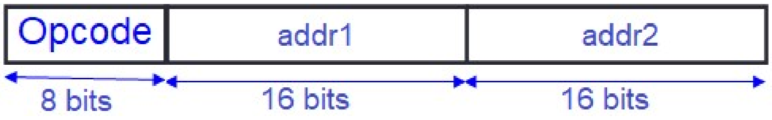
Computer Structure and Language

Hamid Sarbazi-Azad
Department of Computer Engineering
Sharif University of Technology (SUT)
Tehran, Iran

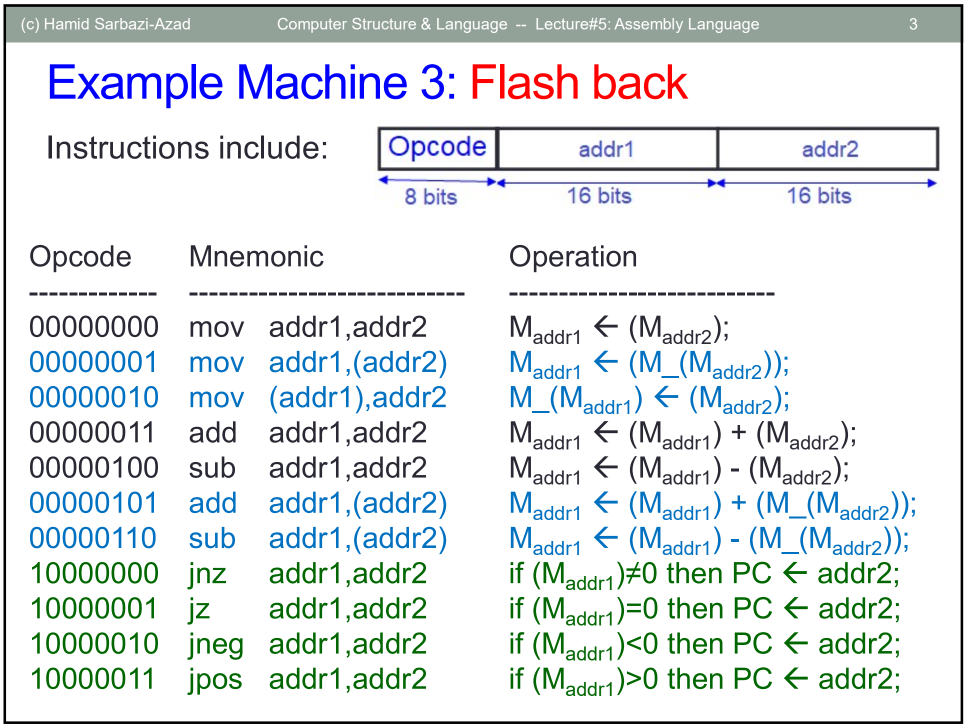
Example Machine 3: Flash Back

In a 2-address machine, we have:

- Main memory size: 2^{16} addressable units (each 8 bits)
- Word size: 16 bits, unaligned, big endian
- Addressing modes: Memory Direct, Memory Indirect
- Conditional jump instructions to make loops
- The instruction format shown below:



