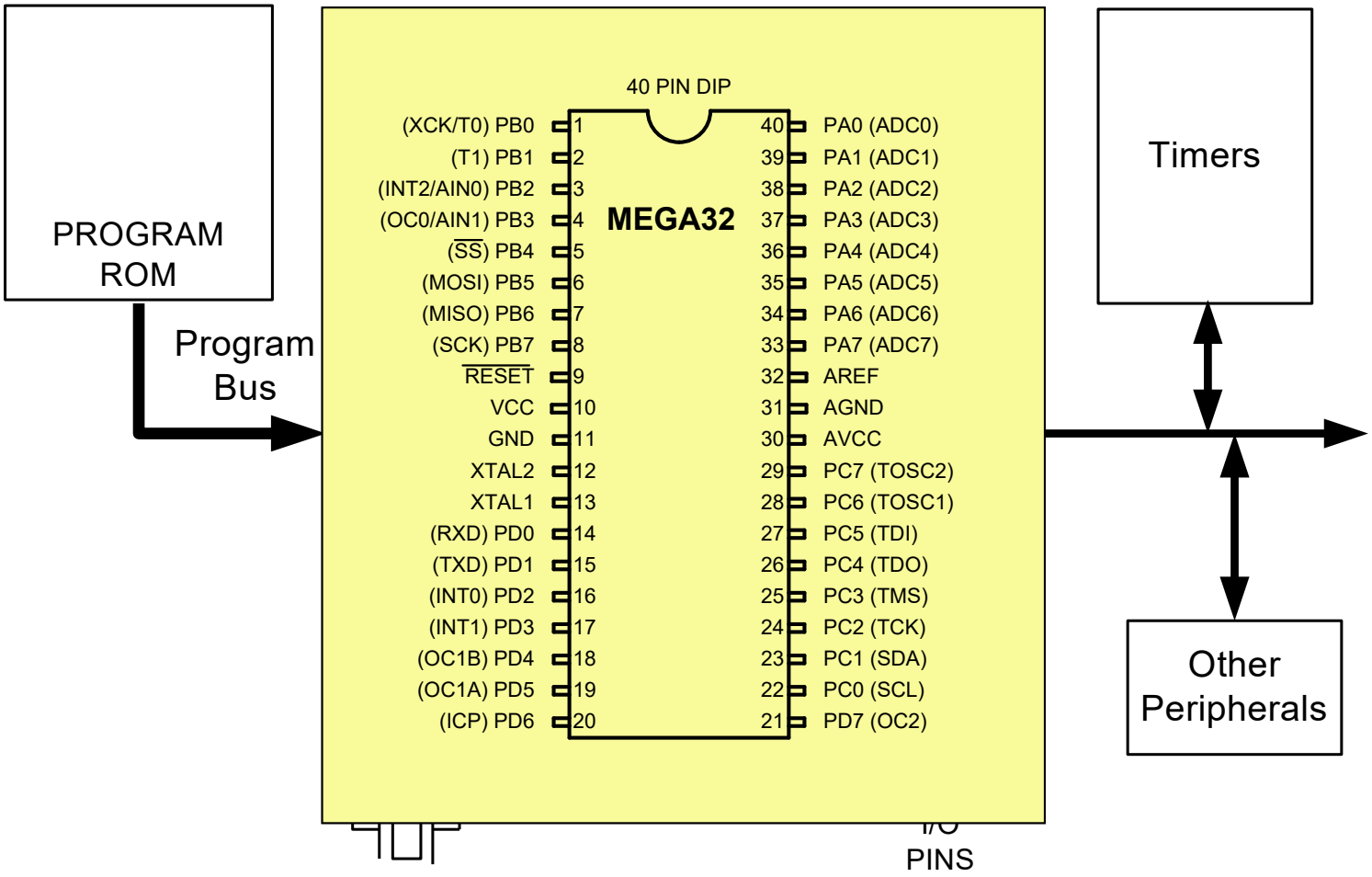
Most common microcontrollers

- 8-bit microcontrollers
  - AVR
  - PIC
  - HCS12
  - 8051
- 32-bit microcontrollers
  - ARM
  - PIC32

AVR internal architecture



AVR internal architecture



AVR different groups

- Classic AVR
  - e.g. AT90S2313, AT90S4433
- Mega
  - e.g. ATmega8, ATmega32, ATmega128
- Tiny
  - e.g. ATtiny13, ATtiny25
- Special Purpose AVR
  - e.g. AT90PWM216,AT90USB1287

AVR different groups

- Classic AVR
  - e.g. AT90S2313, AT90S4433
- Mega
  - e.g
- Tiny
  - e.g
- Spec
  - e.g. AT90PWM216,AT90USB1287

Table 1-3: Some Members of the Classic Family

Part Num	Code ROM	Data RAM	Data EEPROM	I/O pins pins	ADC	Timers	Pin numbers & Package
AT90S2313	2K	128	128	15	0	2	SOIC20,PDIP20
AT90S2323	2K	128	128	3	0	1	SOIC8,PDIP8
AT90S4433	4K	128	256	20	6	2	TQFP32,PDIP28

Notes:
1. All ROM, RAM, and EEPROM memories are in bytes.
2. Data RAM (General-Purpose RAM) is the amount of RAM available for data manipulation (scratch pad) in addition to the Registers space.

AVR different groups

- Classic AVR
  - e.g. AT90S2313, AT90S4433
- Mega
  - e.g
- Tiny
  - e.g
- Spec
  - e.g

Table 1-3: Some Members of the Classic Family

Part Num	Code ROM	Data RAM	Data EEPROM	I/O pins	ADC	Timers	Pin numbers & Package

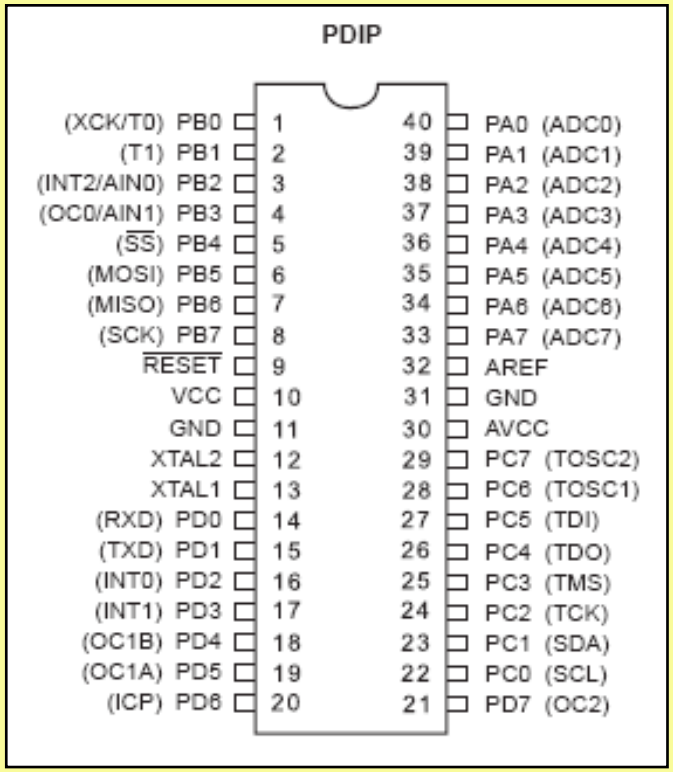
Table 1-4: Some Members of the Mega Family

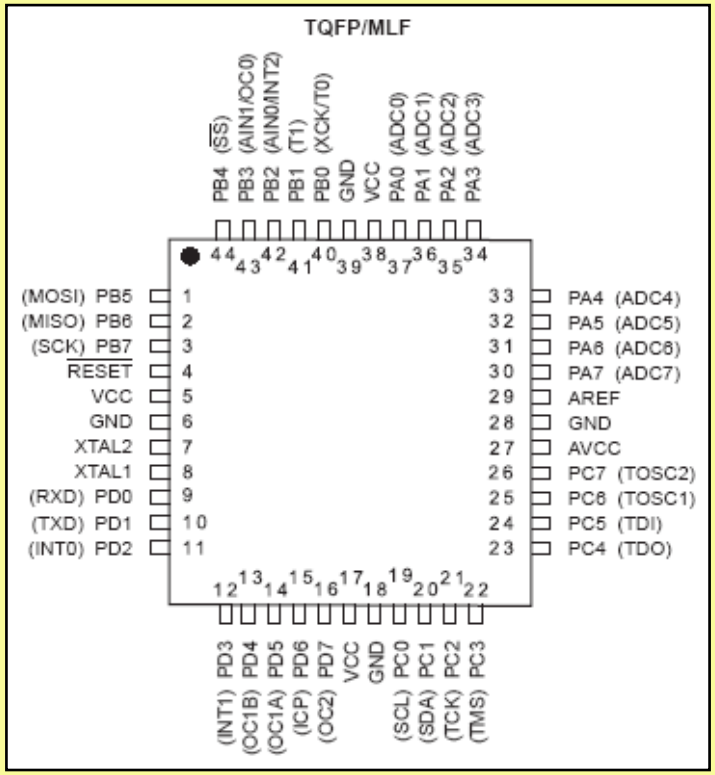
Part Num	Code ROM	Data RAM	Data EEPROM	I/O pins	ADC	Timers	Pin numbers & Package
ATmega8	8K	1K	0.5K	23	8	3	TQFP32,PDIP28
ATmega16	16K	1K	0.5K	32	8	3	TQFP44,PDIP40
ATmega32	32K	2K	1K	32	8	3	TQFP44,PDIP40
ATmega64	64K	4K	2K	54	8	4	TQFP64,MLF64
ATmega1280	128K	8K	4K	86	16	6	TQFP100,CBGA

Notes:
1. All ROM, RAM, and EEPROM memories are in bytes.
2. Data RAM (General-Purpose RAM) is the amount of RAM available for data manipulation (scratch pad) in addition to the Registers space.
3. All the above chips have USART for serial data transfer.

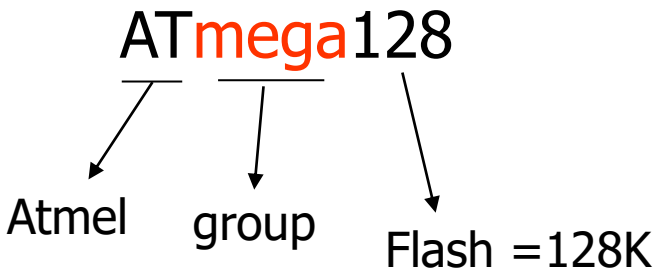
AVR different groups

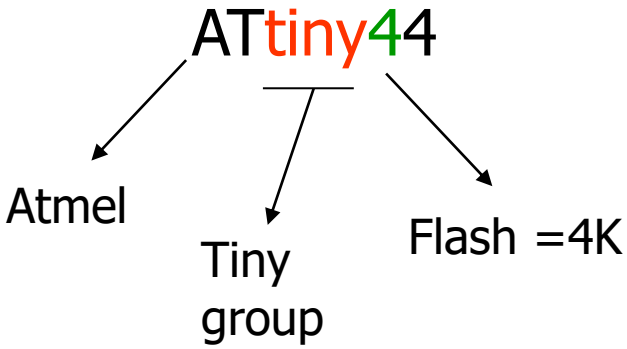
- Classic AVR
  - e.g. AT90S2313, AT90S4433
- M
  -
- Ti
  -
- Sp
  -

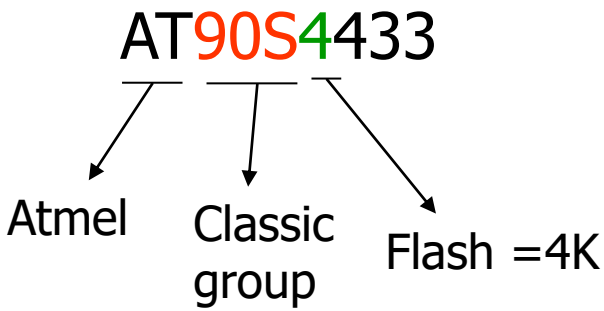




Let's get familiar with the AVR part numbers





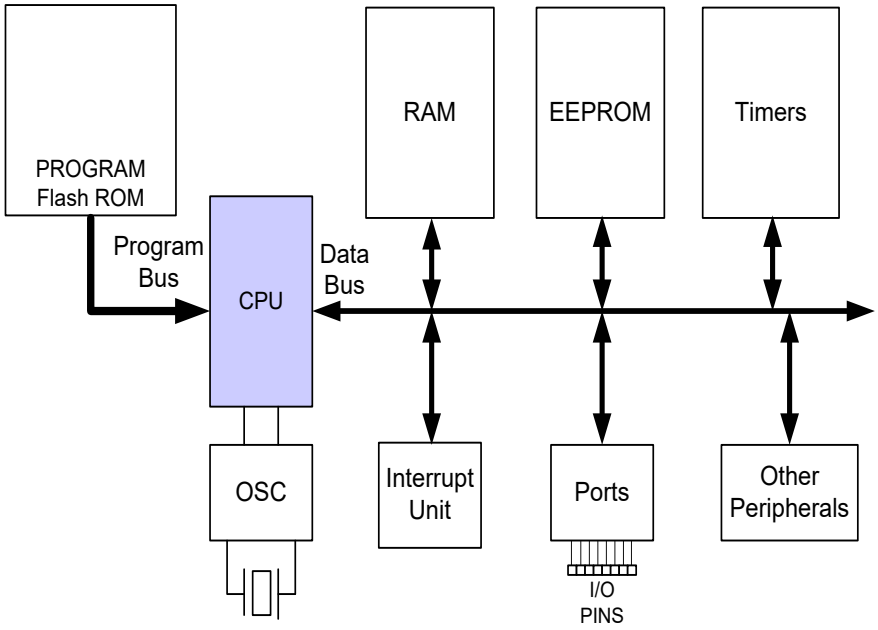


Introduction to Assembly
Chapter 2

The AVR microcontroller and embedded systems
using assembly and c

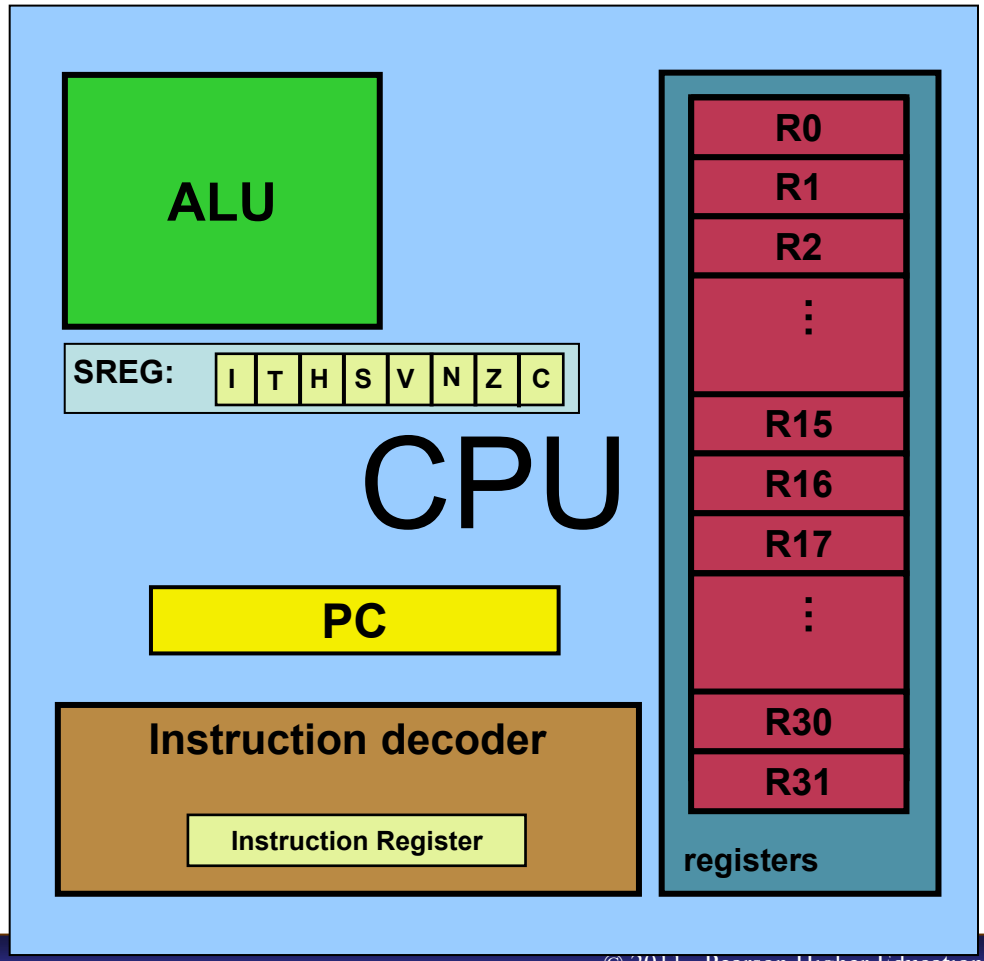
Topics

- AVR's CPU
  - Its architecture
  - Some simple programs
- Data Memory access
- Program memory
- RISC architecture



AVR's CPU

- AVR's CPU
  - ALU
  - 32 General Purpose registers (R0 to R31)
  - PC register
  - Instruction decoder

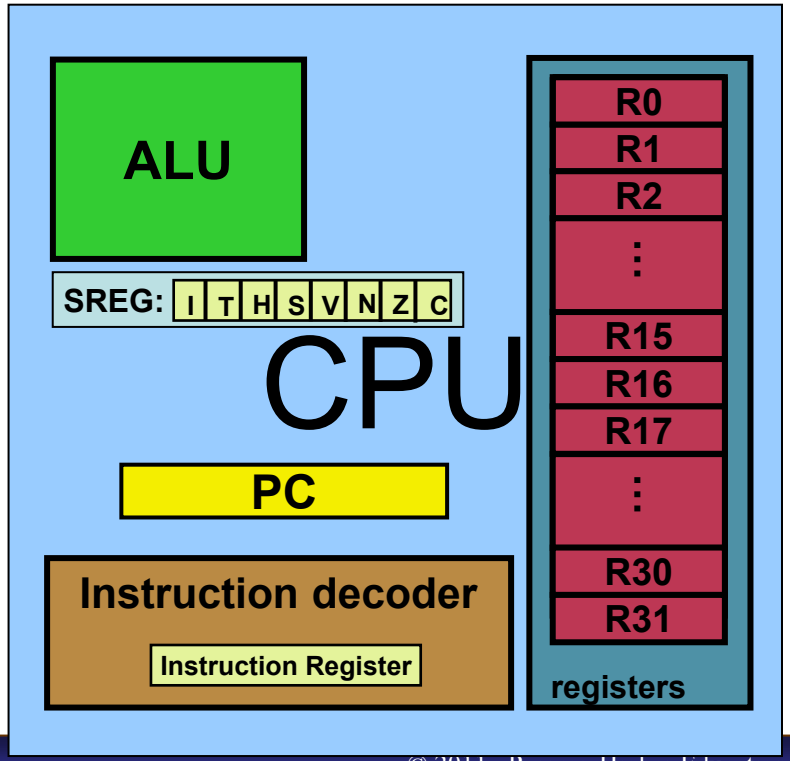


Some simple instructions
1. Loading values into the general purpose registers

LDI (**Loa**d **I**mmediate)

- LDI Rd, k
  - Its equivalent in high level languages:

    Rd = k
- Example:
  - LDI R16,53
    - R16 = 53
  - LDI R19,132
  - LDI R23,0x27
    - R23 = 0x27



