Q1

There are 10 32-bit and 6 16-bit registers.

(P49)

General Purpose:
- data registers (4 32-bit)
- Pointer registers (2 32-bit)
- Index registers (2 32-bit)

Control registers → (2 32-bit)
Segment registers → (6 16-bit)

data registers used for arithmetic, logical and other operations (P50)

	16-bit	8-bit	8-bit	
EAX		AH	AL	AX
EBX		BH	BL	BX
ECX		CH	CL	CX
EDX		DH	DL	DX

★ we can access them in

32-bit EAX, EBX, ECX, EDX or,
16-bit AX, BX, CX, DX or,
8-bit AH, AL, BH, BL, CH, CL, DH, DL

- AX (Accumulator): mostly used to store results of arithmetic operations but also used in I/O as we saw in lectures.

- BX (Base): mostly used to store memory address (along SI, DI, BP), also can store offset (of type memory) for DS. (slides)

- CX (counter): store loop counters.

- DX (Data) in division, it store the remainder. also we can store starting memory address of a string by LEA.

Pointers and Index registers

	16-bit	16-bit
ESI		SI
EDI		DI

	16-bit	16-bit
ESP		SP
EBP		BP

★ they could be used as 16-bit or 32-bit. they also can be used as general purpose registers.

- SI, DI: they play special role in string processing instructions (P50)

- SP, BP: they mainly used to maintain stack. (P 50)

Control registers
- IP
- flag

EIP		IP

- Instruction Pointer: it points to the next instruction to be execute (IP for 16-bit addresses and EIP for 32-bit addresses) (P 51)

- flag registers: EFLAGS has
  - 6 status flag → showing information of the most recent arithmetic/logical operation.
  - 1 control flag → useful for string operations
  - 10 system flag → they control the operation of processor.

(P 51-52)

Segment Registers

16-bit
CS
DS
SS
ES
FS
GS

★ they used for memory organization, each program can have access up to 6 segments and each rigester points to corresponding segment

- CS, code segment, where program instructions are.

- DS, Data segment, points to the data part of the program.

- SS, Stack segment, points to stack segment

- ES, FS, GS are used for extended (additional) segments if needed.

The execution cycle:

Fetch + Decode + Execut

Fetch:

1, Put memory address on address bus and activate "memory read signal"

2, Access time (the time it takes for memory to read data)

3, memory put data on data bus

4, The processor which was waiting to get the data, now can read the instruction.

★ In practice, instructions are not fetched from memory. but they mostly use cache!

Decode: Identifying the instruction. machine-language instructions follow a particular instruction-encoding schema.

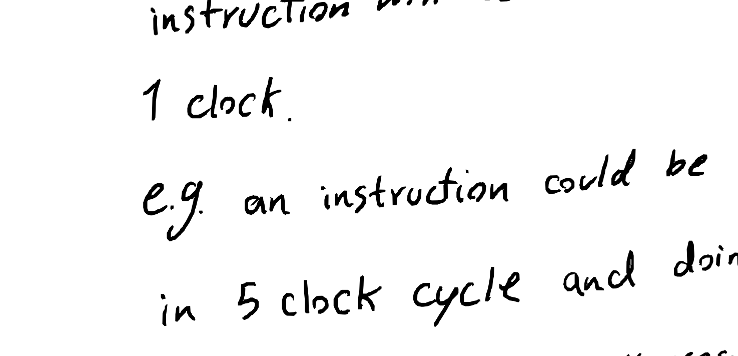
Execute:

ALU (Arithmetic and Logic unit) is responsible for arithmetic and logical operations.

There is also a Control circuitry, which is responsible for timing control and to instruct the internal hardware components to perform a specific operation.

The system clock:

it provides a way to synchronize the operations of the system.



