Computer Architecture and System Programming Laboratory

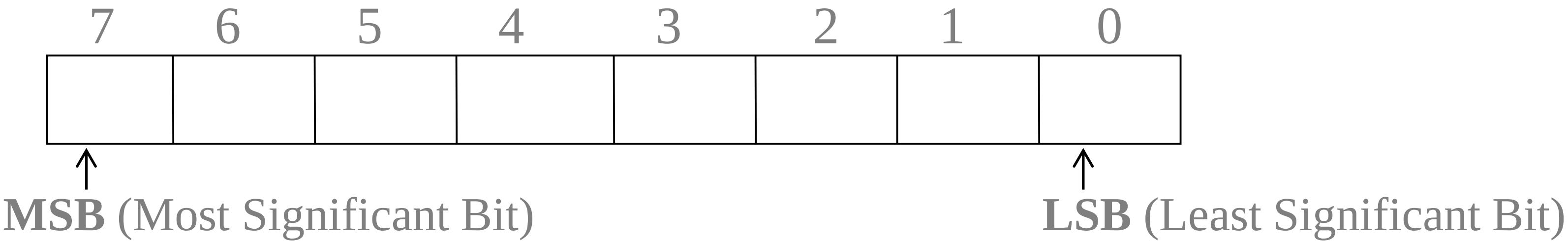
TA Session 1

Memory basics
Register file
Assembly language basics
MOV + RESB code example

Data Representation Basics

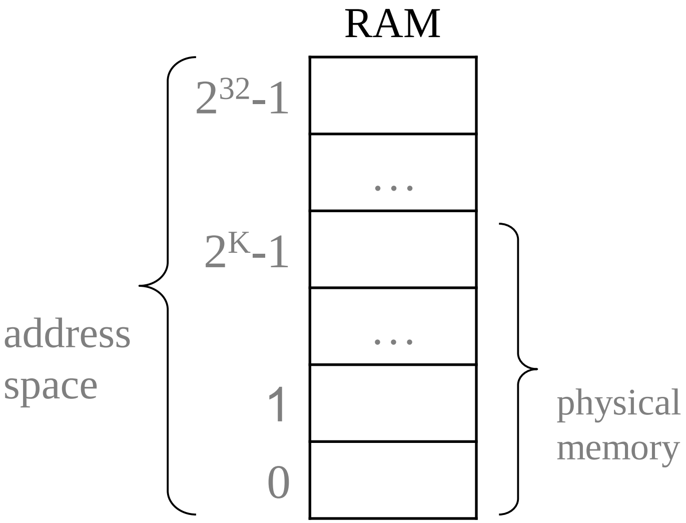
- **bit – b**asic **b**inary **i**nformation uni**t**: 0/1

- **byte –** sequence of 8 bits:

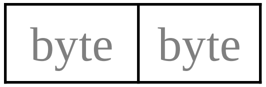

7
6
5
4
3
2
1
0
MSB (Most Significant Bit)
LSB (Least Significant Bit)

- **Main Memory (RAM)** is **array of bytes**, addressed by 0 to 2^{32}-1.

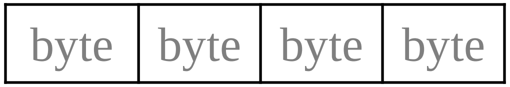
  2^{32} bytes = $4 \cdot 2^{10 \cdot 3}$ bytes = 4 G bytes