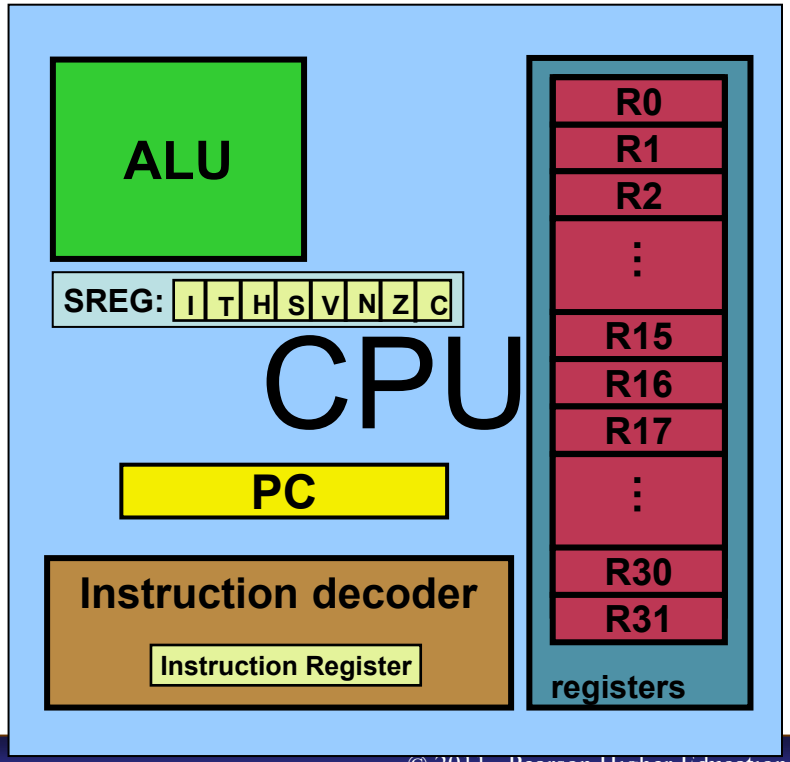
Some simple instructions

2. Arithmetic calculation

- There are some instructions for doing Arithmetic and logic operations; such as:

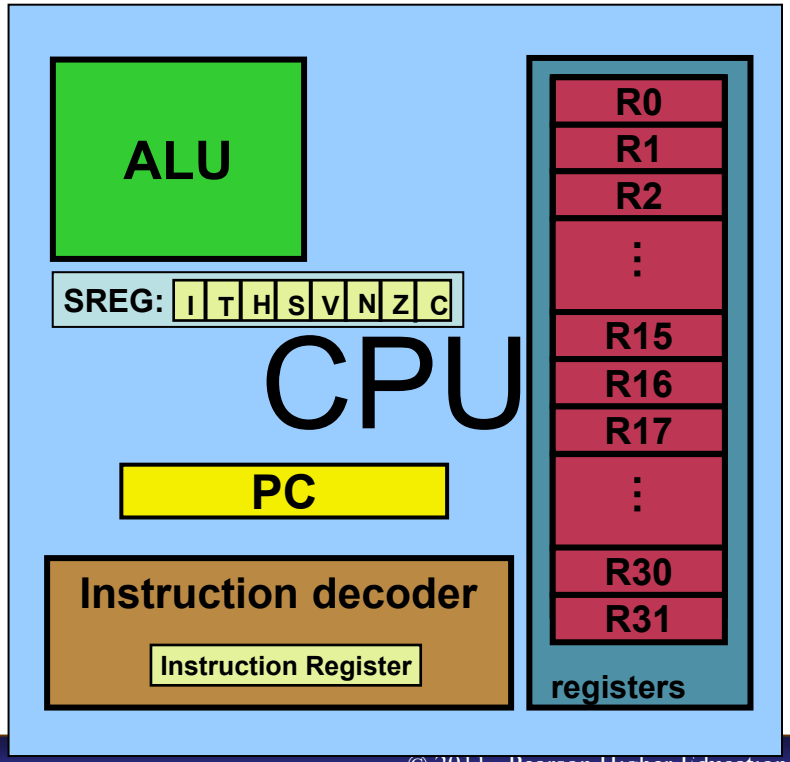
  ADD, SUB, MUL, AND, etc.

- ADD Rd,Rs
  - Rd = Rd + Rs
  - Example:
  - ADD R25, R9
    - R25 = R25 + R9
  - ADD R17,R30
    - R17 = R17 + R30



A simple program

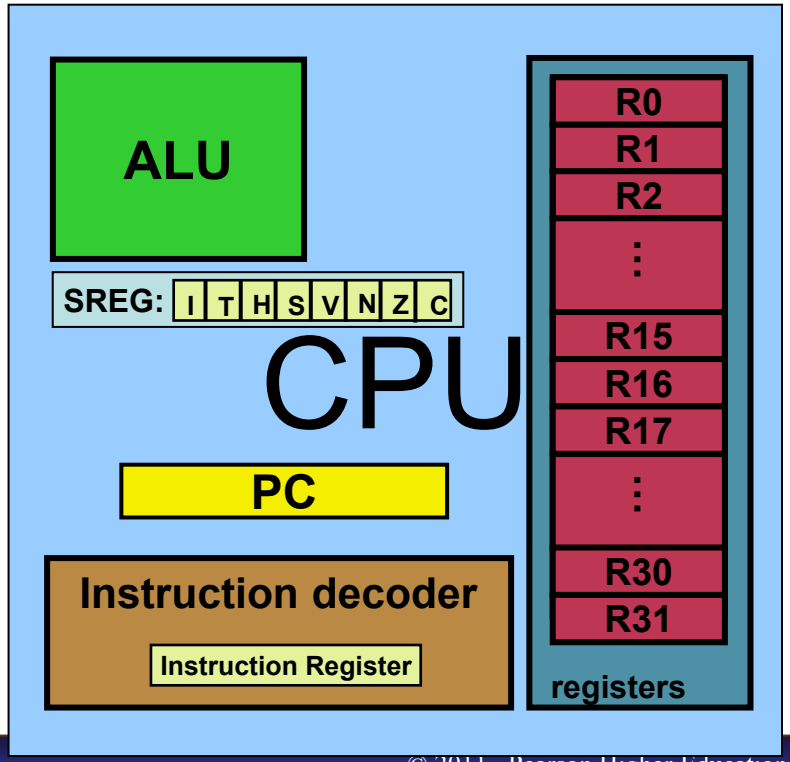
- Write a program that calculates 19 + 95



A simple program

- Write a program that calculates 19 + 95

```
LDI R16, 19     ;R16 = 19
LDI R20, 95     ;R20 = 95
ADD R16, R20    ;R16 = R16 + R20
```



A simple program

- Write a program that calculates 19 + 95 + 5

A simple program

- Write a program that calculates 19 + 95 + 5

```
LDI     R16, 19          ;R16 = 19
LDI     R20, 95          ;R20 = 95
LDI     R21, 5           ;R21 = 5
ADD     R16, R20         ;R16 = R16 + R20
ADD     R16, R21         ;R16 = R16 + R21
```

A simple program

- Write a program that calculates 19 + 95 + 5

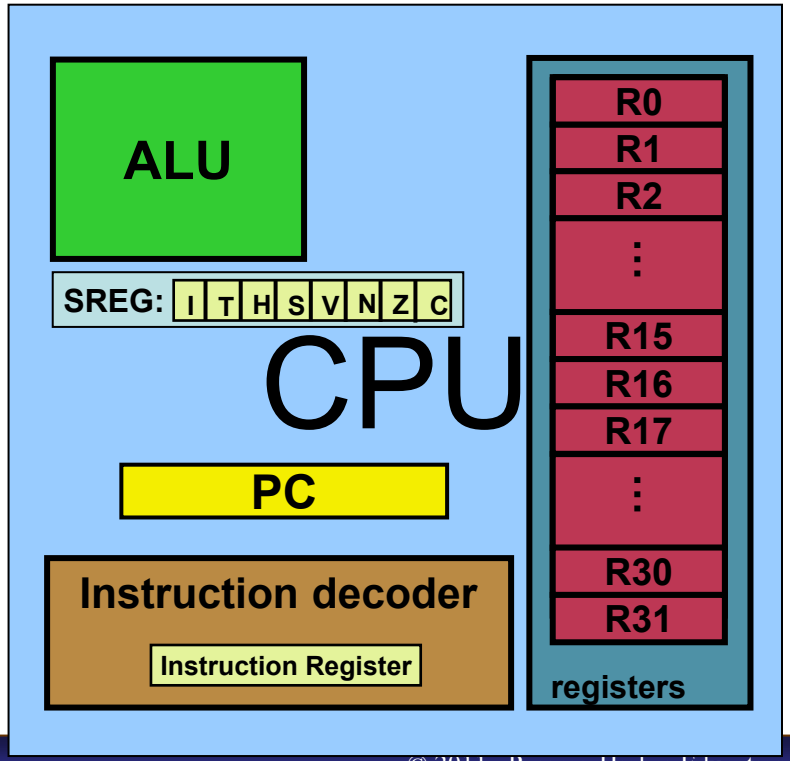
```
LDI     R16, 19         ;R16 = 19
LDI     R20, 95         ;R20 = 95
LDI     R21, 5          ;R21 = 5
ADD     R16, R20        ;R16 = R16 + R20
ADD     R16, R21        ;R16 = R16 + R21
```

```
LDI     R16, 19         ;R16 = 19
LDI     R20, 95         ;R20 = 95
ADD     R16, R20        ;R16 = R16 + R20
LDI     R20, 5          ;R20 = 5
ADD     R16, R20        ;R16 = R16 + R20
```

Some simple instructions

2. Arithmetic calculation

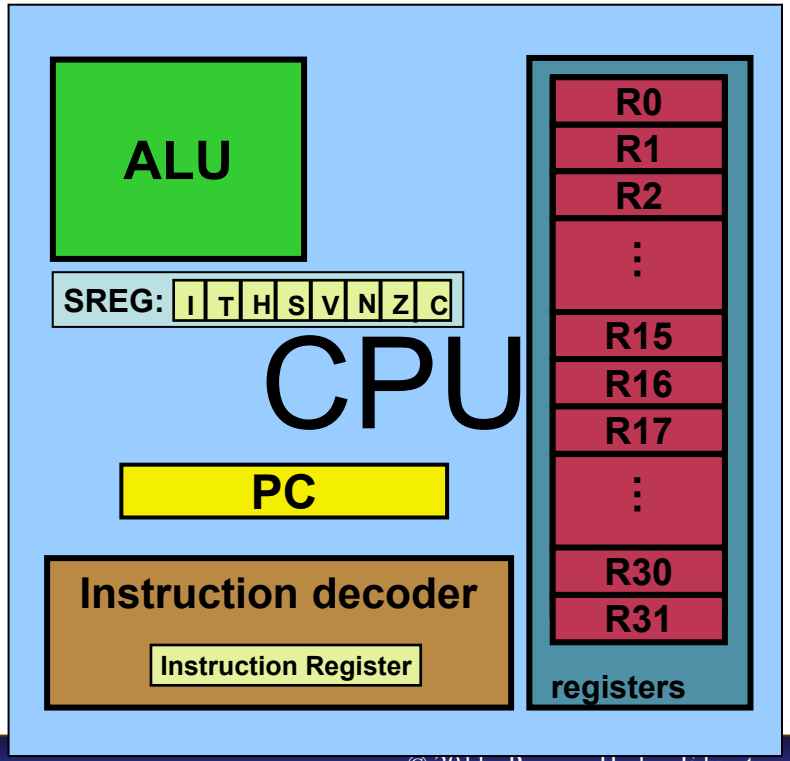
- SUB Rd,Rs
  - Rd = Rd - Rs
- Example:
  - SUB R25, R9
    - R25 = R25 - R9
  - SUB R17,R30
    - R17 = R17 - R30



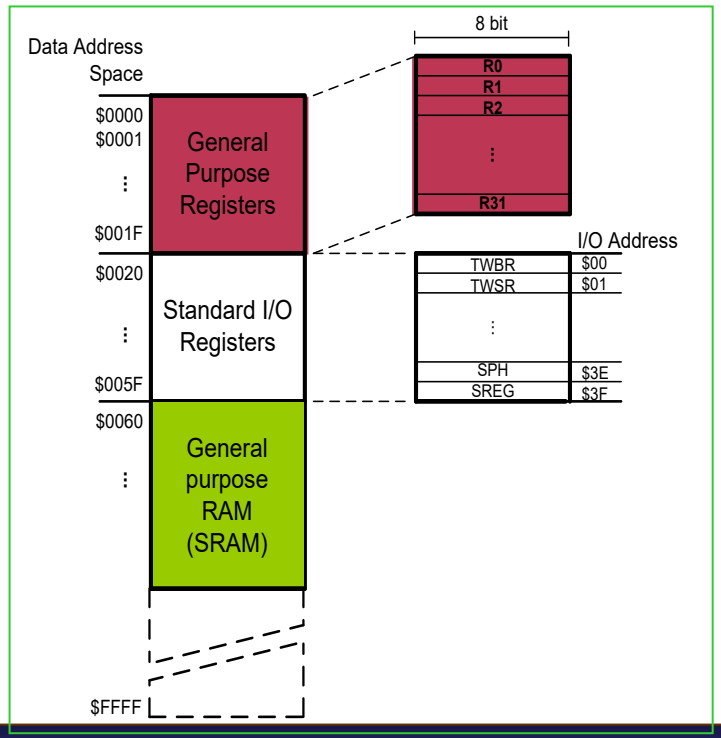
Some simple instructions

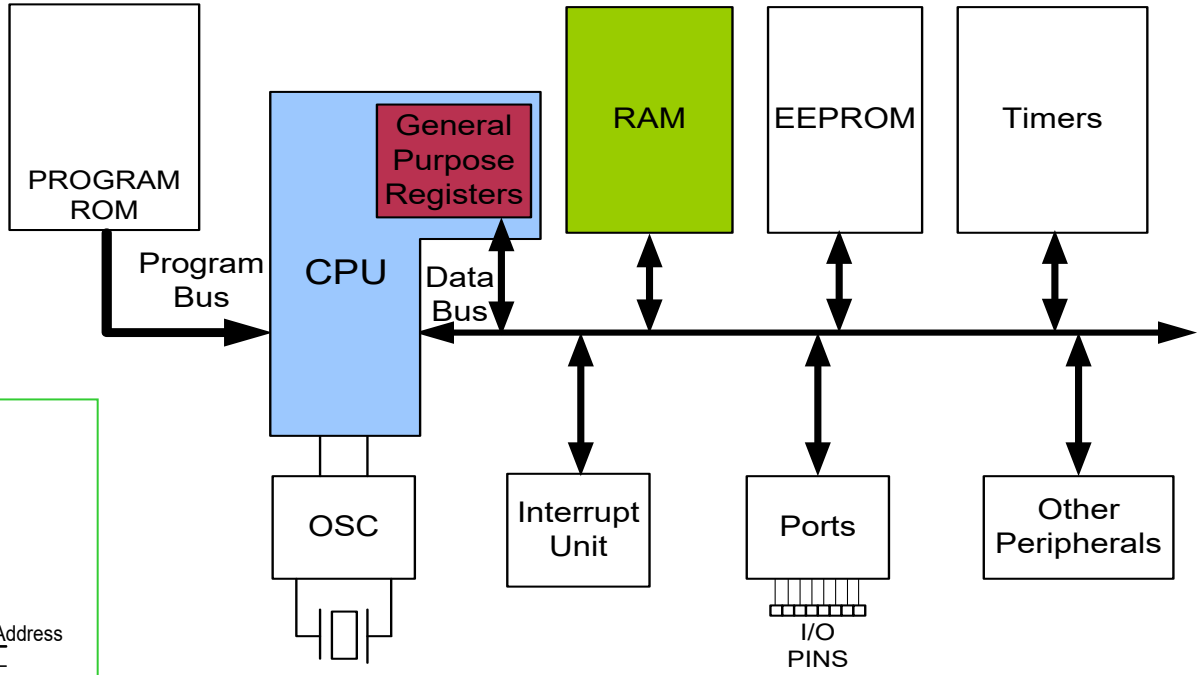
2. Arithmetic calculation

- INC Rd
  - Rd = Rd + 1
- Example:
  - INC R25
    - R25 = R25 + 1
- DEC Rd
  - Rd = Rd - 1
- Example:
  - DEC R23
    - R23 = R23 - 1

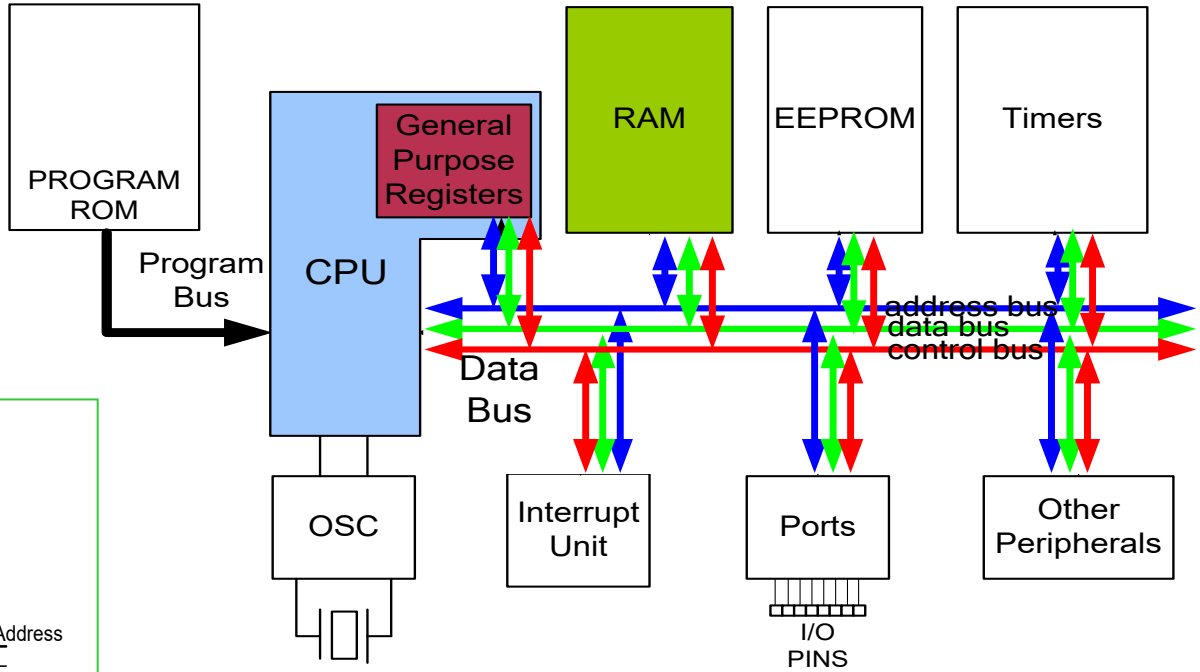


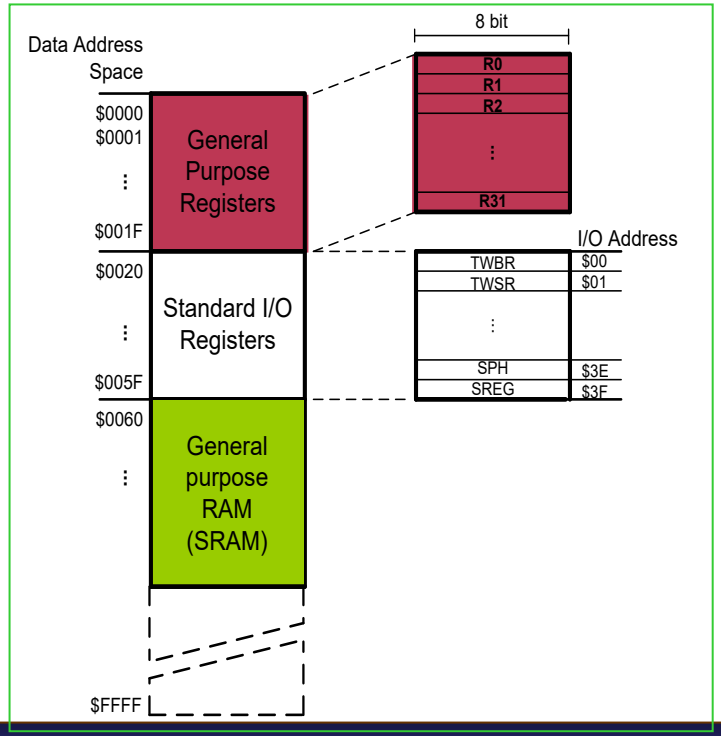
Data Address Space





Data Address Space





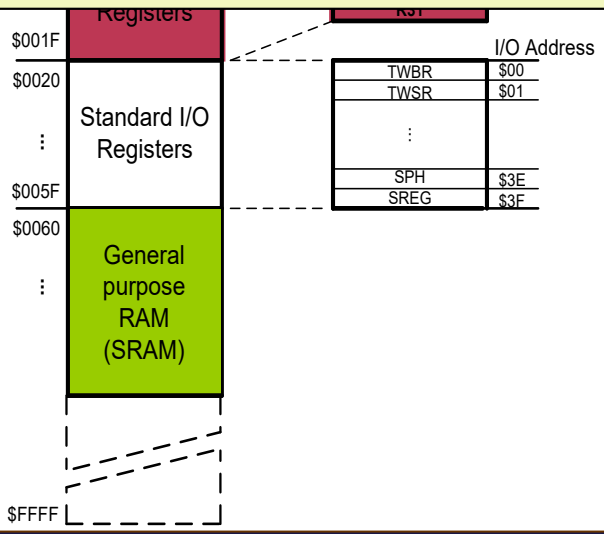
AVR Tutorials - Working With I/O Registers (rjhcoding.com)