Opcode	addr1	addr2
8 bits	16 bits	16 bits

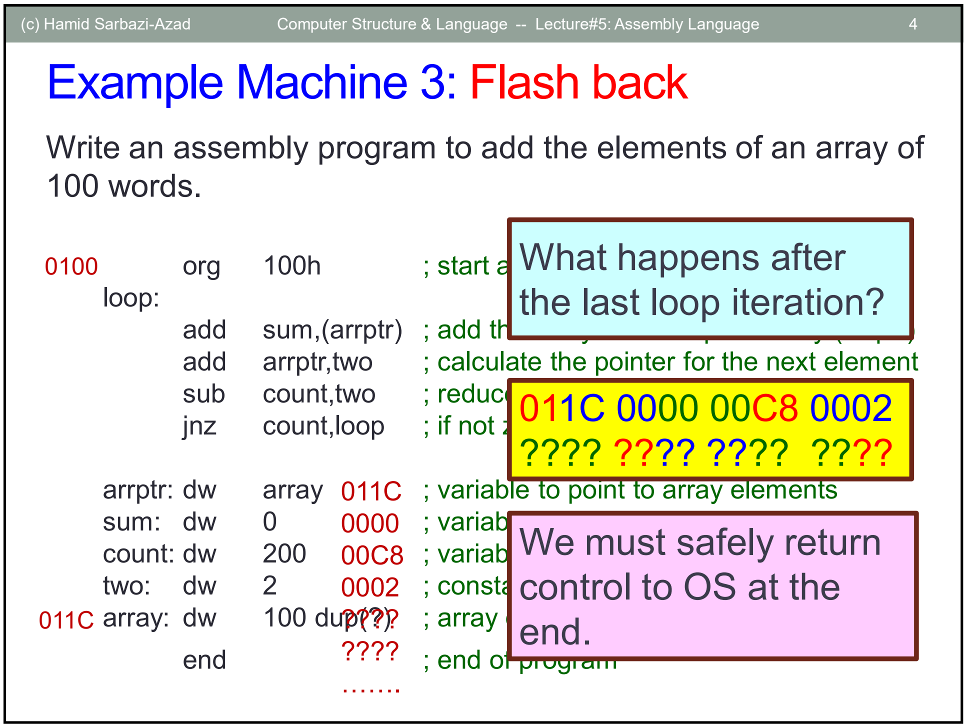


Example Machine 3: Flash back

Instructions include:

Opcode	addr1	addr2
8 bits	16 bits	16 bits

Opcode	Mnemonic		Operation
-------------	-----------------------------		----------------------------
00000000	mov	addr1,addr2	$M_{addr1} \leftarrow (M_{addr2})$;
00000001	mov	addr1,(addr2)	$M_{addr1} \leftarrow (M\_(M_{addr2}))$;
00000010	mov	(addr1),addr2	$M\_(M_{addr1}) \leftarrow (M_{addr2})$;
00000011	add	addr1,addr2	$M_{addr1} \leftarrow (M_{addr1}) + (M_{addr2})$;
00000100	sub	addr1,addr2	$M_{addr1} \leftarrow (M_{addr1}) - (M_{addr2})$;
00000101	add	addr1,(addr2)	$M_{addr1} \leftarrow (M_{addr1}) + (M\_(M_{addr2}))$;
00000110	sub	addr1,(addr2)	$M_{addr1} \leftarrow (M_{addr1}) - (M\_(M_{addr2}))$;
10000000	jnz	addr1,addr2	if $(M_{addr1}) \neq 0$ then PC $\leftarrow$ addr2;
10000001	jz	addr1,addr2	if $(M_{addr1}) = 0$ then PC $\leftarrow$ addr2;
10000010	jneg	addr1,addr2	if $(M_{addr1}) < 0$ then PC $\leftarrow$ addr2;
10000011	jpos	addr1,addr2	if $(M_{addr1}) > 0$ then PC $\leftarrow$ addr2;



Example Machine 3: Flash back

Write an assembly program to add the elements of an array of 100 words.