- **word –** sequence of 2 bytes



- **dword –** sequence of 4 bytes



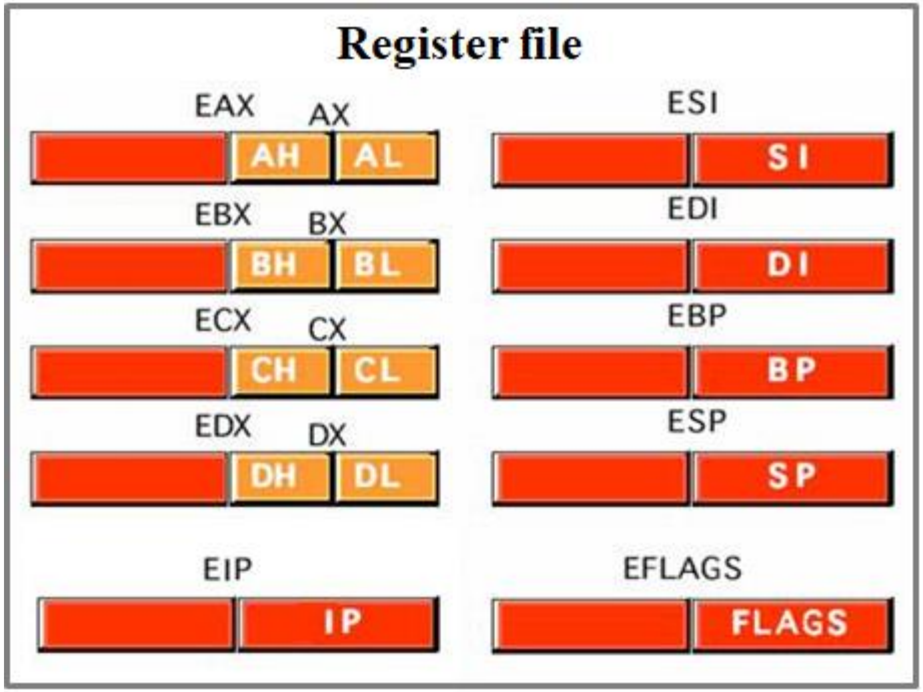
Registers

Register file **-** CPU unit which contains 32-bit registers.

general purpose registers

EAX, EBX, ECX, EDX

(Accumulator, Base, Counter, Data)



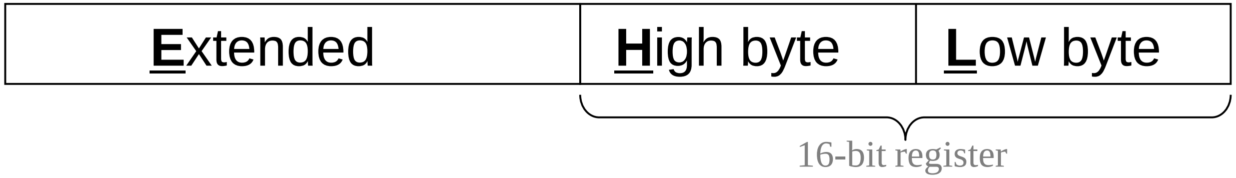
index registers

ESP, EBP, ESI, EDI

(Stack pointer - contains address of last used dword in the stack, Base pointer – contains address of current activation frame, Source index, Destination Index)