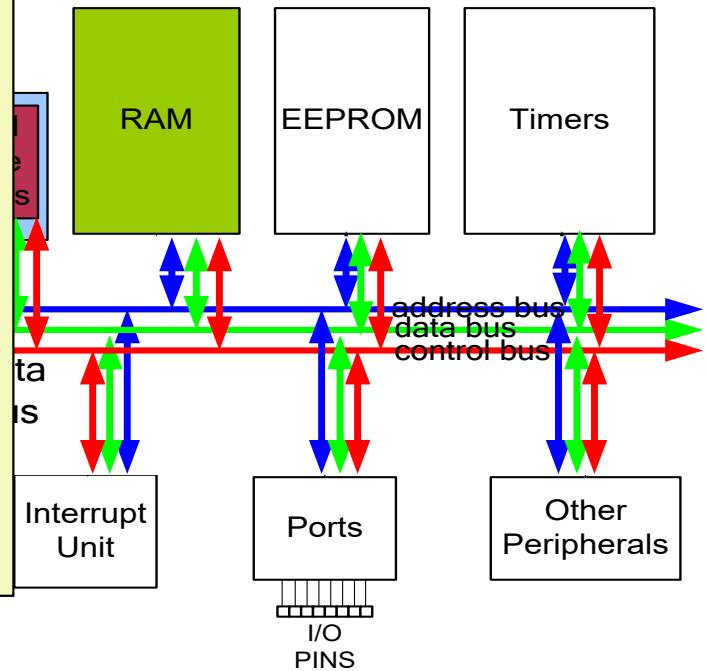
Space

Address I/O	Address Mem.	Name
$00	$20	TWBR
$01	$21	TWSR
$02	$22	TWAR
$03	$23	TWDR
$04	$24	ADCL
$05	$25	ADCH
$06	$26	ADCSRA
$07	$27	ADMUX
$08	$28	ACSR
$09	$29	UBRRL
$0A	$2A	UCSRB
$0B	$2B	UCSRA
$0C	$2C	UDR
$0D	$2D	SPCR
$0E	$2E	SPSR
$0F	$2F	SPDR
$10	$30	PIND
$11	$31	DDRD
$12	$32	PORTD
$13	$33	PINC
$14	$34	DDRC
$15	$35	PORTC

Address I/O	Address Mem.	Name
$16	$36	PINB
$17	$37	DDRB
$18	$38	PORTB
$19	$39	PINA
$1A	$3A	DDRA
$1B	$3B	PORTA
$1C	$3C	EECR
$1D	$3D	EEDR
$1E	$3E	EEARL
$1F	$3F	EEARH
$20	$40	UBRRC
		UBRRH
$21	$41	WDTCR
$22	$42	ASSR
$23	$43	OCR2
$24	$44	TCNT2
$25	$45	TCCR2
$26	$46	ICR1L
$27	$47	ICR1H
$28	$48	OCR1BL
$29	$49	OCR1BH
$2A	$4A	OCR1AL

Address I/O	Address Mem.	Name
$2B	$4B	OCR1AH
$2C	$4C	TCNT1L
$2D	$4D	TCNT1H
$2E	$4E	TCCR1B
$2F	$4F	TCCR1A
$30	$50	SFIOR
$31	$51	OCDR
		OSCCAL
$32	$52	TCNT0
$33	$53	TCCR0
$34	$54	MCUCSR
$35	$55	MCUCR
$36	$56	TWCR
$37	$57	SPMCR
$38	$58	TIFR
$39	$59	TIMSK
$3A	$5A	GIFR
$3B	$5B	GICR
$3C	$5C	OCR0
$3D	$5D	SPL
$3E	$5E	SPH
$3E	$5E	SREG





LDS (Load direct from data space)

LDS Rd, addr ;Rd = [addr]

Example:

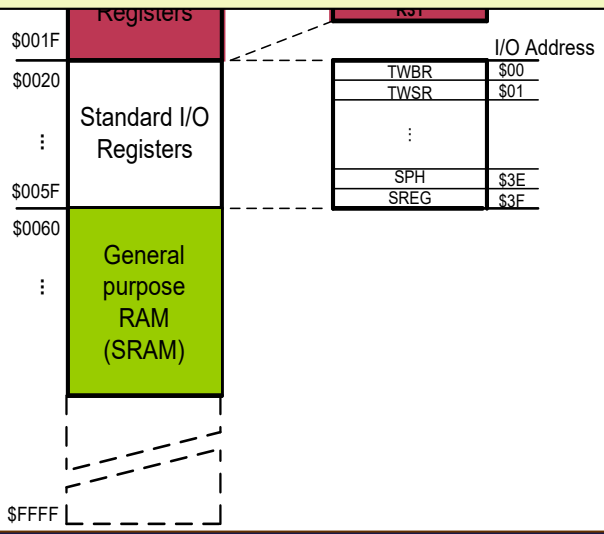
LDS R1, 0x60

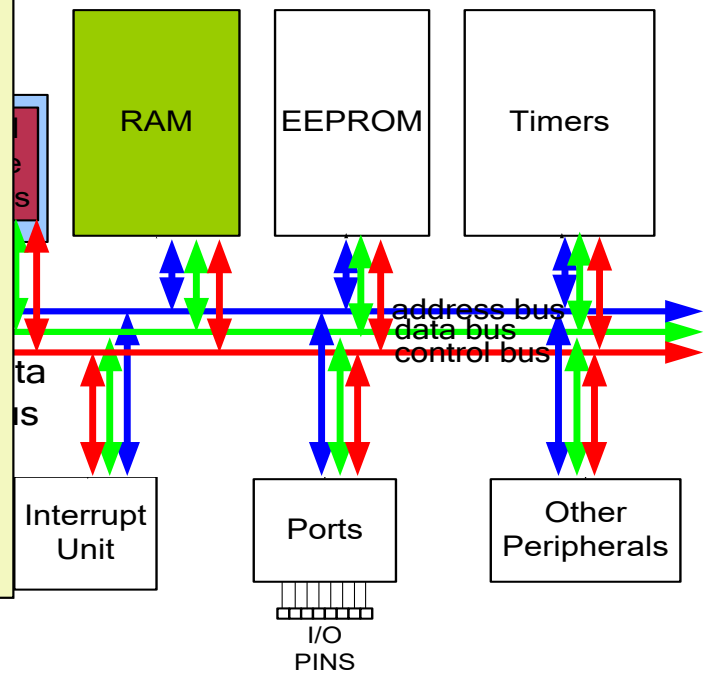
Space

Address I/O	Address Mem.	Name
$00	$20	TWBR
$01	$21	TWSR
$02	$22	TWAR
$03	$23	TWDR
$04	$24	ADCL
$05	$25	ADCH
$06	$26	ADCSRA
$07	$27	ADMUX
$08	$28	ACSR
$09	$29	UBRRL
$0A	$2A	UCSRB
$0B	$2B	UCSRA
$0C	$2C	UDR
$0D	$2D	SPCR
$0E	$2E	SPSR
$0F	$2F	SPDR
$10	$30	PIND
$11	$31	DDRD
$12	$32	PORTD
$13	$33	PINC
$14	$34	DDRC
$15	$35	PORTC

Address I/O	Address Mem.	Name
$16	$36	PINB
$17	$37	DDRB
$18	$38	PORTB
$19	$39	PINA
$1A	$3A	DDRA
$1B	$3B	PORTA
$1C	$3C	EECR
$1D	$3D	EEDR
$1E	$3E	EEARL
$1F	$3F	EEARH
$20	$40	UBRRC
		UBRRH
$21	$41	WDTCR
$22	$42	ASSR
$23	$43	OCR2
$24	$44	TCNT2
$25	$45	TCCR2
$26	$46	ICR1L
$27	$47	ICR1H
$28	$48	OCR1BL
$29	$49	OCR1BH
$2A	$4A	OCR1AL

Address I/O	Address Mem.	Name
$2B	$4B	OCR1AH
$2C	$4C	TCNT1L
$2D	$4D	TCNT1H
$2E	$4E	TCCR1B
$2F	$4F	TCCR1A
$30	$50	SFIOR
$31	$51	OCDR
		OSCCAL
$32	$52	TCNT0
$33	$53	TCCR0
$34	$54	MCUCSR
$35	$55	MCUCR
$36	$56	TWCR
$37	$57	SPMCR
$38	$58	TIFR
$39	$59	TIMSK
$3A	$5A	GIFR
$3B	$5B	GICR
$3C	$5C	OCR0
$3D	$5D	SPL
$3E	$5E	SPH
$3E	$5E	SREG





STS (Store direct to data space)

STS addr,Rd ;[addr]=Rd

Example:

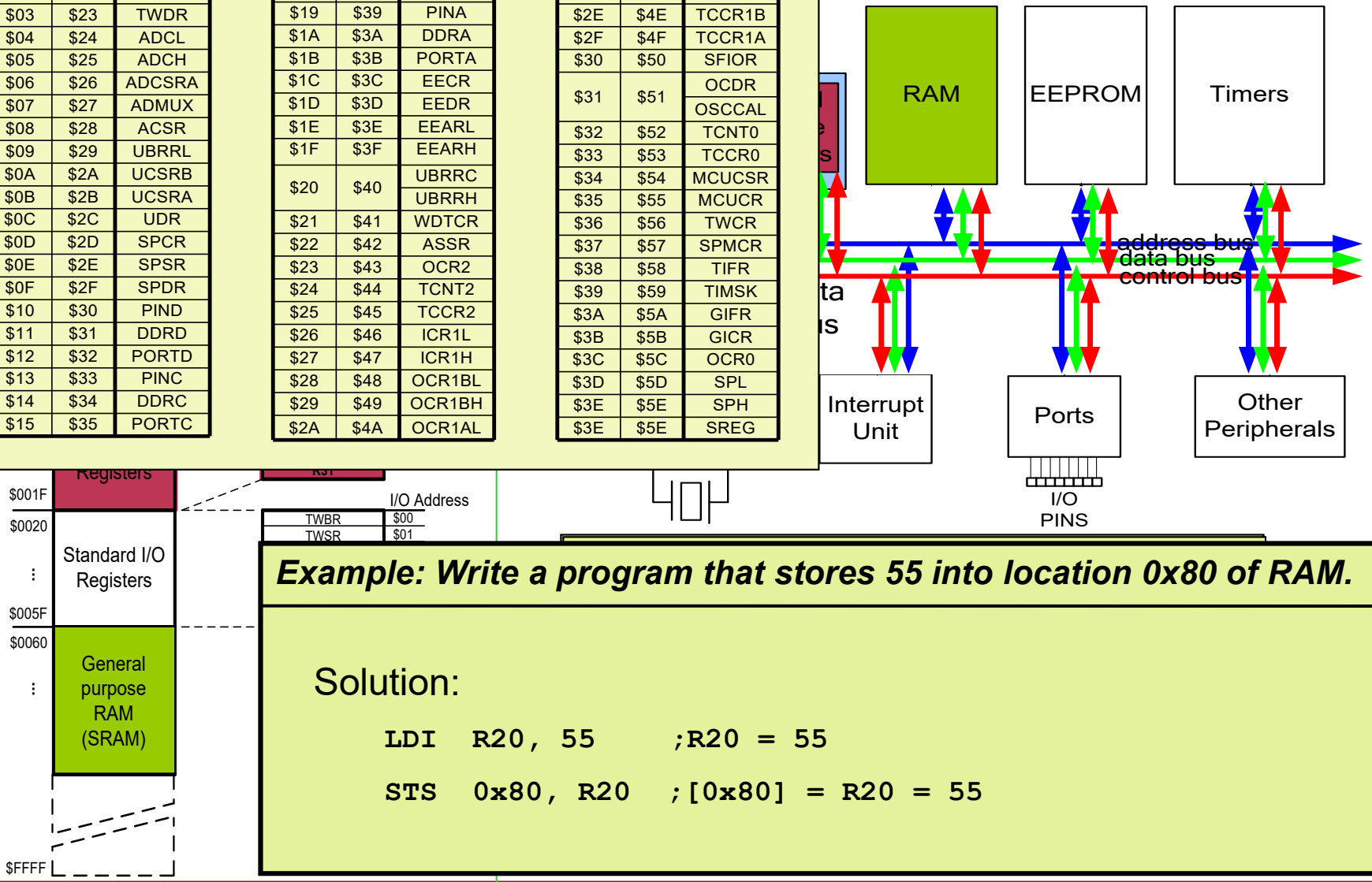
STS 0x60,R15 ; [0x60] = R15

Space

Address I/O	Address Mem.	Name
$00	$20	TWBR
$01	$21	TWSR
$02	$22	TWAR
$03	$23	TWDR
$04	$24	ADCL
$05	$25	ADCH
$06	$26	ADCSRA
$07	$27	ADMUX
$08	$28	ACSR
$09	$29	UBRRL
$0A	$2A	UCSRB
$0B	$2B	UCSRA
$0C	$2C	UDR
$0D	$2D	SPCR
$0E	$2E	SPSR
$0F	$2F	SPDR
$10	$30	PIND
$11	$31	DDRD
$12	$32	PORTD
$13	$33	PINC
$14	$34	DDRC
$15	$35	PORTC

Address I/O	Address Mem.	Name
$16	$36	PINB
$17	$37	DDRB
$18	$38	PORTB
$19	$39	PINA
$1A	$3A	DDRA
$1B	$3B	PORTA
$1C	$3C	EECR
$1D	$3D	EEDR
$1E	$3E	EEARL
$1F	$3F	EEARH
$20	$40	UBRRC
		UBRRH
$21	$41	WDTCR
$22	$42	ASSR
$23	$43	OCR2
$24	$44	TCNT2
$25	$45	TCCR2
$26	$46	ICR1L
$27	$47	ICR1H
$28	$48	OCR1BL
$29	$49	OCR1BH
$2A	$4A	OCR1AL

Address I/O	Address Mem.	Name
$2B	$4B	OCR1AH
$2C	$4C	TCNT1L
$2D	$4D	TCNT1H
$2E	$4E	TCCR1B
$2F	$4F	TCCR1A
$30	$50	SFIOR
$31	$51	OCDR
		OSCCAL
$32	$52	TCNT0
$33	$53	TCCR0
$34	$54	MCUCSR
$35	$55	MCUCR
$36	$56	TWCR
$37	$57	SPMCR
$38	$58	TIFR
$39	$59	TIMSK
$3A	$5A	GIFR
$3B	$5B	GICR
$3C	$5C	OCR0
$3D	$5D	SPL
$3E	$5E	SPH
$3E	$5E	SREG



Example: Write a program that stores 55 into location 0x80 of RAM.

Solution:

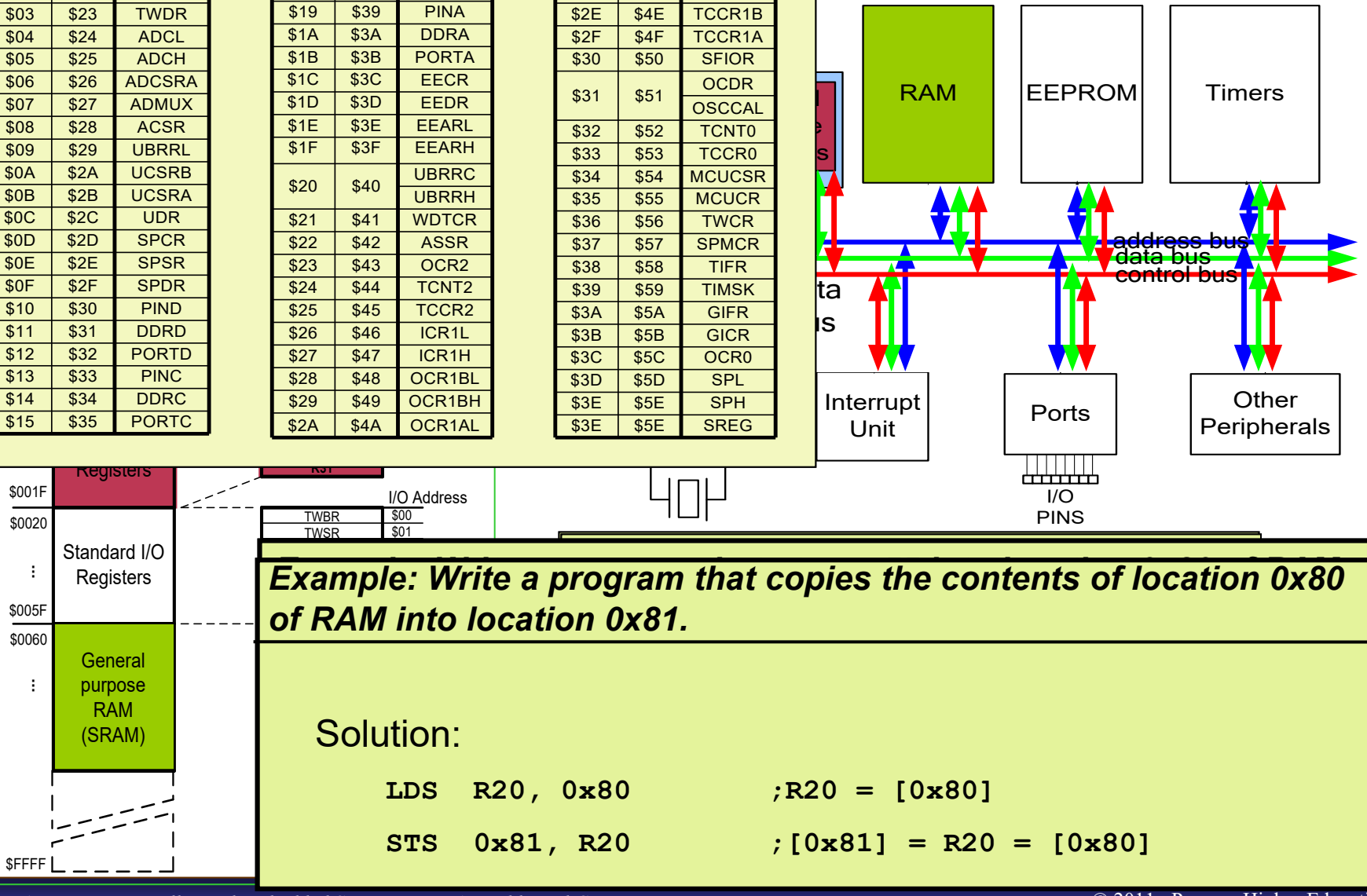
```
LDI   R20, 55     ;R20 = 55
STS   0x80, R20   ;[0x80] = R20 = 55
```

Space

Address I/O	Address Mem.	Name
$00	$20	TWBR
$01	$21	TWSR
$02	$22	TWAR
$03	$23	TWDR
$04	$24	ADCL
$05	$25	ADCH
$06	$26	ADCSRA
$07	$27	ADMUX
$08	$28	ACSR
$09	$29	UBRRL
$0A	$2A	UCSRB
$0B	$2B	UCSRA
$0C	$2C	UDR
$0D	$2D	SPCR
$0E	$2E	SPSR
$0F	$2F	SPDR
$10	$30	PIND
$11	$31	DDRD
$12	$32	PORTD
$13	$33	PINC
$14	$34	DDRC
$15	$35	PORTC

Address I/O	Address Mem.	Name
$16	$36	PINB
$17	$37	DDRB
$18	$38	PORTB
$19	$39	PINA
$1A	$3A	DDRA
$1B	$3B	PORTA
$1C	$3C	EECR
$1D	$3D	EEDR
$1E	$3E	EEARL
$1F	$3F	EEARH
$20	$40	UBRRC
		UBRRH
$21	$41	WDTCR
$22	$42	ASSR
$23	$43	OCR2
$24	$44	TCNT2
$25	$45	TCCR2
$26	$46	ICR1L
$27	$47	ICR1H
$28	$48	OCR1BL
$29	$49	OCR1BH
$2A	$4A	OCR1AL

Address I/O	Address Mem.	Name
$2B	$4B	OCR1AH
$2C	$4C	TCNT1L
$2D	$4D	TCNT1H
$2E	$4E	TCCR1B
$2F	$4F	TCCR1A
$30	$50	SFIOR
$31	$51	OCDR
		OSCCAL
$32	$52	TCNT0
$33	$53	TCCR0
$34	$54	MCUCSR
$35	$55	MCUCR
$36	$56	TWCR
$37	$57	SPMCR
$38	$58	TIFR
$39	$59	TIMSK
$3A	$5A	GIFR
$3B	$5B	GICR
$3C	$5C	OCR0
$3D	$5D	SPL
$3E	$5E	SPH
$3E	$5E	SREG



Example: Write a program that copies the contents of location 0x80 of RAM into location 0x81.