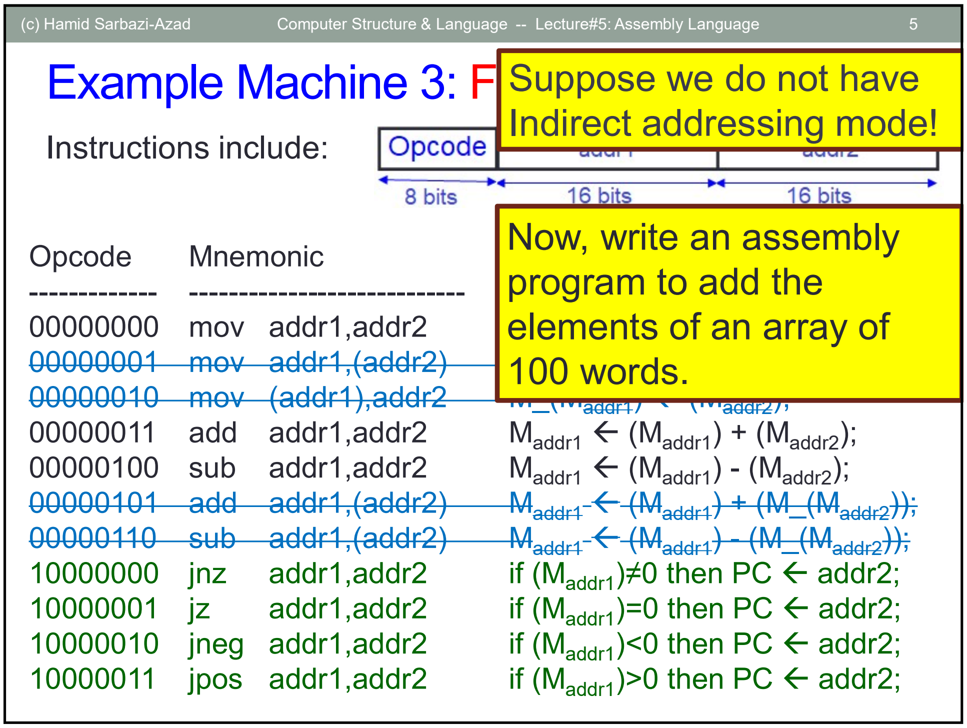
```
0100         org     100h            ; start a
     loop:
             add     sum,(arrptr)    ; add th
             add     arrptr,two      ; calculate the pointer for the next element
             sub     count,two       ; reduc
             jnz     count,loop      ; if not z

     arrptr: dw      array  011C     ; variable to point to array elements
     sum:    dw      0      0000     ; variab
     count:  dw      200    00C8     ; variab
     two:    dw      2      0002     ; consta
011C array:  dw      100 dup(?)  ???? ; array
             end            ????     ; end of program
                            .......
```

What happens after the last loop iteration?

011C 0000 00C8 0002
???? ???? ???? ????

We must safely return control to OS at the end.



Example Machine 3: F

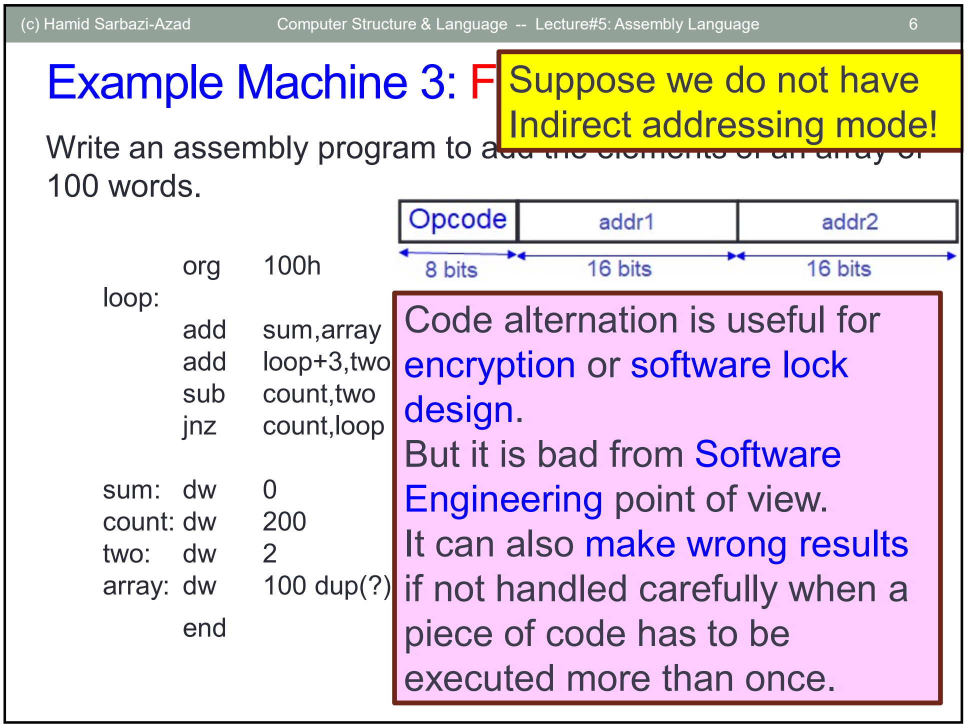
Suppose we do not have Indirect addressing mode!