flag register / status register

EFLAGS



Extended | **H**igh byte | **L**ow byte

16-bit register

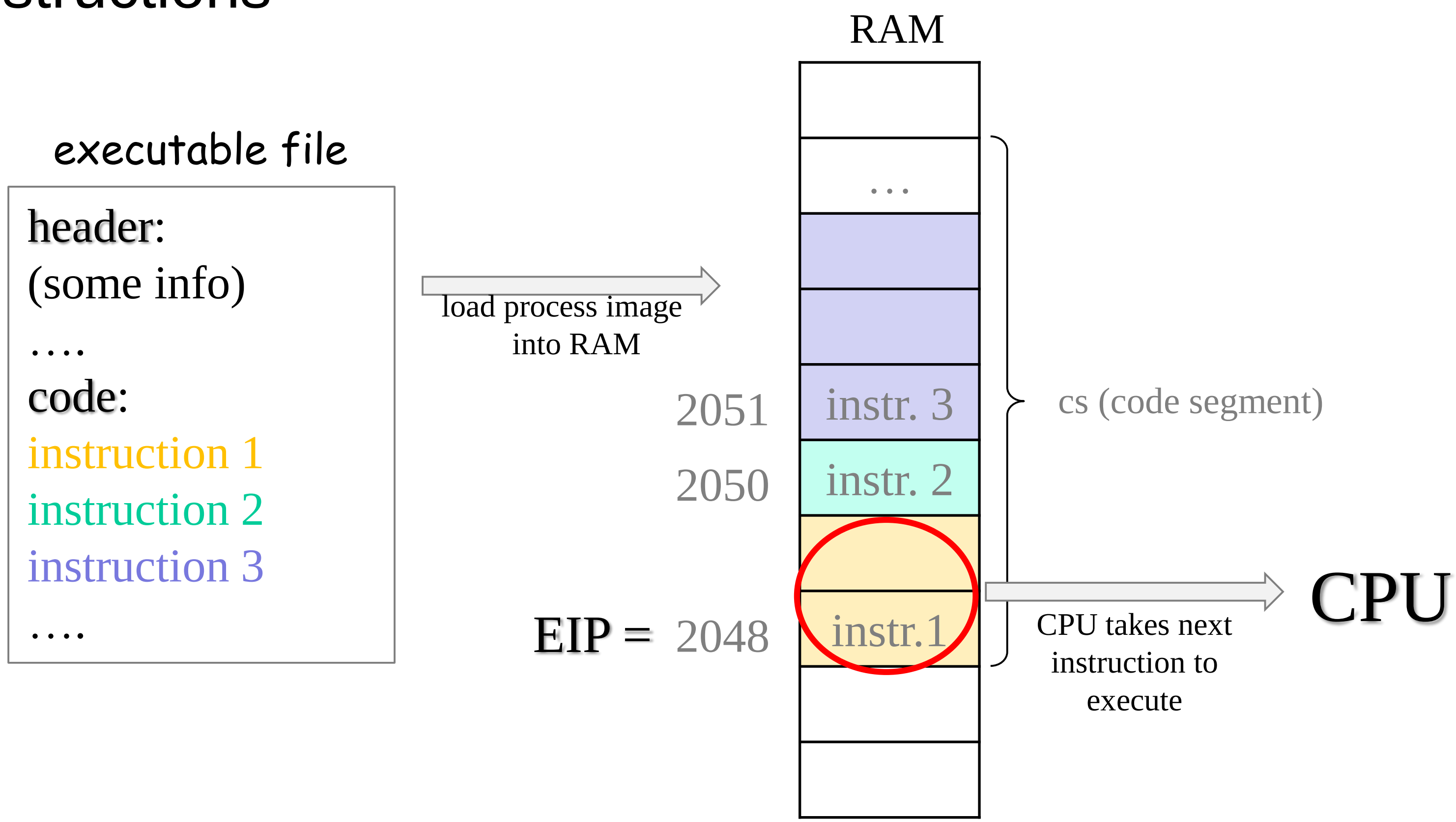
Instruction Pointer / Program Counter EIP

- contains address of next instruction that is going to be executed (at run time)
- automatically is changed by jump, procedure call, and return instructions

Full list of registers can be found here.

Instruction pointer EIP

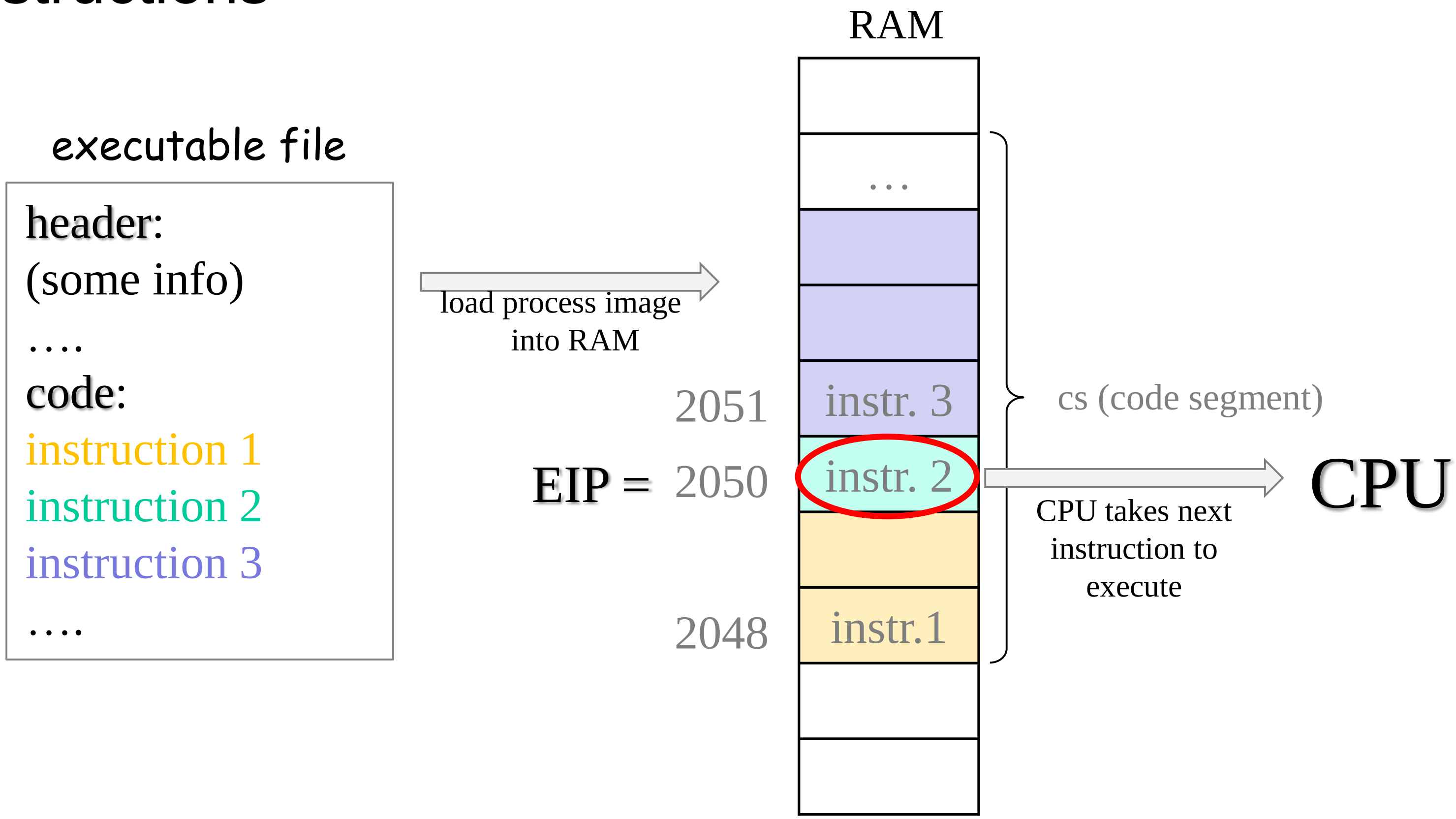
- contains address of next instruction that is going to be executed (at run time)
- automatically is changed by jump, procedure call, and return instructions