Solution:

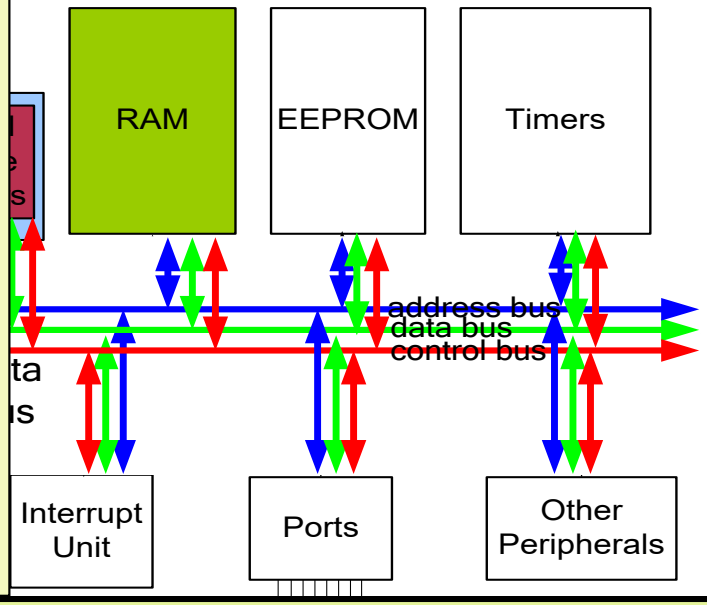
```
LDS   R20, 0x80          ;R20 = [0x80]
STS   0x81, R20          ;[0x81] = R20 = [0x80]
```

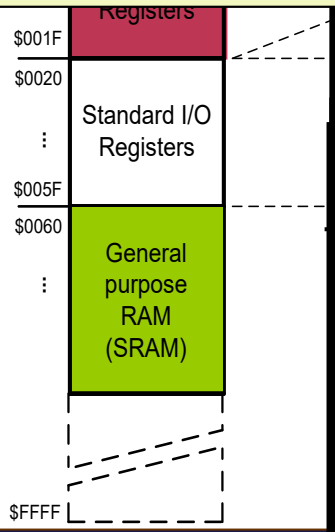
Space

Address I/O	Address Mem.	Name
$00	$20	TWBR
$01	$21	TWSR
$02	$22	TWAR
$03	$23	TWDR
$04	$24	ADCL
$05	$25	ADCH
$06	$26	ADCSRA
$07	$27	ADMUX
$08	$28	ACSR
$09	$29	UBRRL
$0A	$2A	UCSRB
$0B	$2B	UCSRA
$0C	$2C	UDR
$0D	$2D	SPCR
$0E	$2E	SPSR
$0F	$2F	SPDR
$10	$30	PIND
$11	$31	DDRD
$12	$32	PORTD
$13	$33	PINC
$14	$34	DDRC
$15	$35	PORTC

Address I/O	Address Mem.	Name
$16	$36	PINB
$17	$37	DDRB
$18	$38	PORTB
$19	$39	PINA
$1A	$3A	DDRA
$1B	$3B	PORTA
$1C	$3C	EECR
$1D	$3D	EEDR
$1E	$3E	EEARL
$1F	$3F	EEARH
$20	$40	UBRRC
		UBRRH
$21	$41	WDTCR
$22	$42	ASSR
$23	$43	OCR2
$24	$44	TCNT2
$25	$45	TCCR2
$26	$46	ICR1L
$27	$47	ICR1H
$28	$48	OCR1BL
$29	$49	OCR1BH
$2A	$4A	OCR1AL

Address I/O	Address Mem.	Name
$2B	$4B	OCR1AH
$2C	$4C	TCNT1L
$2D	$4D	TCNT1H
$2E	$4E	TCCR1B
$2F	$4F	TCCR1A
$30	$50	SFIOR
$31	$51	OCDR
		OSCCAL
$32	$52	TCNT0
$33	$53	TCCR0
$34	$54	MCUCSR
$35	$55	MCUCR
$36	$56	TWCR
$37	$57	SPMCR
$38	$58	TIFR
$39	$59	TIMSK
$3A	$5A	GIFR
$3B	$5B	GICR
$3C	$5C	OCR0
$3D	$5D	SPL
$3E	$5E	SPH
$3E	$5E	SREG





Example: Add contents of location 0x90 to contents of location 0x95 and store the result in location 0x313.

Solution:

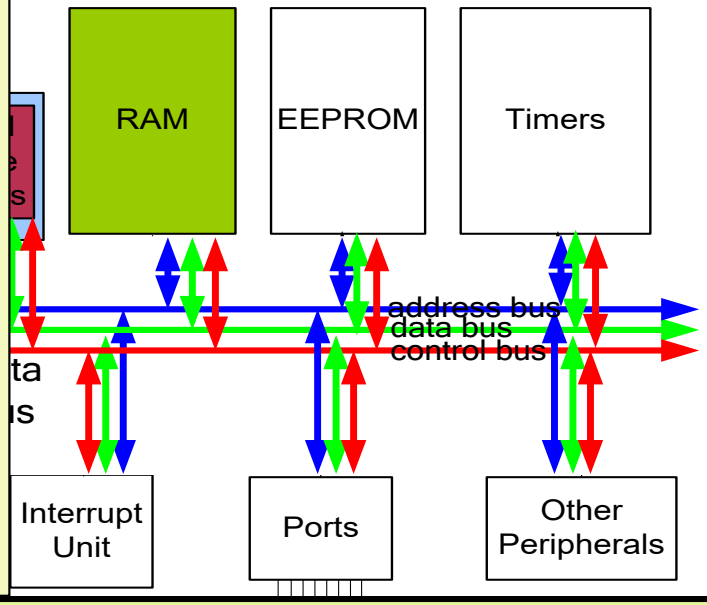
```
LDS   R20, 0x90        ;R20 = [0x90]
LDS   R21, 0x95        ;R21 = [0x95]
ADD   R20, R21         ;R20 = R20 + R21
STS   0x313, R20       ;[0x313] = R20
```

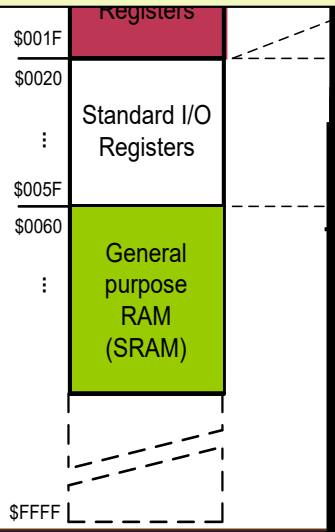
Space

Address I/O	Address Mem.	Name
$00	$20	TWBR
$01	$21	TWSR
$02	$22	TWAR
$03	$23	TWDR
$04	$24	ADCL
$05	$25	ADCH
$06	$26	ADCSRA
$07	$27	ADMUX
$08	$28	ACSR
$09	$29	UBRRL
$0A	$2A	UCSRB
$0B	$2B	UCSRA
$0C	$2C	UDR
$0D	$2D	SPCR
$0E	$2E	SPSR
$0F	$2F	SPDR
$10	$30	PIND
$11	$31	DDRD
$12	$32	PORTD
$13	$33	PINC
$14	$34	DDRC
$15	$35	PORTC

Address I/O	Address Mem.	Name
$16	$36	PINB
$17	$37	DDRB
$18	$38	PORTB
$19	$39	PINA
$1A	$3A	DDRA
$1B	$3B	PORTA
$1C	$3C	EECR
$1D	$3D	EEDR
$1E	$3E	EEARL
$1F	$3F	EEARH
$20	$40	UBRRC
		UBRRH
$21	$41	WDTCR
$22	$42	ASSR
$23	$43	OCR2
$24	$44	TCNT2
$25	$45	TCCR2
$26	$46	ICR1L
$27	$47	ICR1H
$28	$48	OCR1BL
$29	$49	OCR1BH
$2A	$4A	OCR1AL

Address I/O	Address Mem.	Name
$2B	$4B	OCR1AH
$2C	$4C	TCNT1L
$2D	$4D	TCNT1H
$2E	$4E	TCCR1B
$2F	$4F	TCCR1A
$30	$50	SFIOR
$31	$51	OCDR
		OSCCAL
$32	$52	TCNT0
$33	$53	TCCR0
$34	$54	MCUCSR
$35	$55	MCUCR
$36	$56	TWCR
$37	$57	SPMCR
$38	$58	TIFR
$39	$59	TIMSK
$3A	$5A	GIFR
$3B	$5B	GICR
$3C	$5C	OCR0
$3D	$5D	SPL
$3E	$5E	SPH
$3E	$5E	SREG





Example: Add contents of location 0x90 to contents of location 0x95

Example: What does the following instruction do?

```
LDS   R20,2
```

Answer:

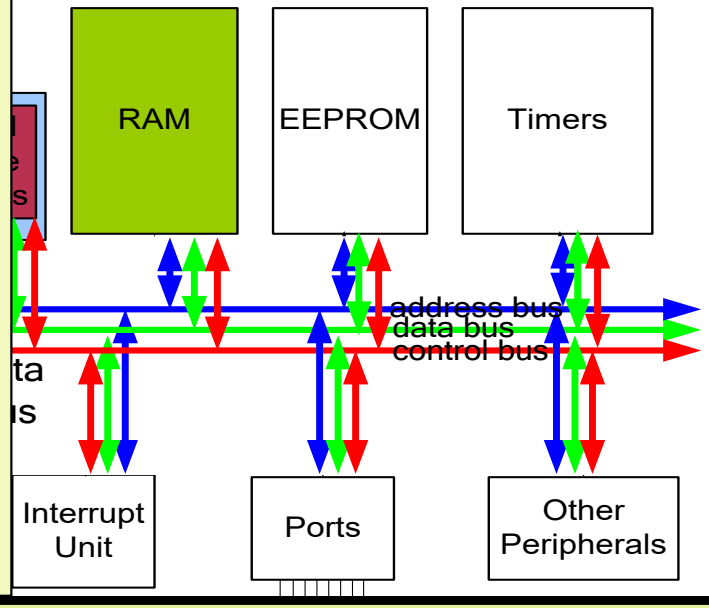
It copies the contents of R2 into R20; as 2 is the address of R2.

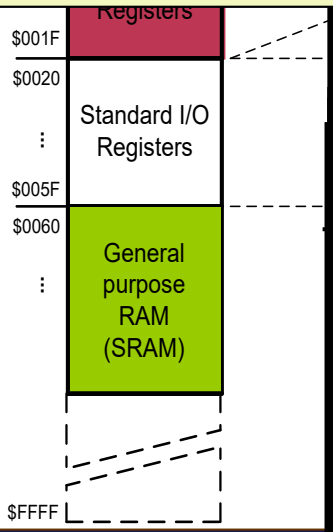
Space

Address I/O	Address Mem.	Name
$00	$20	TWBR
$01	$21	TWSR
$02	$22	TWAR
$03	$23	TWDR
$04	$24	ADCL
$05	$25	ADCH
$06	$26	ADCSRA
$07	$27	ADMUX
$08	$28	ACSR
$09	$29	UBRRL
$0A	$2A	UCSRB
$0B	$2B	UCSRA
$0C	$2C	UDR
$0D	$2D	SPCR
$0E	$2E	SPSR
$0F	$2F	SPDR
$10	$30	PIND
$11	$31	DDRD
$12	$32	PORTD
$13	$33	PINC
$14	$34	DDRC
$15	$35	PORTC

Address I/O	Address Mem.	Name
$16	$36	PINB
$17	$37	DDRB
$18	$38	PORTB
$19	$39	PINA
$1A	$3A	DDRA
$1B	$3B	PORTA
$1C	$3C	EECR
$1D	$3D	EEDR
$1E	$3E	EEARL
$1F	$3F	EEARH
$20	$40	UBRRC
		UBRRH
$21	$41	WDTCR
$22	$42	ASSR
$23	$43	OCR2
$24	$44	TCNT2
$25	$45	TCCR2
$26	$46	ICR1L
$27	$47	ICR1H
$28	$48	OCR1BL
$29	$49	OCR1BH
$2A	$4A	OCR1AL

Address I/O	Address Mem.	Name
$2B	$4B	OCR1AH
$2C	$4C	TCNT1L
$2D	$4D	TCNT1H
$2E	$4E	TCCR1B
$2F	$4F	TCCR1A
$30	$50	SFIOR
$31	$51	OCDR
		OSCCAL
$32	$52	TCNT0
$33	$53	TCCR0
$34	$54	MCUCSR
$35	$55	MCUCR
$36	$56	TWCR
$37	$57	SPMCR
$38	$58	TIFR
$39	$59	TIMSK
$3A	$5A	GIFR
$3B	$5B	GICR
$3C	$5C	OCR0
$3D	$5D	SPL
$3E	$5E	SPH
$3E	$5E	SREG





Example: Add conten

Example: What does

```
LDS   R20,2
```

Answer:

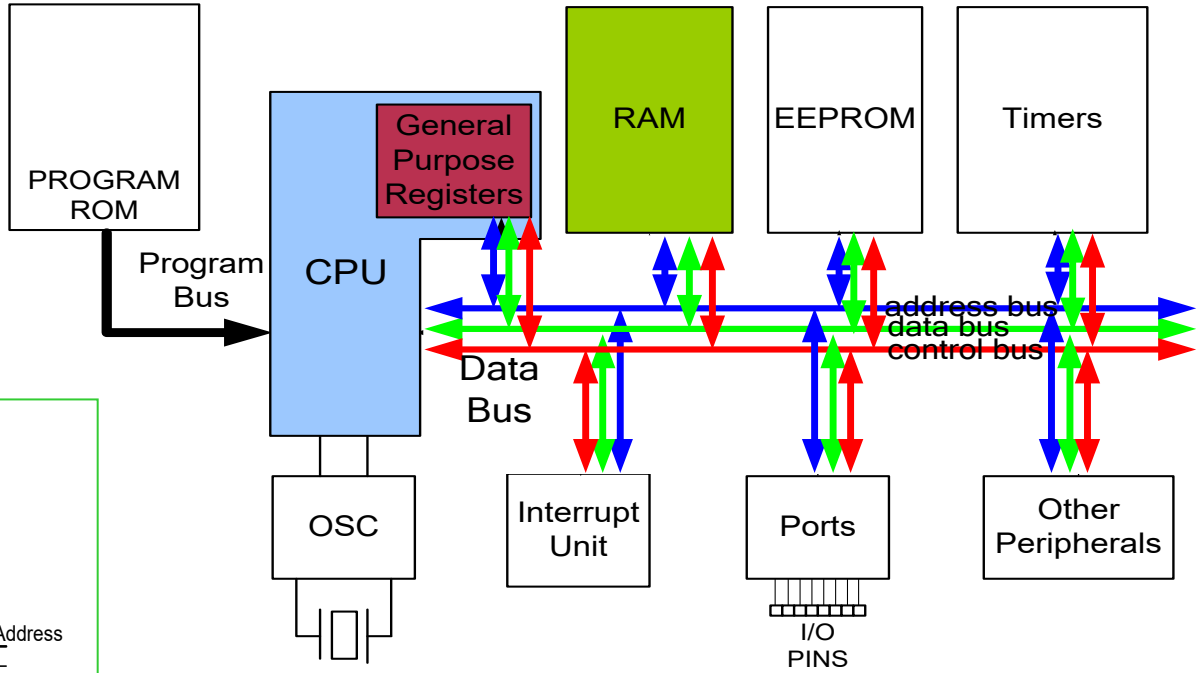
It copies the conte

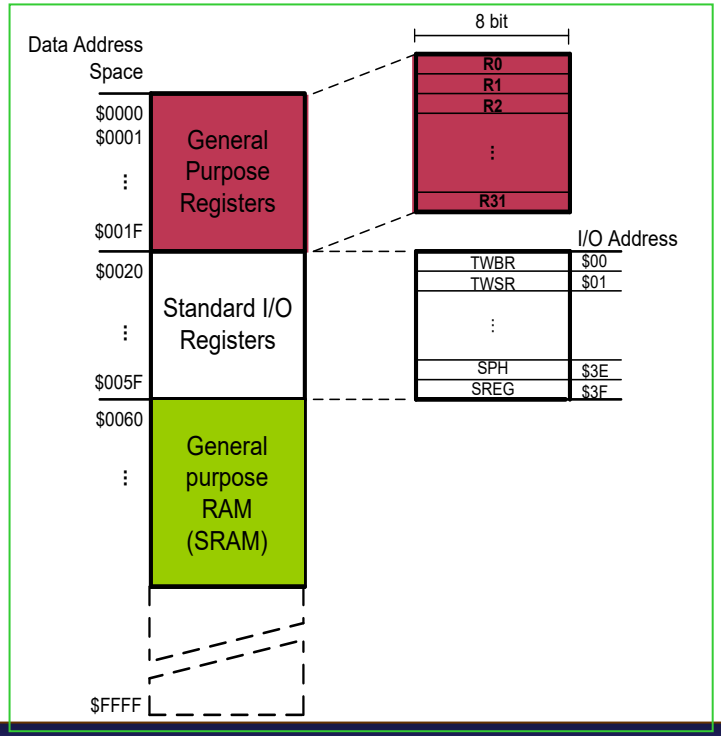
Example: Store 0x53 into the SPH register. The address of SPH is 0x5E

Solution:

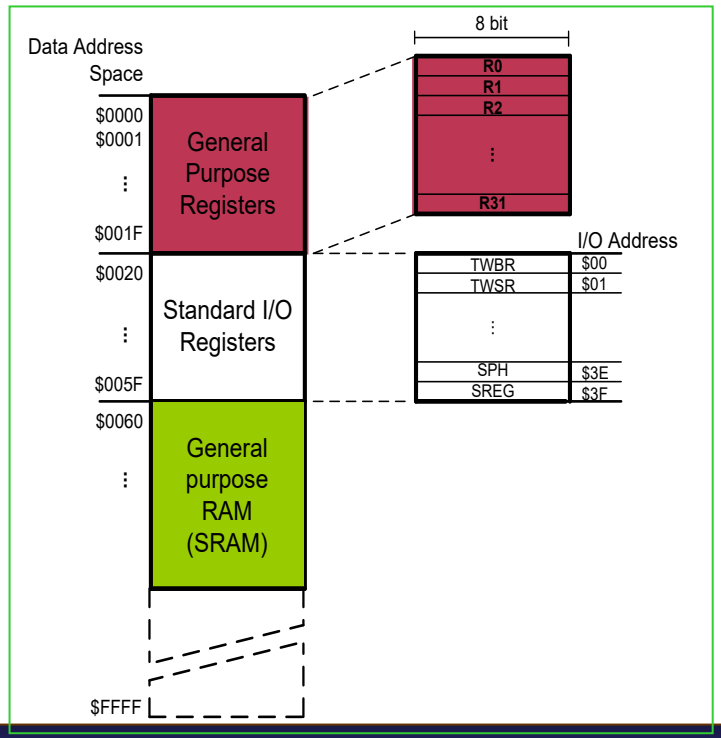
```
LDI   R20, 0x53          ;R20 = 0x53
STS   0x5E, R20          ;SPH = R20
```