Instructions include:

Opcode
addr1
addr2
8 bits
16 bits
16 bits

Now, write an assembly program to add the elements of an array of 100 words.

Opcode	Mnemonic		
00000000	mov	addr1,addr2	
~~00000001~~	~~mov~~	~~addr1,(addr2)~~	
~~00000010~~	~~mov~~	~~(addr1),addr2~~	~~$M\_(M_{addr1}) \leftarrow (M_{addr2})$;~~
00000011	add	addr1,addr2	$M_{addr1} \leftarrow (M_{addr1}) + (M_{addr2})$;
00000100	sub	addr1,addr2	$M_{addr1} \leftarrow (M_{addr1}) - (M_{addr2})$;
~~00000101~~	~~add~~	~~addr1,(addr2)~~	~~$M_{addr1} \leftarrow (M_{addr1}) + (M\_(M_{addr2}))$;~~
~~00000110~~	~~sub~~	~~addr1,(addr2)~~	~~$M_{addr1} \leftarrow (M_{addr1}) - (M\_(M_{addr2}))$;~~
10000000	jnz	addr1,addr2	if $(M_{addr1}) \neq 0$ then PC ← addr2;
10000001	jz	addr1,addr2	if $(M_{addr1}) = 0$ then PC ← addr2;
10000010	jneg	addr1,addr2	if $(M_{addr1}) < 0$ then PC ← addr2;
10000011	jpos	addr1,addr2	if $(M_{addr1}) > 0$ then PC ← addr2;



Example Machine 3: F

Suppose we do not have Indirect addressing mode!

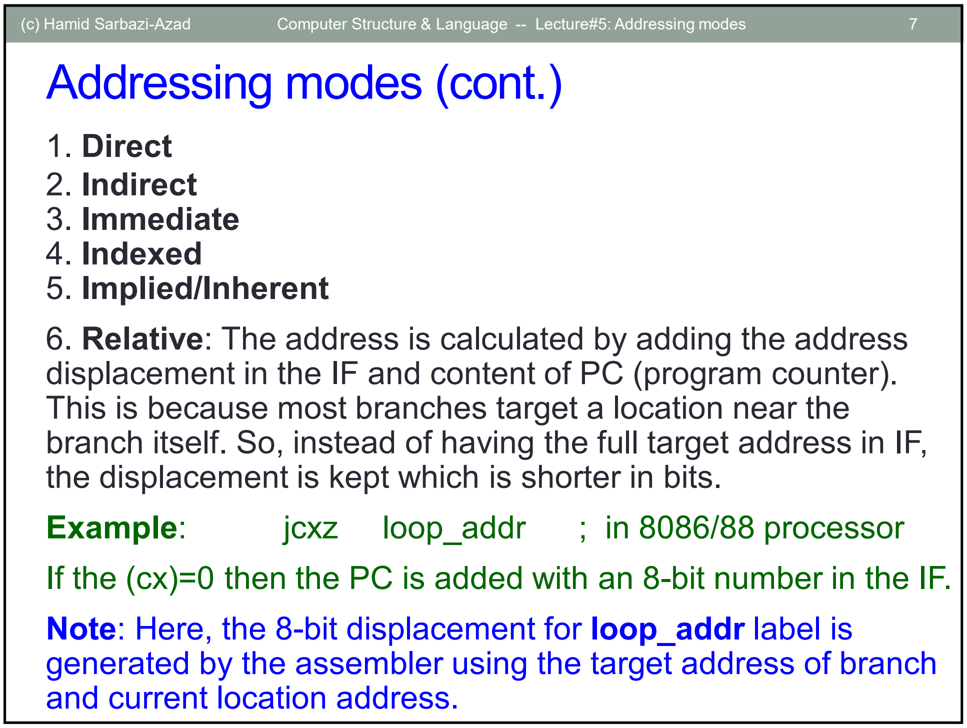
Write an assembly program to a[dd the elements of an array of]
100 words.