Full list of registers can be found here.

Instruction pointer EIP

- contains address of next instruction that is going to be executed (at run time)
- automatically is changed by jump, procedure call, and return instructions



Full list of registers can be found here.

Assembly Language Program

- consists of processor instructions, assembler directives, and data
- translated by assembler into machine language instructions (binary code) that can be loaded into memory and executed
- **NASM - N**etwide **As**se**m**bler - is assembler for x86 architecture

Example:

assembly code:

```
mov al, 0x61   ; load al with 97 decimal (61 hex)
```

binary code:

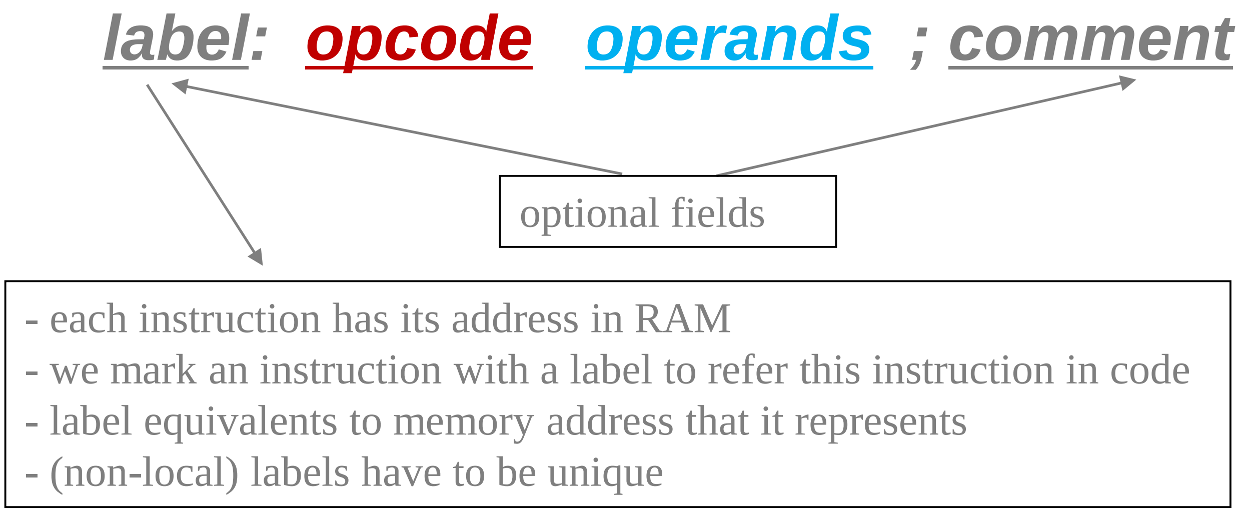
```
10110000    01100001
```

AL register

0	1	1	0	0	0	0	1

1011	a binary code (opcode) of instruction 'MOV'
0	specifies if data is byte (‘0’) or full size 16/32 bits (‘1’)
000	a binary identifier for a register 'AL'
01100001	a binary representation of 97 decimal (97_d = (int)(97/16)*10 + (97%16 converted to hex digit) = 61_h)