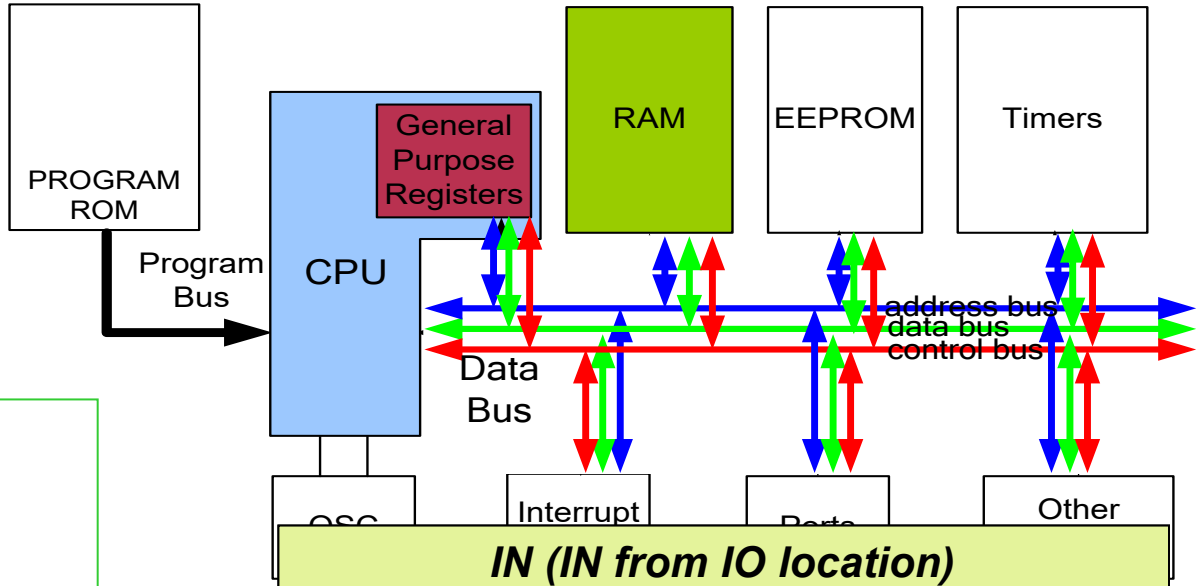
Data Address Space





Data Address Space





IN (IN from IO location)

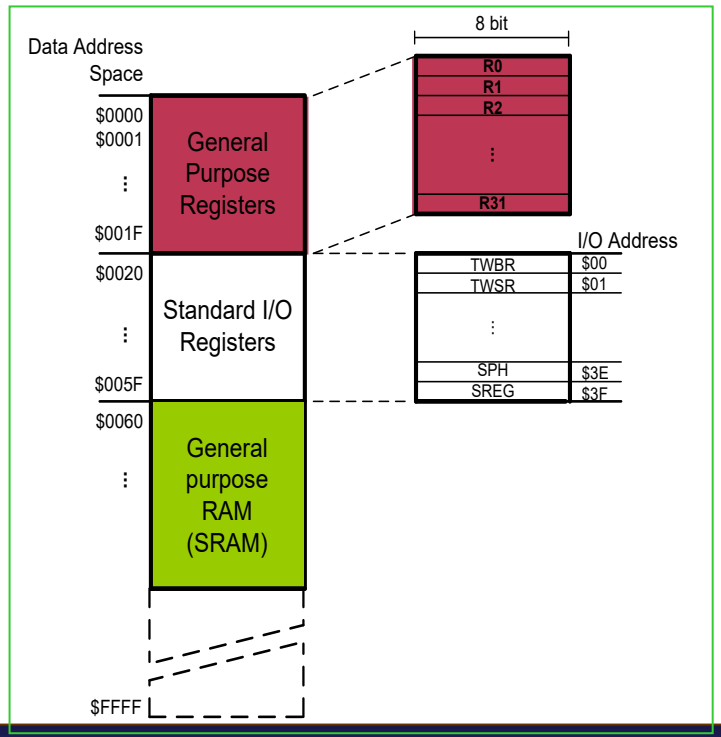
IN Rd,IOaddress ;Rd = [addr]

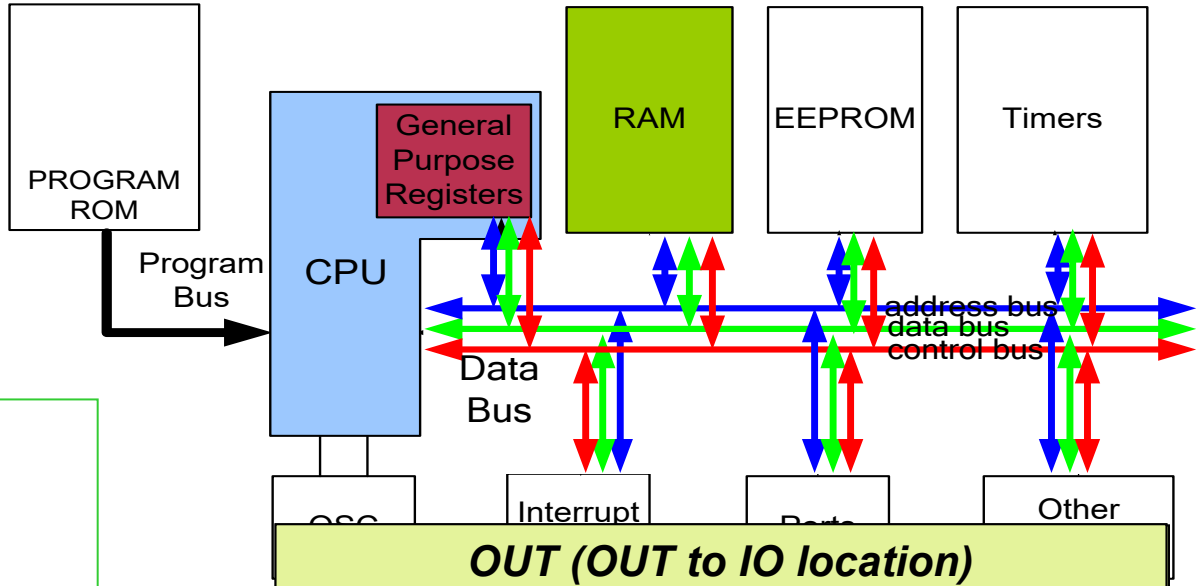
Example:

IN R1, 0x3F ;R1 = SREG

IN R17,0x3E ;R17 = SPH

Data Address Space





OUT (OUT to IO location)

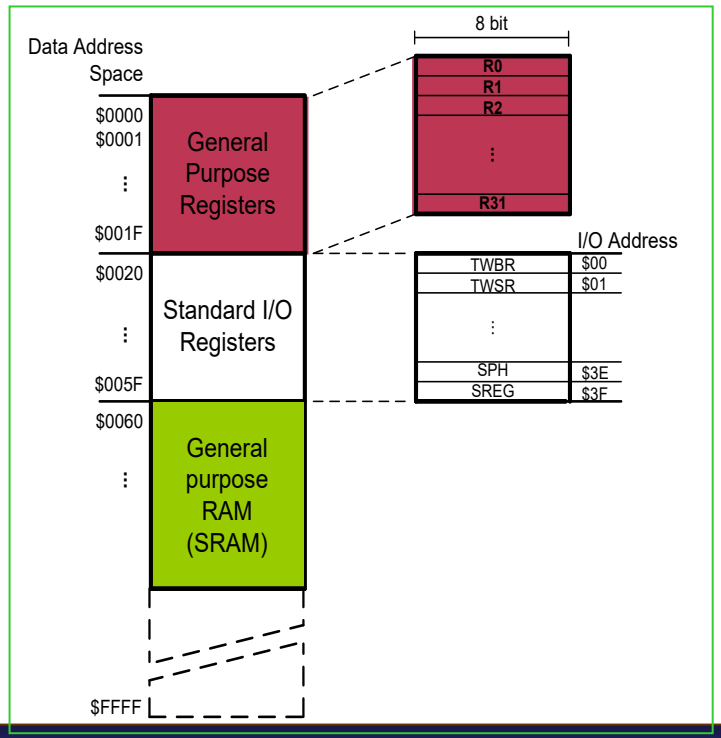
OUT IOAddr,Rd ;[addr]=Rd

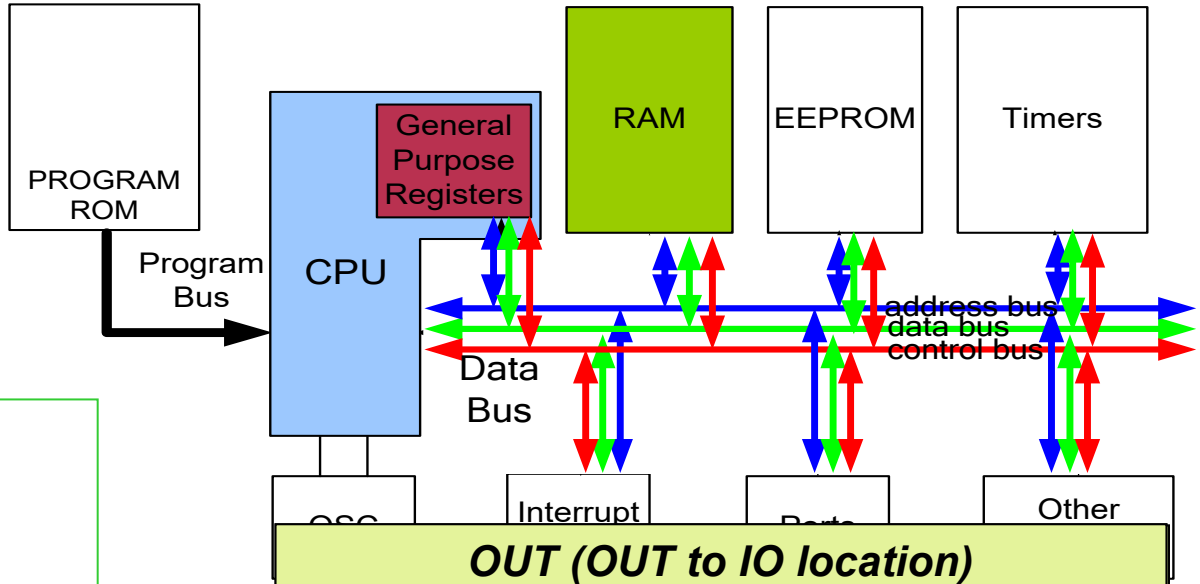
Example:

OUT 0x3F,R12 ;SREG = R12

OUT 0x3E,R15 ;SPH = R15

Data Address Space





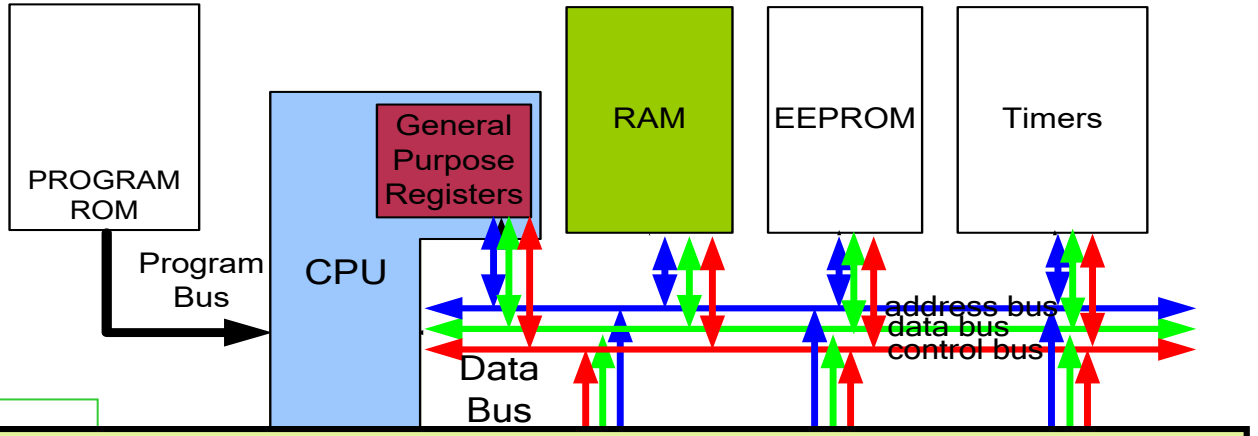
OUT (OUT to IO location)

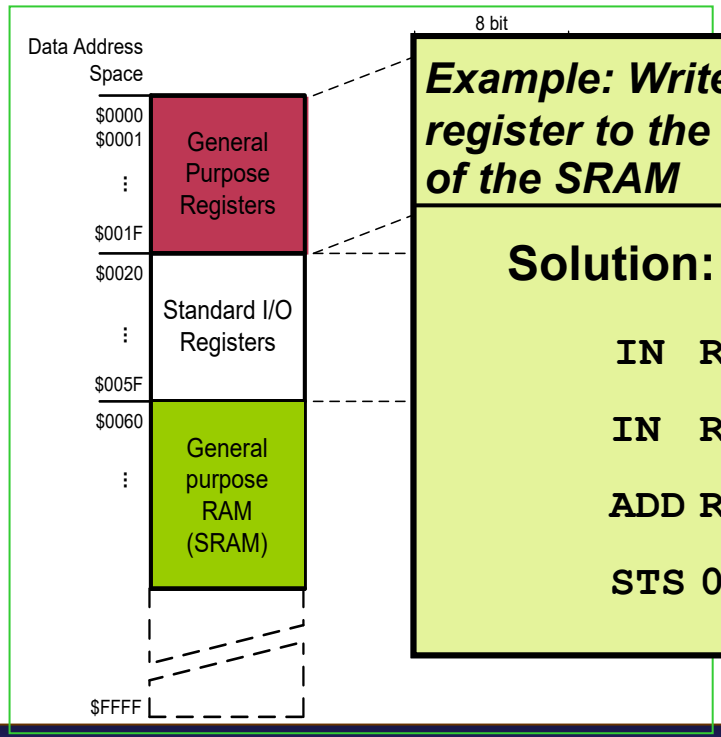
Using Names of IO registers

Example:

```
OUT   SPH,R12      ;OUT   0x3E,R12
IN    R15,SREG     ;IN    R15,0x3F
```

Data Address Space



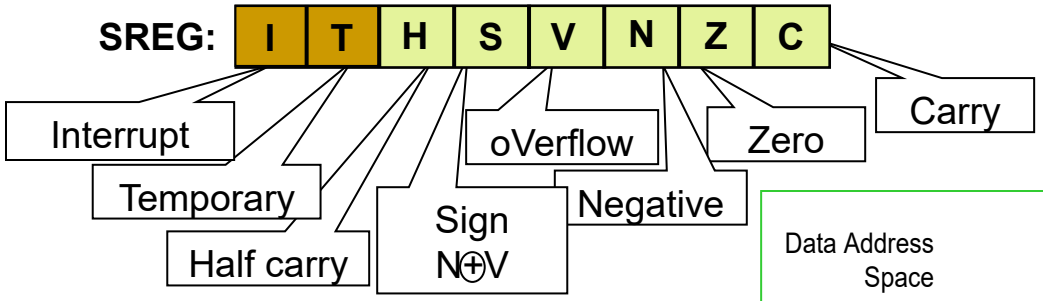


Example: Write a program that adds the contents of the PINC IO register to the contents of PIND and stores the result in location 0x90 of the SRAM

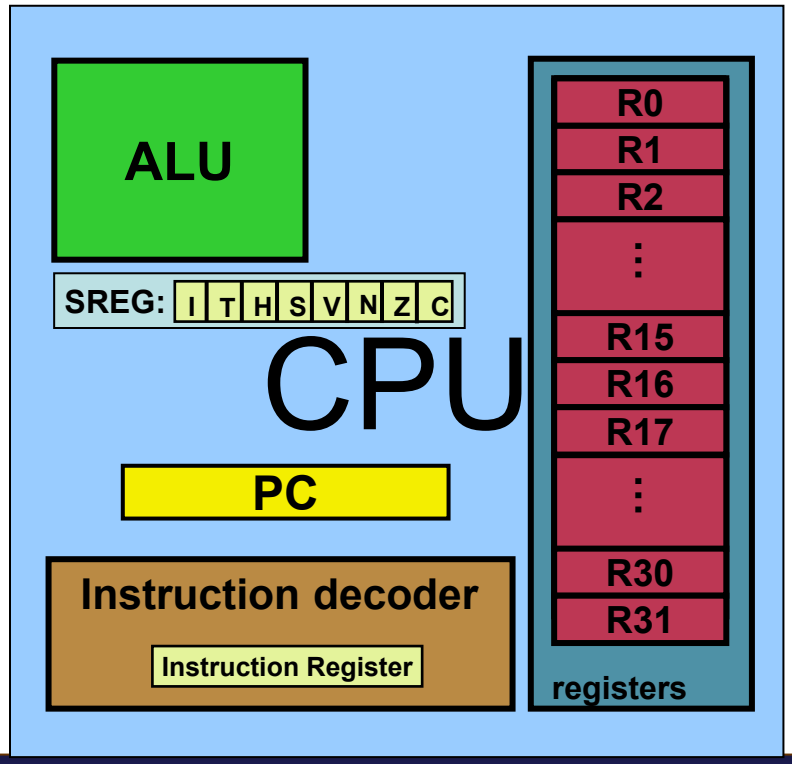
Solution:

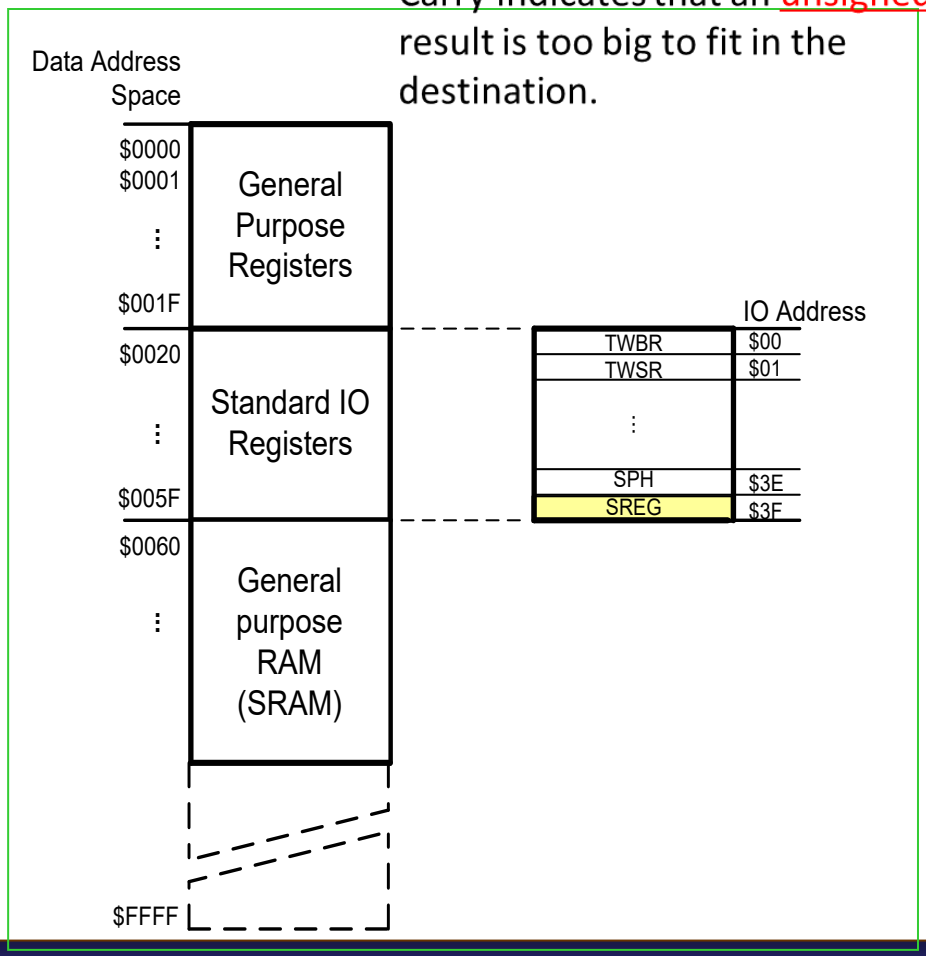
```
IN  R20,PINC   ;R20 = PINC
IN  R21,PIND   ;R21 = PIND
ADD R20,R21    ;R20 = R20 + R21
STS 0x90,R20   ;[0x90] = R20
```

Status Register (SREG)

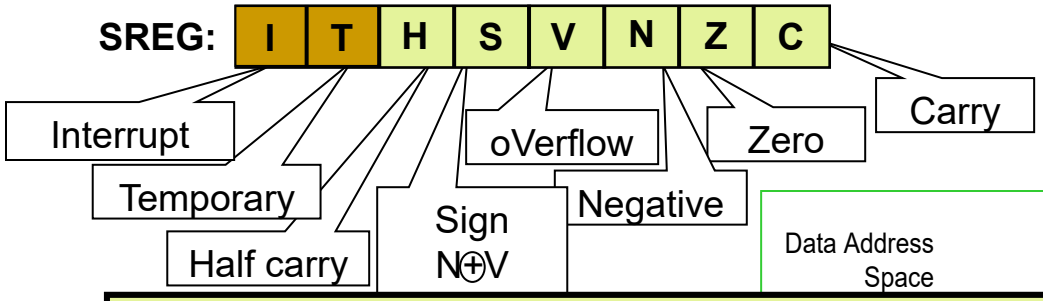


Overflow indicates that a signed result is too big or too small to fit in the destination;
Carry indicates that an unsigned result is too big to fit in the destination.