1 clock cycle

measured in Hz

★ It doesn't mean that each instruction will be done at 1 clock.

e.g. an instruction could be done in 5 clock cycle and doing so with 1 GHz CPU will result in,

each clock is $\frac{1}{1\,GHz} = \frac{1}{10^9\,Hz} = 1\,ns$

$1 \cdot 5 = 5\,ns$ takes for that instruction to be completed.

Q2,

a,

$$\begin{array}{r} 01111010 \\ \& \; 11011001 \\ \hline (01011000)_2 \end{array}$$

$\equiv (88)_d$

b,

~ 11110101

$= (00001010)_2$

$\equiv (10)_d$

c,

$$\begin{array}{r} 11001101 \\ \hline (11111111)_2 \end{array}$$

now

$\sim (11111111)_2$

$= (00000000)_2$

$\equiv (0)_d$

d,

first 8 bits 15

$$\begin{array}{r} 01000 \\ 01111 \\ \hline 01000 \end{array}$$

then

$$\begin{array}{r} 01000 \\ \& \; 10000 \\ \hline 00000 \end{array}$$

at the end

$\sim (00000)$

$= (11111)_2 = (31)_d$

e,

32
16
8
4
2
1

40

$111101 \gg 3$

00011111

$\sim (000111)$

$= (111000)_2 = (56)_d$

f,

$$\begin{array}{r} 101110 \\ 101011 \\ \hline 111110 \ll 4 \end{array}$$

1111100000

$= (100000)_2 = (32)_d$

Q 3

1,

$(-80)_d = -128 + (32 + 16)$

$= (1011\ 0000)_2$

$(42)_d = 32 + 8 + 2$

$= (0010\ 1010)_2$

$$\begin{array}{r} 1011\ 0000 \\ +\ 0010\ 1010 \\ \hline (1101\ 1010)_2 \end{array}$$

2,

$(-99)_d = -128 + (16 + 8 + 4 + 1)$

$= (1001\ 1101)_2$

$(-20) = -128 + (64 + 32 + 8 + 4)$

$= (1110\ 1100)_2$

$-99 + (-20)$

$$\begin{array}{r} 1001\ 1101 \\ +\ 1110\ 1100 \\ \hline 1\ 1000\ 1001 \end{array}$$

answer is

$1000\ 1001$

3,

$(60)_d = 32 + 16 + 8 + 4$

$= (0011\ 1100)_2$

$(-70)_d = -128 + (32 + 16 + 8 + 2)$

$= (1011\ 1010)_2$

$60 + (-70)$

$$\begin{array}{r} 0011\ 1100 \\ +\ 1011\ 1010 \\ \hline 1111\ 0110 \end{array}$$

answer is

$1111\ 0110$

4,

5,

$(52)_d = 32 + 16 + 4$

$= (0011\ 0100)_2$

$(5)_d = 4 + 1$

$= (0000\ 0101)_2$

$$\begin{array}{r} 0011\ 0100 \\ \times\ 0000\ 0101 \\ \hline 0011\ 0100 \\ +\ 0000000000 \\ +\ 0011010000 \\ +\ \cdots\ 0 \\ \cdots\ 0 \\ \vdots \\ \hline (1000\ 0100)_2 \end{array}$$

the answer is

$(0000\ 0100)_2$

6,

Q 4

a,

$$(2304)_d =$$

$$(\cdot 1001\ 0000\ 0000)_2$$

therefore

$$(2304)_{10} = (1.001\ 0000\ 0000 \times 2^{11})_2$$

normalized form

$S = 0$ since 2304 is positive

exp = 11

bias exp with $K = 7$,

$$2^K - 1 = 2^7 - 1 = 63$$

So,

$$E = 11 + 63 = (74)_d$$

$$= (1001\ 010)_2$$ * 7-bit!

$$frac = (0010\ 0000)_2$$ * 8-bit

* since leading 1 is implied

$$n = (1 + frac)_d$$

$$n = 1.f_{n-1} \cdots f_0$$

$$n = 1 + 1 \cdot 2^{-3} = 1.125$$

[S | E | frac]

= 0 1001010 00100000