Basic structure of an assembly-language instruction



label**:* ***opcode ***operands*** ***; comment***

optional fields

- each instruction has its address in RAM
- we mark an instruction with a label to refer this instruction in code
- label equivalents to memory address that it represents
- (non-local) labels have to be unique

Example:

```
movLabel: mov al, 0x61
…
jmp movLabel
```

JMP tells the processor that the next instruction to be executed is located at the label that is given as part of jmp instruction.

Notes:
- **backslash (\)** : if a line ends with backslash, the next line is considered to be a part of the backslash-ended line
- **no restrictions on white space** within a line

Example of assembler directive

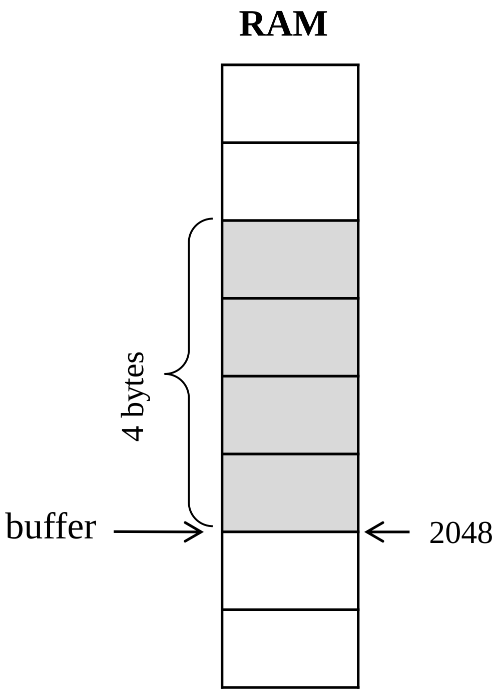
buffer: resb 4   ; reserves 4 bytes

mov buffer, 2 = mov 2048, 2 ☹

mov [buffer], 2 = mov [2048], 2 ☹

Appropriate C code:

```c
int buffer;
buffer = 2;
```



mov dword [buffer], 2   = mov dword [2048], 2 ☺

or *mov byte [buffer], 2* ↗

```c
char buffer[4];
buffer[0] = 2;
```

or *mov word [buffer], 2* ↘

```c
short int buffer[2];
buffer[0] = 2;
```

or …

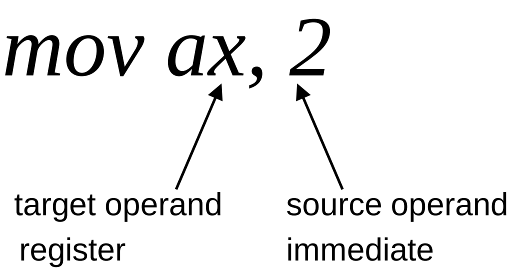
Instruction Arguments

A typical instruction has two operands:

- target operand (left)
- source operand (right)

3 kinds of operands:

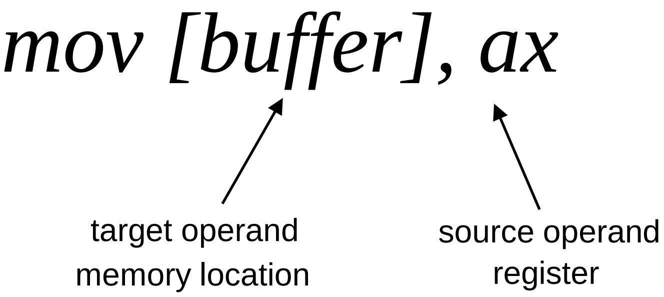
- immediate
- register
- memory location



mov ax, 2

target operand
register

source operand
immediate



mov [buffer], ax

target operand
memory location

source operand
register

Note that x86 processor does not allow both operands be memory locations.

~~mov [var1],[var2]~~