Status Register (SREG)



Data Address Space

dress

AL

SREG: I

Instru

Ins

Example: Show the status of the C, H, and Z flags after the addition of 0x38 and 0x2F in the following instructions:

```
LDI   R16, 0x38       ;R16 = 0x38
LDI   R17, 0x2F       ;R17 = 0x2F
ADD   R16, R17        ;add R17 to R16
```

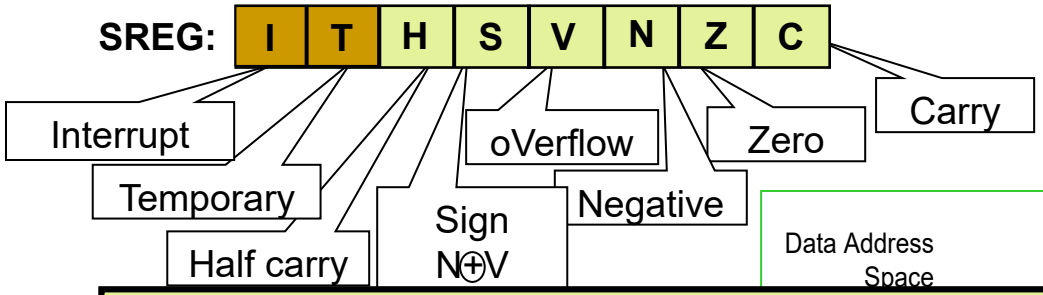
Solution:

	1	
	$38	0011 1000
+	$2F	0010 1111
	$67	0110 0111

R16 = 0x67

C = 0 because there is no carry beyond the D7 bit.
H = 1 because there is a carry from the D3 to the D4 bit.
Z = 0 because the R16 (the result) has a value other than 0 after the addition.

Status Register (SREG)



Data Address Space

dress

AL

SREG:

Instru

Ins

Example: Show the status of the C, H, and Z flags after the addition of 0x9C and 0x64 in the following instructions:

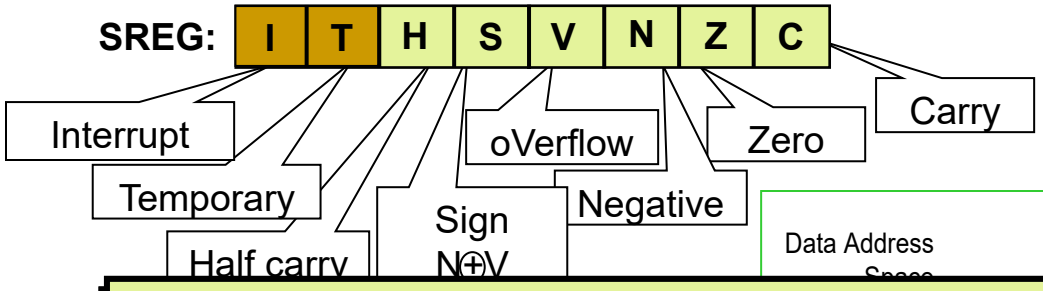
```
LDI     R20, 0x9C
LDI     R21, 0x64
ADD     R20, R21        ;add R21 to R20
```

Solution:

```
          1
  $9C     1001 1100
+ $64     0110 0100
 $100   1 0000 0000      R20 = 00
```

C = 1 because there is a carry beyond the D7 bit.
H = 1 because there is a carry from the D3 to the D4 bit.
Z = 1 because the R20 (the result) has a value 0 in it after the addition.

Status Register (SREG)



Data Address

dress

AL

SREG:

Instru

Ins

Example: Show the status of the C, H, and Z flags after the subtraction of 0x23 from 0xA5 in the following instructions:

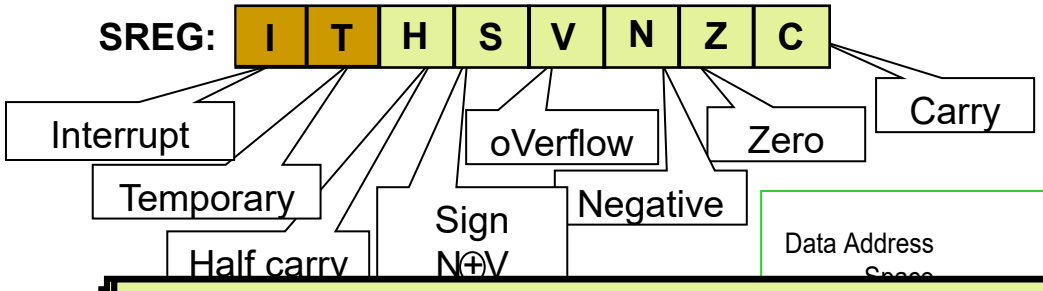
```
LDI     R20, 0xA5
LDI     R21, 0x23
SUB     R20, R21        ;subtract R21 from R20
```

Solution:

	$A5	1010 0101	
-	$23	0010 0011	
	$82	1000 0010	R20 = $82

C = 0 because R21 is not bigger than R20 and there is no borrow from D8 bit.
Z = 0 because the R20 has a value other than 0 after the subtraction.
H = 0 because there is no borrow from D4 to D3.

Status Register (SREG)



Example: Show the status of the C, H, and Z flags after the subtraction of 0x73 from 0x52 in the following instructions:

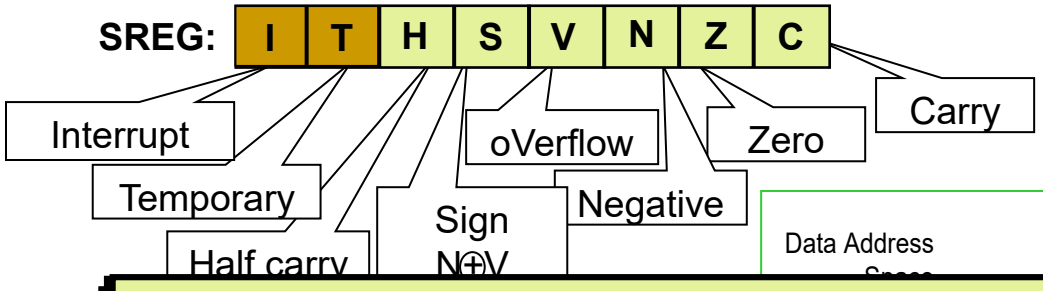
```
LDI     R20, 0x52
LDI     R21, 0x73
SUB     R20, R21        ;subtract R21 from R20
```

Solution:

	$52	0101 0010	
-	$73	0111 0011	
	$DF	1101 1111	R20 = $DF

C = 1 because R21 is bigger than R20 and there is a borrow from D8 bit.
Z = 0 because the R20 has a value other than zero after the subtraction.
H = 1 because there is a borrow from D4 to D3.

Status Register (SREG)



Data Address

Example: Show the status of the C, H, and Z flags after the subtraction of 0x9C from 0x9C in the following instructions:

```
LDI     R20, 0x9C
LDI     R21, 0x9C
SUB     R20, R21        ;subtract R21 from R20
```

Solution:

	$9C	1001 1100
-	$9C	1001 1100
	$00	0000 0000

R20 = $00

C = 0 because R21 is not bigger than R20 and there is no borrow from D8 bit.
Z = 1 because the R20 is zero after the subtraction.
H = 0 because there is no borrow from D4 to D3.

Status Register (SREG)

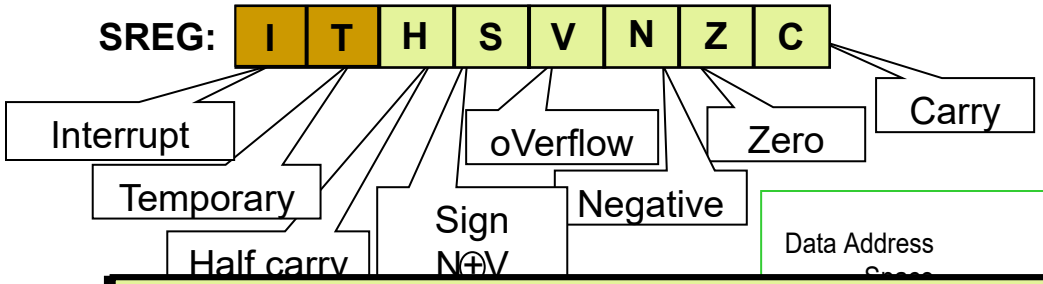


Table 2-5: AVR Branch (Jump) Instructions Using Flag Bits

Instruction	Action
BRLO	Branch if C = 1
BRSH	Branch if C = 0
BREQ	Branch if Z = 1
BRNE	Branch if Z = 0
BRMI	Branch if N = 1
BRPL	Branch if N = 0
BRVS	Branch if V = 1
BRVC	Branch if V = 0

Example: Show the status of the C, H, and Z fla
subtraction of 0x9C from 0x9C in the following

```
LDI     R20, 0x9C
LDI     R21, 0x9C
SUB     R20, R21        ;subtract R21 from R20
```

Solution:

	$9C	1001 1100
-	$9C	1001 1100
	$00	0000 0000

R20 = $00

C = 0 because R21 is not bigger than R20 and there is no borrow from D8 bit.
Z = 1 because the R20 is zero after the subtraction.
H = 0 because there is no borrow from D4 to D3.

Assembler Directives

.EQU and .SET

- .EQU *name* = *value*
  - *Example:*

```
.EQU    COUNT = 0x25
LDI     R21, COUNT              ;R21 = 0x25
LDI     R22, COUNT + 3          ;R22 = 0x28
```

- .SET *name* = *value*
  - *Example:*

```
.SET    COUNT = 0x25
 LDI    R21, COUNT              ;R21 = 0x25
 LDI    R22, COUNT + 3          ;R22 = 0x28
.SET    COUNT = 0x19
 LDI    R21, COUNT              ;R21 = 0x19
```

Assembler Directives

.INCLUDE

- .INCLUDE "*filename.ext*"

Table 2-6: Some of the common AVRs and their include files

MEGA		TINY		Special Purpose	
Mega8	m8def.inc	Tiny11	tn11def.inc	90CAN32	can32def.inc
Mega16	m16def.inc	Tiny12	tn12def.inc	90CAN64	can64def.inc
Mega32	m32def.inc	Tiny22	tn22def.inc	90PWM2	pwm2def.inc
Mega64	m4def.inc	Tiny44	tn44def.inc	90PWM3	pwm3def.inc
Mega128	m128def.inc	Tiny85	tn85def.inc	86RF401	at86rf401def.inc
Mega256	m256def.inc				
Mega2560	m2560def.inc				

Assembler Directives

.INCLUDE

- .INCLUDE "*filename.ext*"

Table 2-6: Some of the common AVRs and their include files

MEGA		TINY		Special Purpose	
Mega8	m8def.inc	Tiny11	tn11def.inc	90CAN32	can32def.inc

M32def.inc

```
.equ    SREG    = 0x3f
.equ    SPL     = 0x3d
.equ    SPH     = 0x3e
....
.equ    INT_VECTORS_SIZE = 42    ; size in words
```

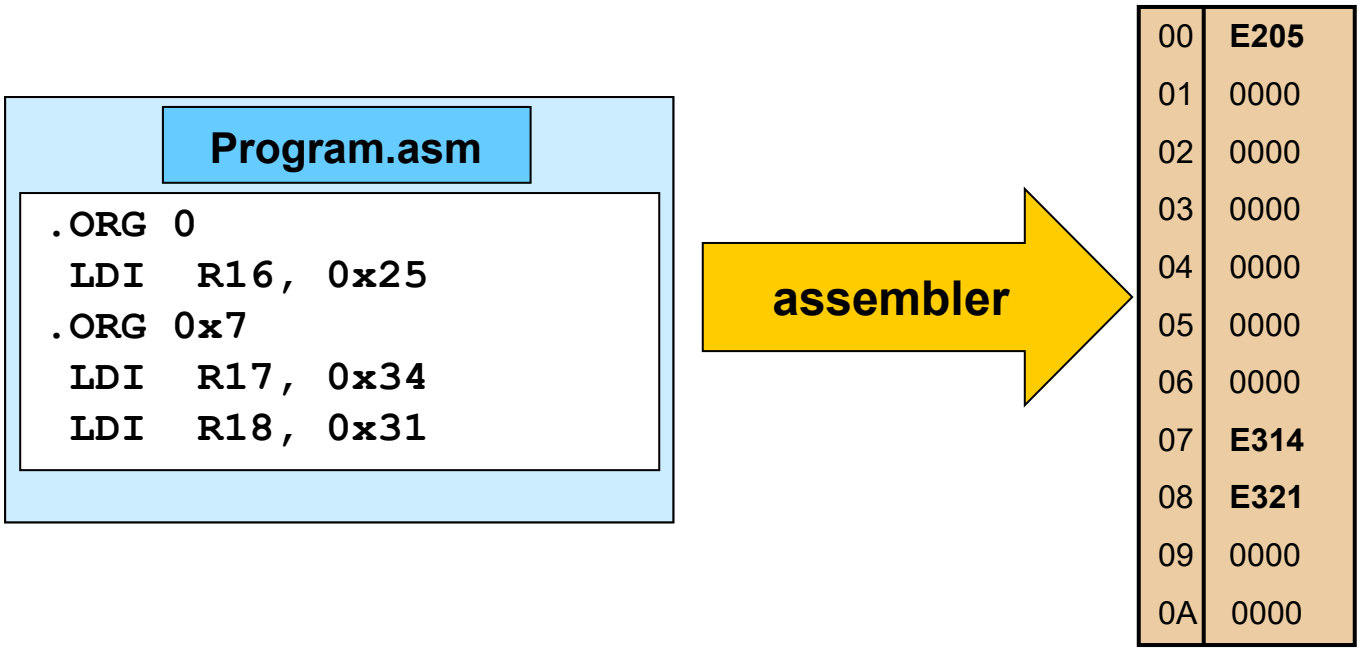
Program.asm

```
.INCLUDE "M32DEF.INC"
    LDI    R20, 10
    OUT    SPL, R20
```

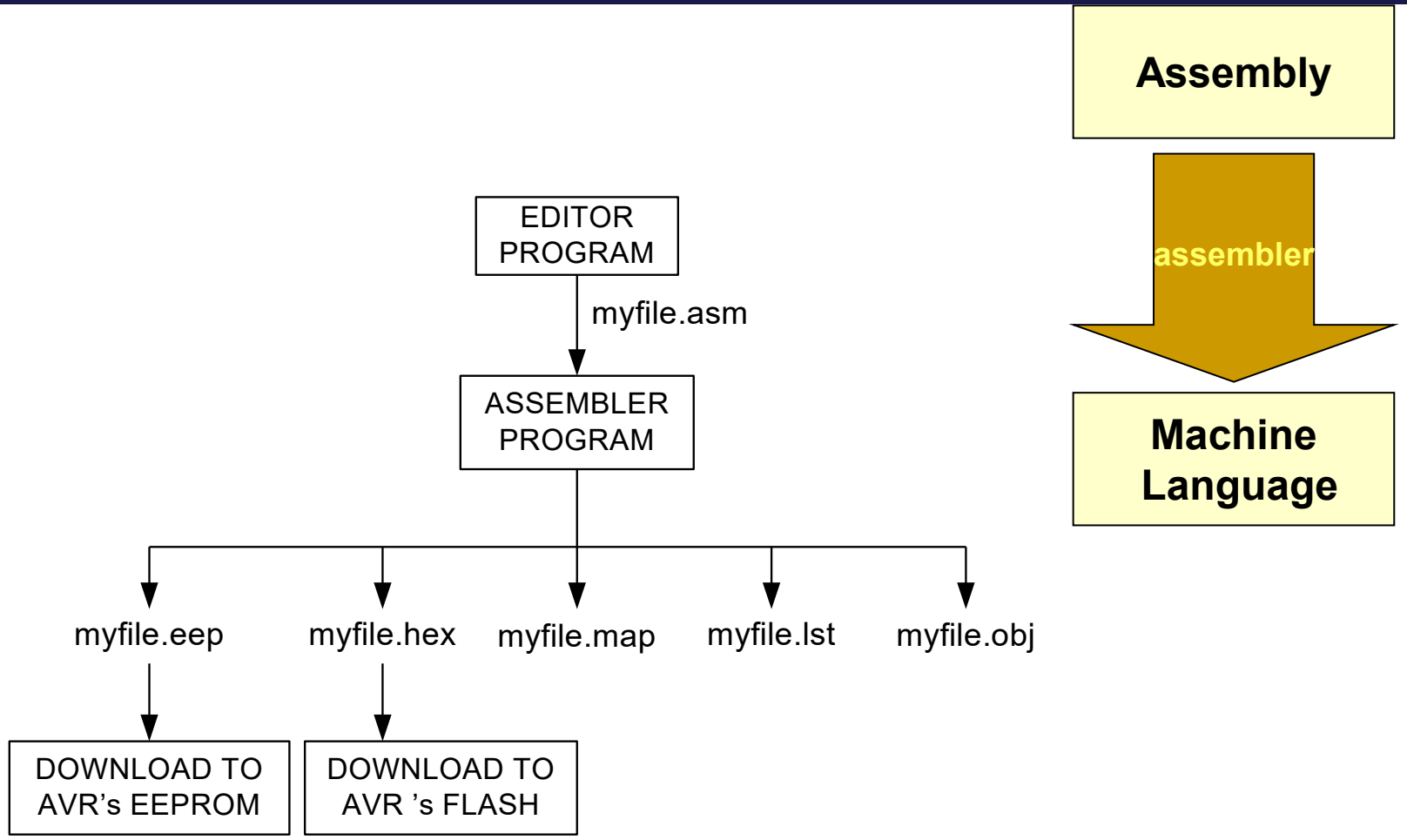
Assembler Directives

.ORG

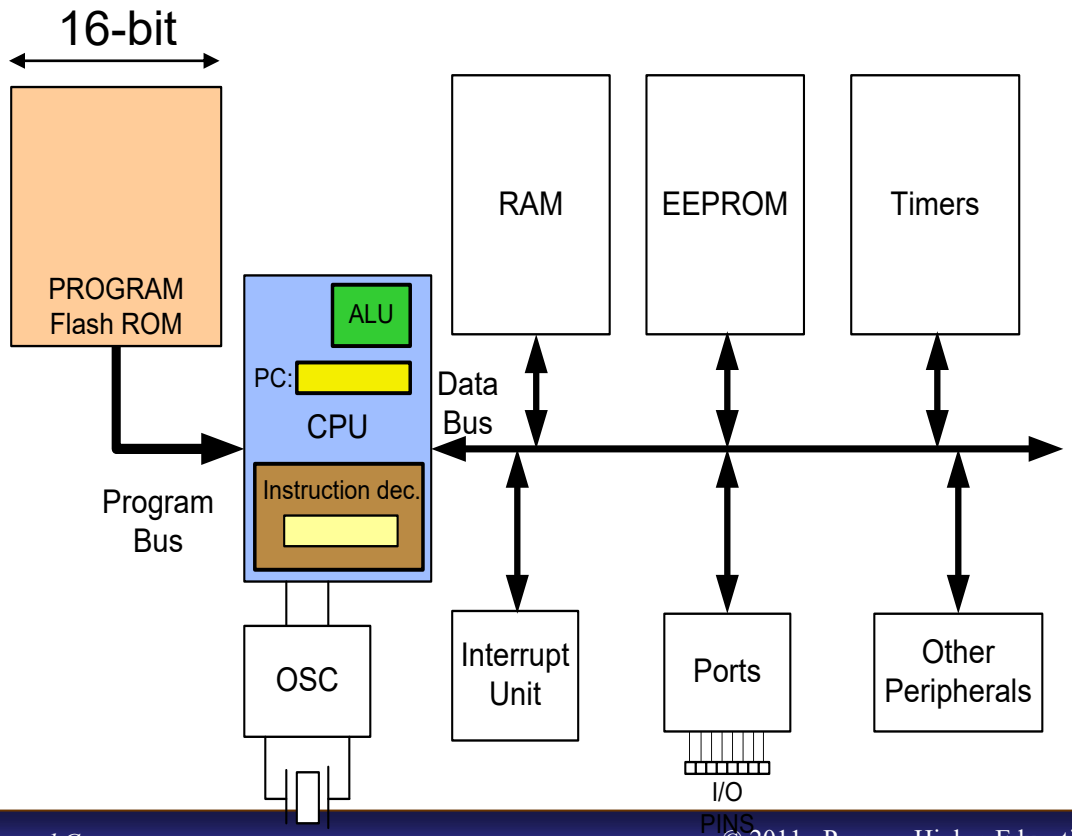
- .ORG *address*



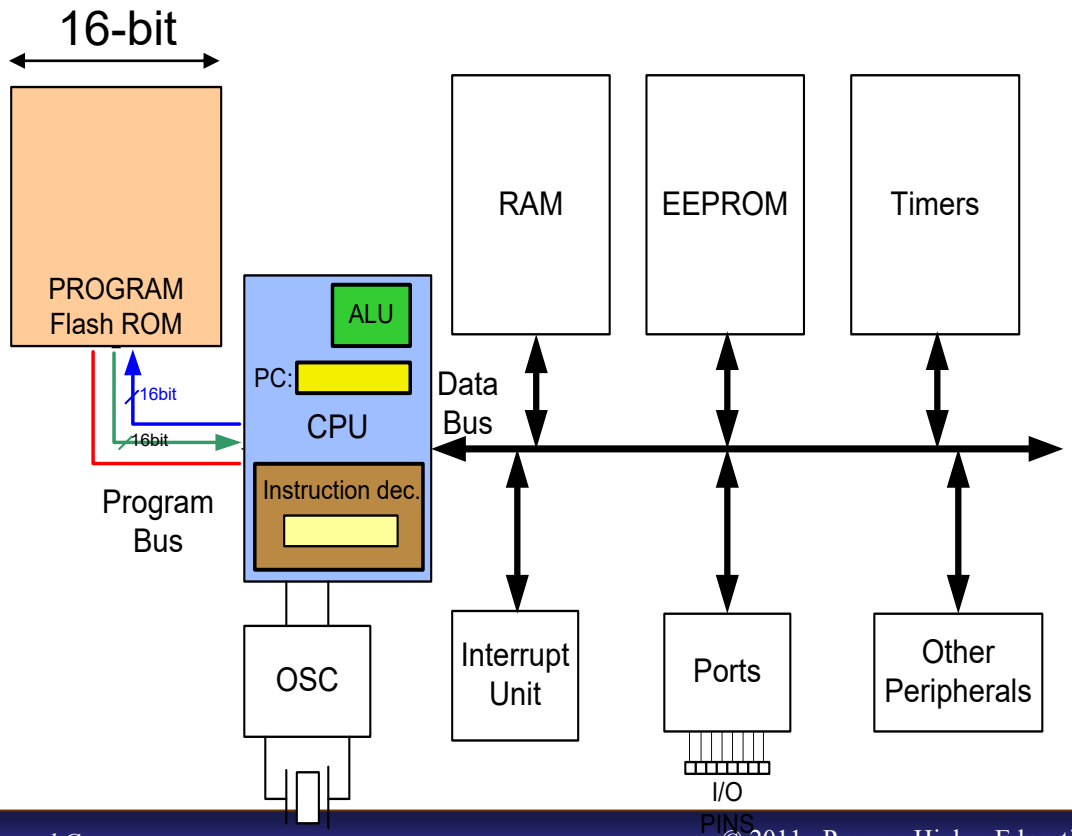
Assembler



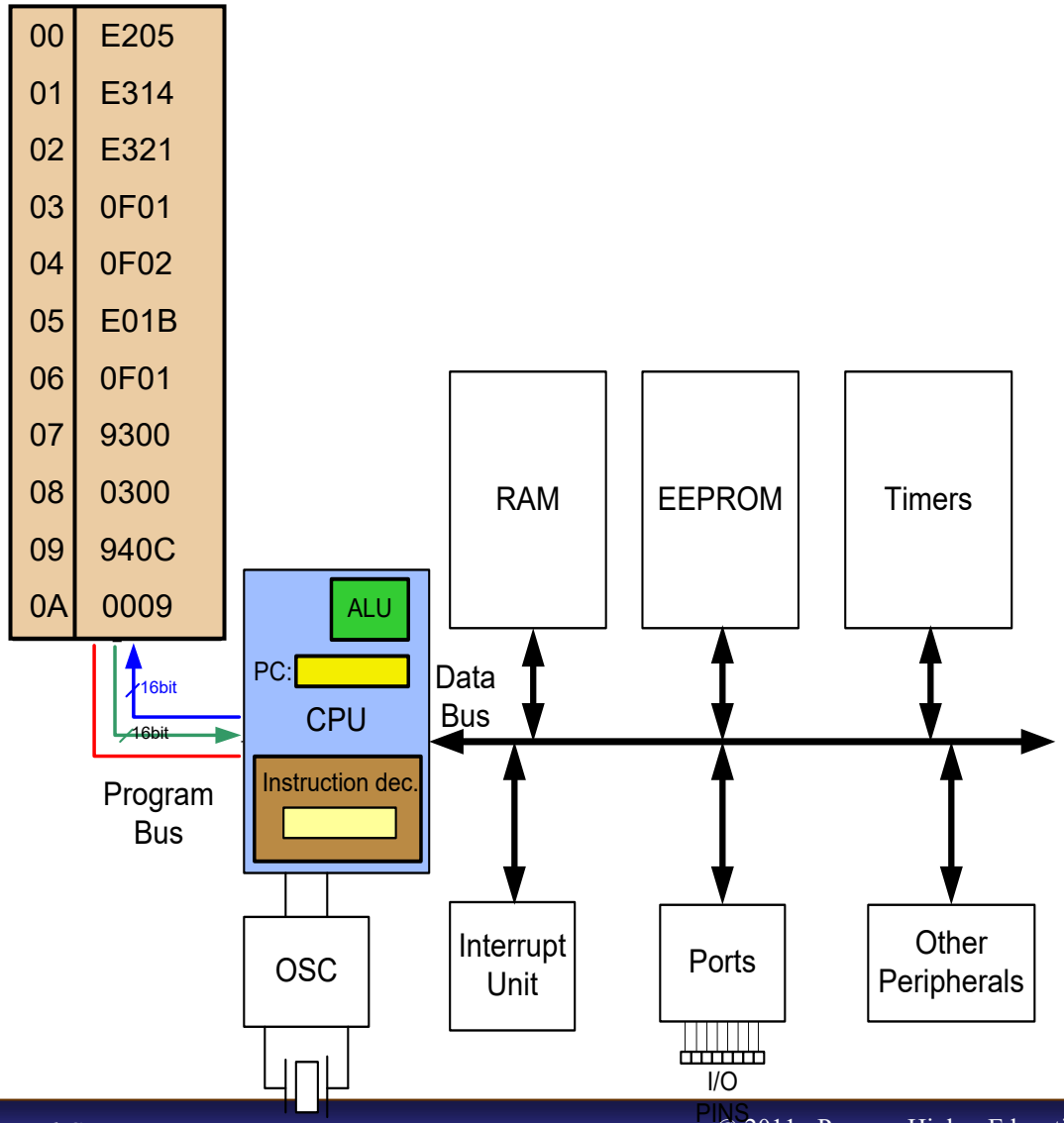
Flash memory and PC register



Flash memory and PC register

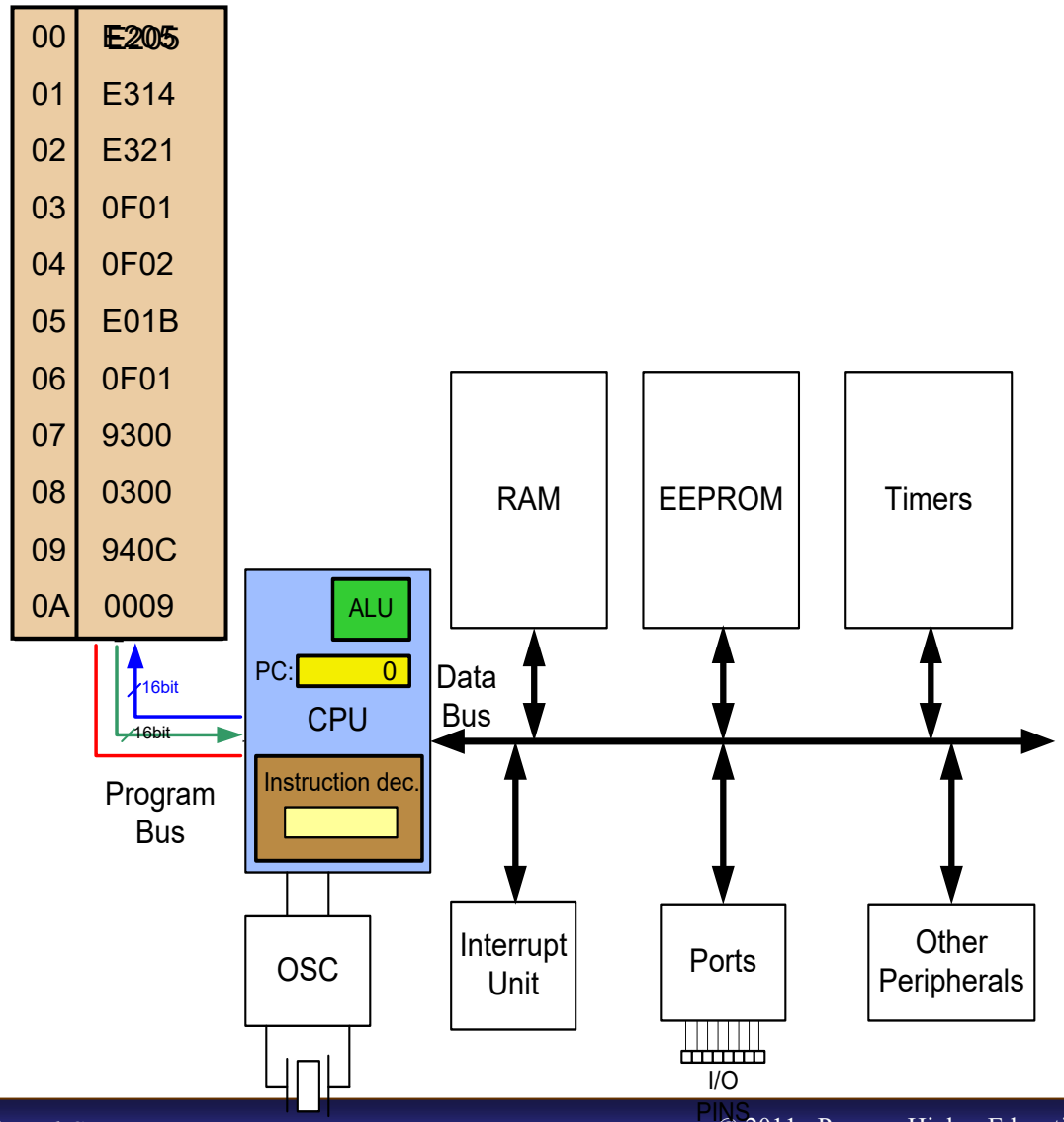


Flash memory and PC register



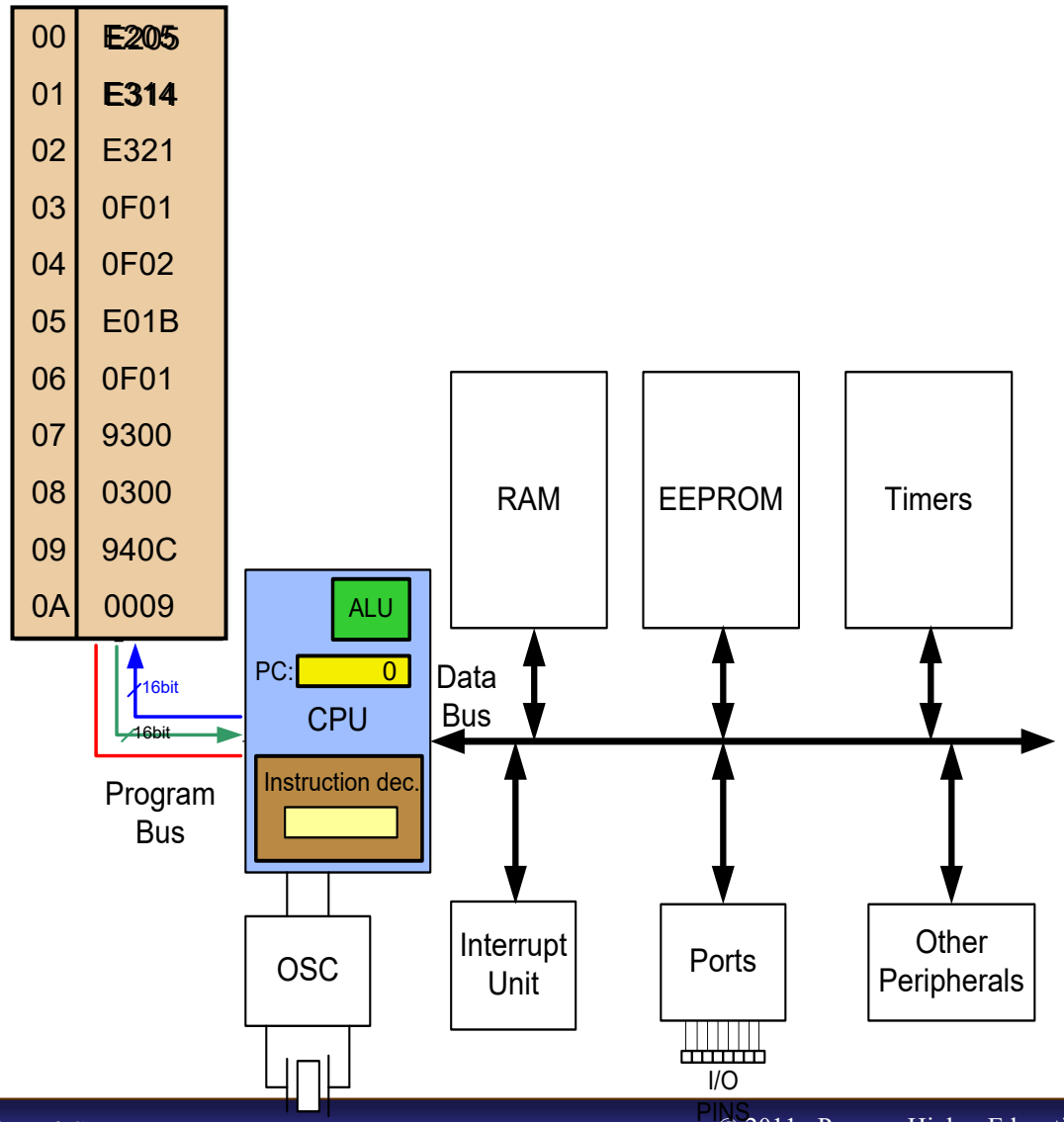
Flash memory and PC register

```
        LDI   R16, 0x25
        LDI   R17, $34
        LDI   R18, 0x31
        ADD   R16, R17
        ADD   R16, R18
        LDI   R17, 11
        ADD   R16, R17
        STS   SUM, R16
HERE:JMP  HERE
```



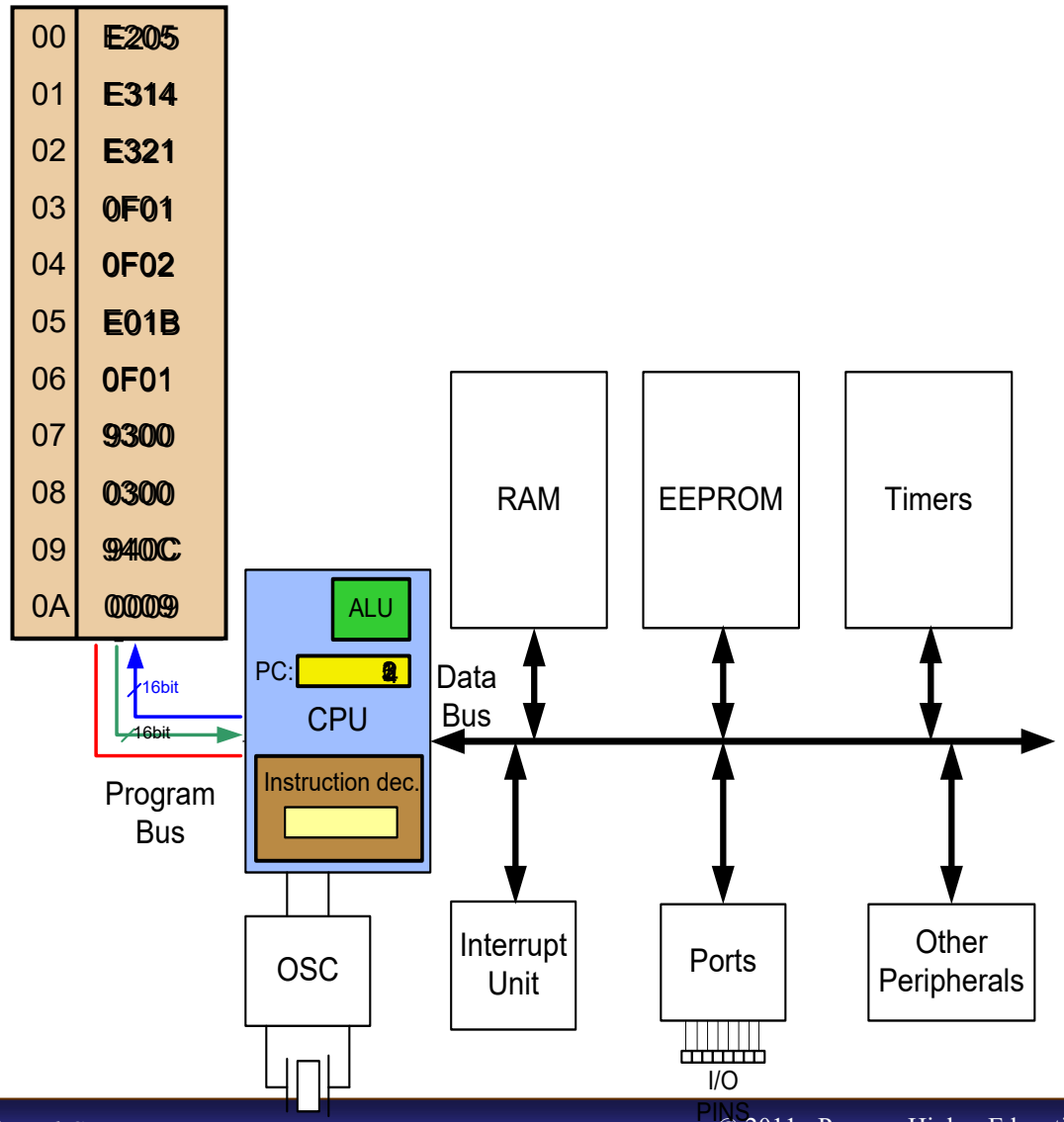
Flash memory and PC register

```
        LDI   R16, 0x25
        LDI   R17, $34
        LDI   R18, 0x31
        ADD   R16, R17
        ADD   R16, R18
        LDI   R17, 11
        ADD   R16, R17
        STS   SUM, R16
HERE:JMP  HERE
```



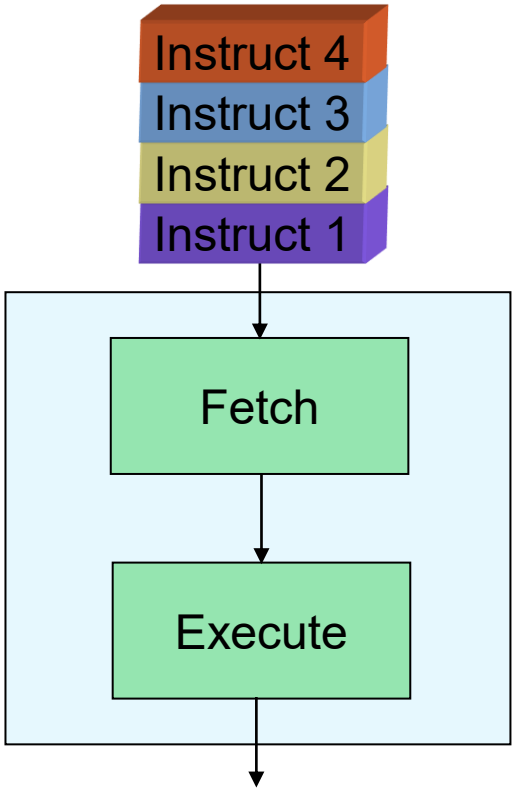
Flash memory and PC register

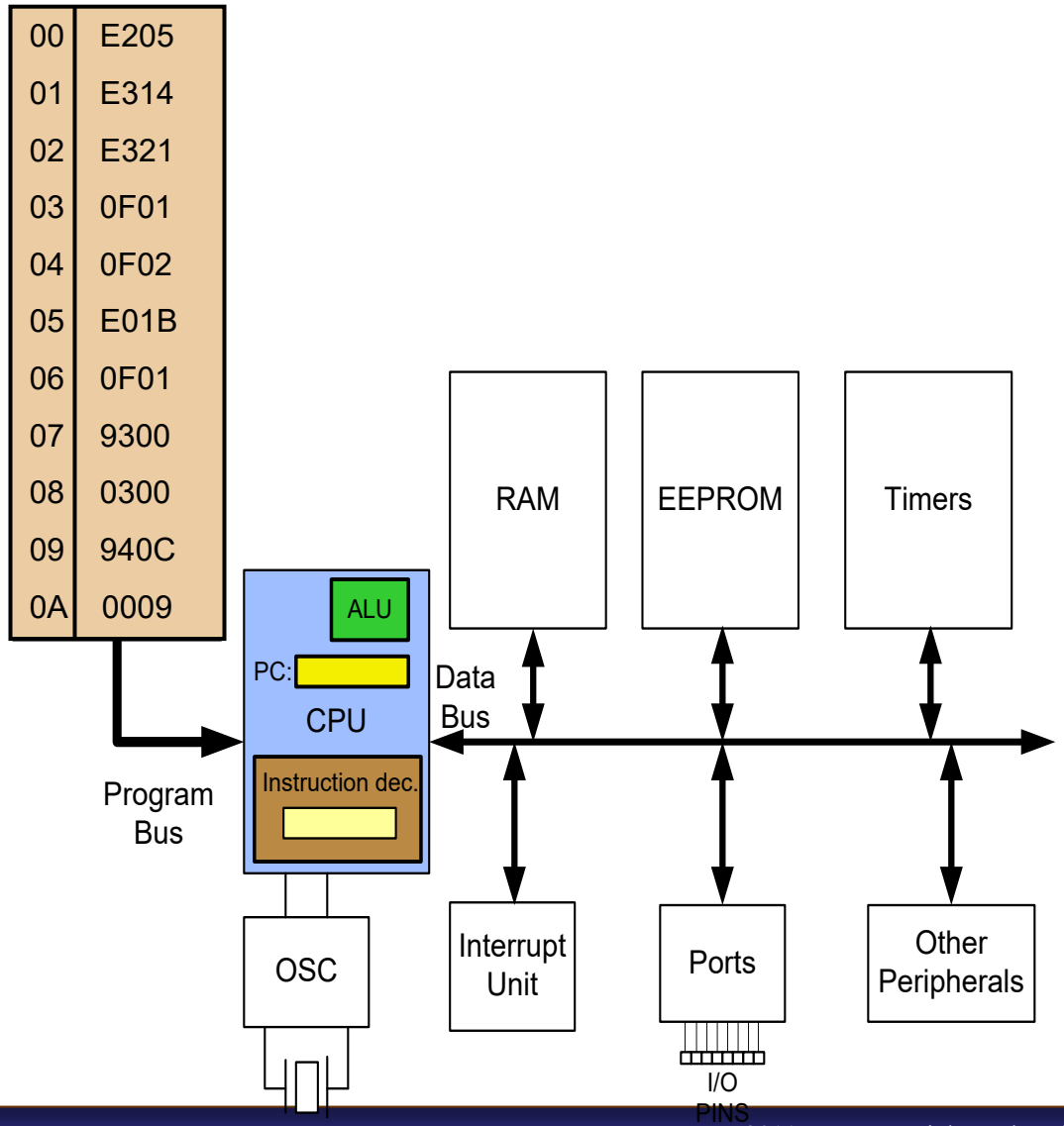
```
        LDI   R16, 0x25
        LDI   R17, $34
        LDI   R18, 0x31
        ADD   R16, R17
        ADD   R16, R18
        LDI   R17, 11
        ADD   R16, R17
        STS   SUM, R16
HERE:JMP  HERE
```



Fetch and execute

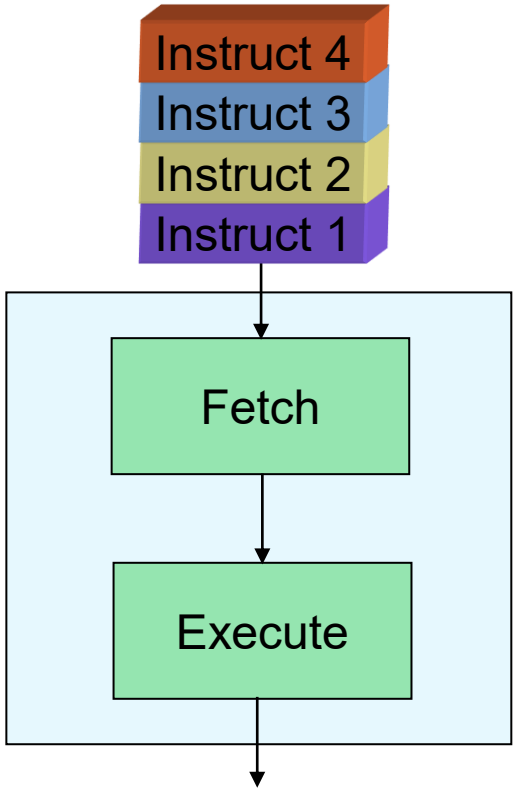
- Old Architectures

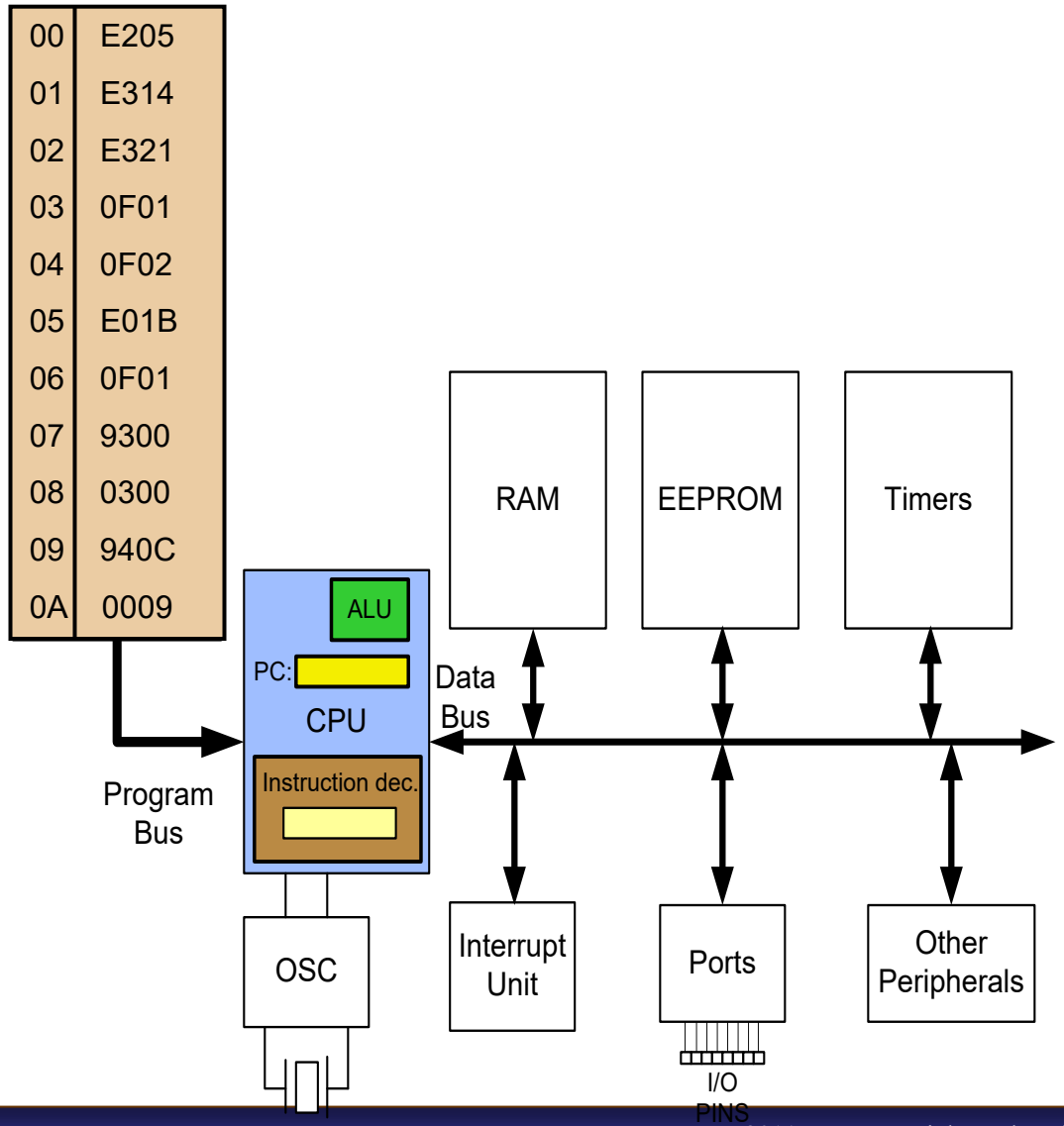




Pipelining

- Pipelining





How to speed up the CPU

- Increase the clock frequency
  - More frequency ➔ More power consumption & more heat
  - Limitations
- Change the architecture
  - Pipelining
  - RISC

Changing the architecture RISC vs. CISC

- CISC (Complex Instruction Set Computer)
  - Put as many instruction as you can into the CPU
- RISC (Reduced Instruction Set Computer)
  - Reduce the number of instructions, and use your facilities in a more proper way.

RISC architecture

- Feature 1
  - RISC processors have a fixed instruction size. It makes the task of instruction decoder easier.
    - In AVR the instructions are 2 or 4 bytes.
  - In CISC processors instructions have different lengths
    - E.g. in 8051
      - CLR C ; a 1-byte instruction
      - ADD A, #20H ; a 2-byte instruction
      - LJMP HERE ; a 3-byte instruction

RISC architecture

- Feature 2: reduce the number of instructions
  - Pros: Reduces the number of used transistors
  - Cons:
    - Can make the assembly programming more difficult
    - Can lead to using more memory

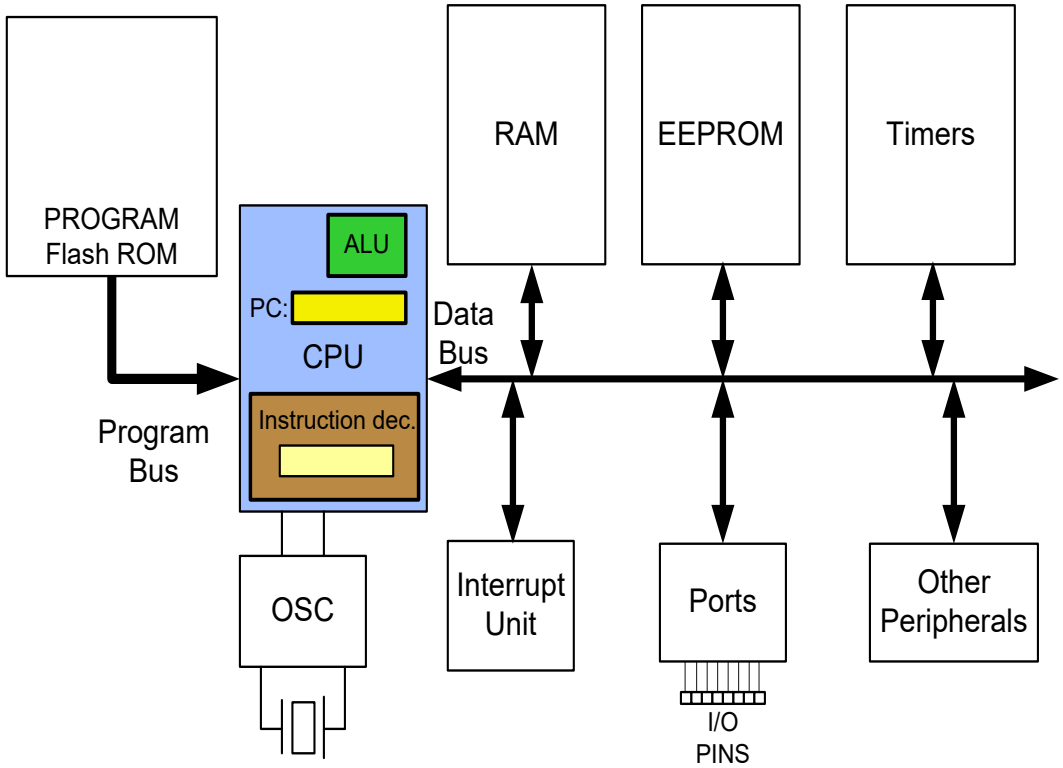
RISC architecture

- Feature 3: limit the addressing mode
  - Advantage
    - hardwiring
  - Disadvantage
    - Can make the assembly programming more difficult

RISC architecture

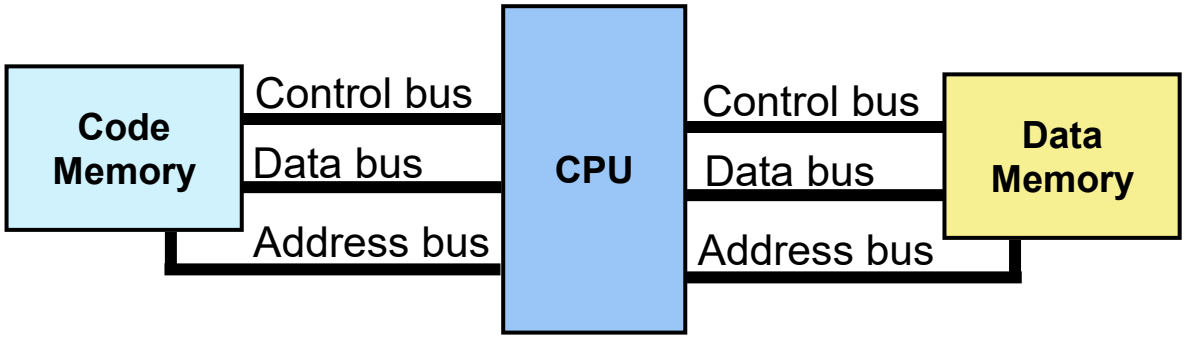
- Feature 4: Load/Store

```
LDS R20, 0x200
LDS R21, 0x220
ADD R20, R21
STS 0x230, R20
```



RISC architecture

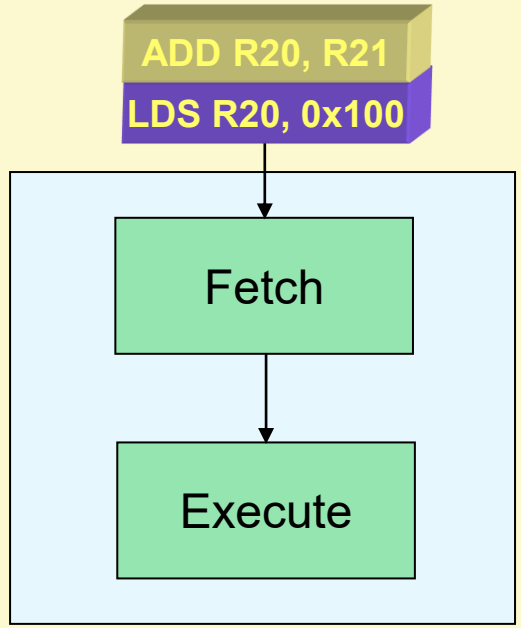
- Feature 5 (Harvard architecture): separate buses for opcodes and operands
  - Advantage: opcodes and operands can go in and out of the CPU together.
  - Disadvantage: leads to more cost in general purpose computers.



RISC architecture

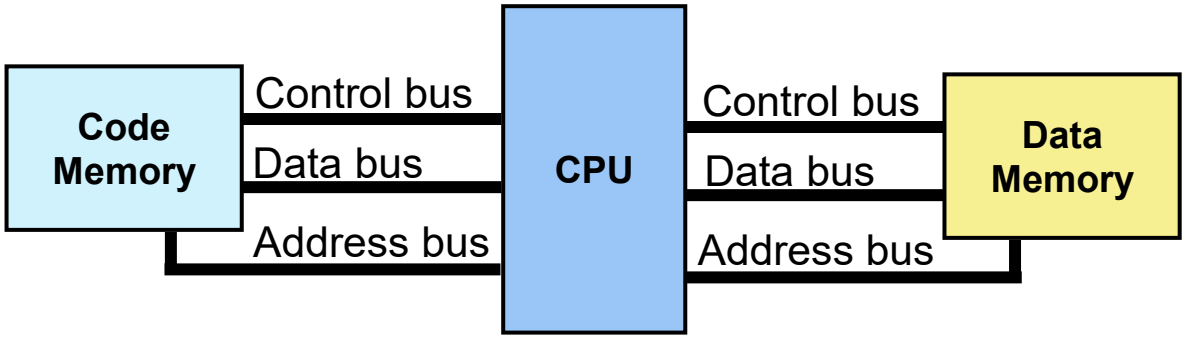
-

```
LDS R20, 0x100 ; R20 = [0x100]
ADD R20,R21    ; R20 = R20 + R21
```



PU

ers.



RISC architecture

- Feature 6: more than 95% of instructions are executed in 1 machine cycle

RISC architecture

- Feature 7
  - RISC processors have at least 32 registers. Decreases the need for stack and memory usages.
    - In AVR there are 32 general purpose registers (R0 to R31)