Opcode
addr1
addr2
8 bits
16 bits
16 bits

```
        org     100h
loop:
        add     sum,array
        add     loop+3,two
        sub     count,two
        jnz     count,loop

sum:    dw      0
count:  dw      200
two:    dw      2
array:  dw      100 dup(?)
        end
```

Code alternation is useful for encryption or software lock design.
But it is bad from Software Engineering point of view.
It can also make wrong results if not handled carefully when a piece of code has to be executed more than once.



Addressing modes (cont.)

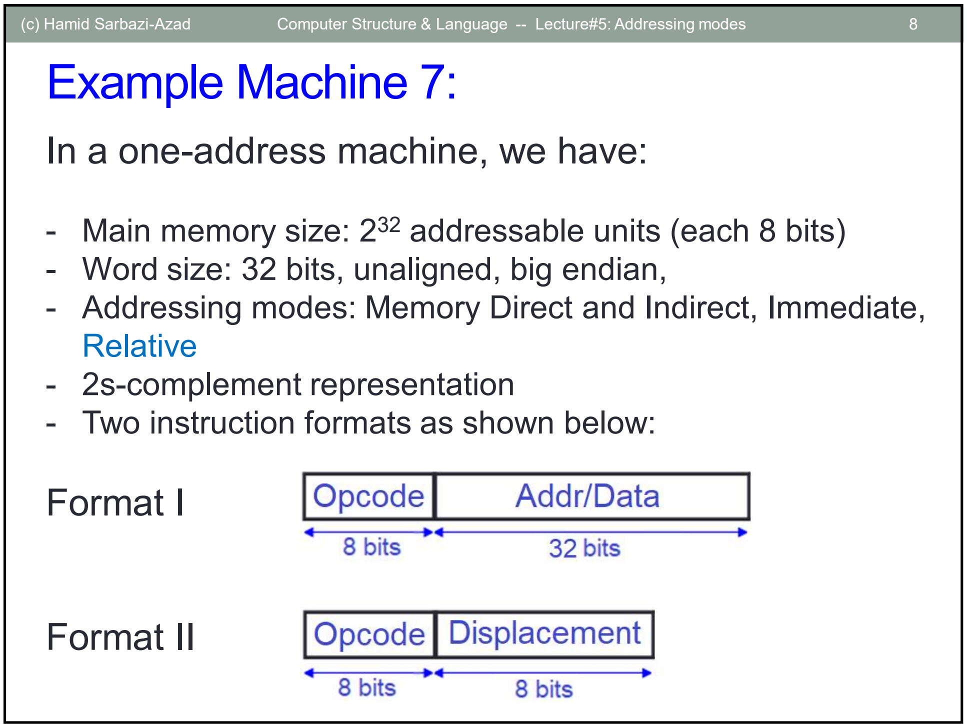
1. **Direct**
2. **Indirect**
3. **Immediate**
4. **Indexed**
5. **Implied/Inherent**

6. **Relative**: The address is calculated by adding the address displacement in the IF and content of PC (program counter). This is because most branches target a location near the branch itself. So, instead of having the full target address in IF, the displacement is kept which is shorter in bits.

Example: jcxz loop_addr ; in 8086/88 processor

If the (cx)=0 then the PC is added with an 8-bit number in the IF.

Note: Here, the 8-bit displacement for **loop_addr** label is generated by the assembler using the target address of branch and current location address.



Example Machine 7:

In a one-address machine, we have:

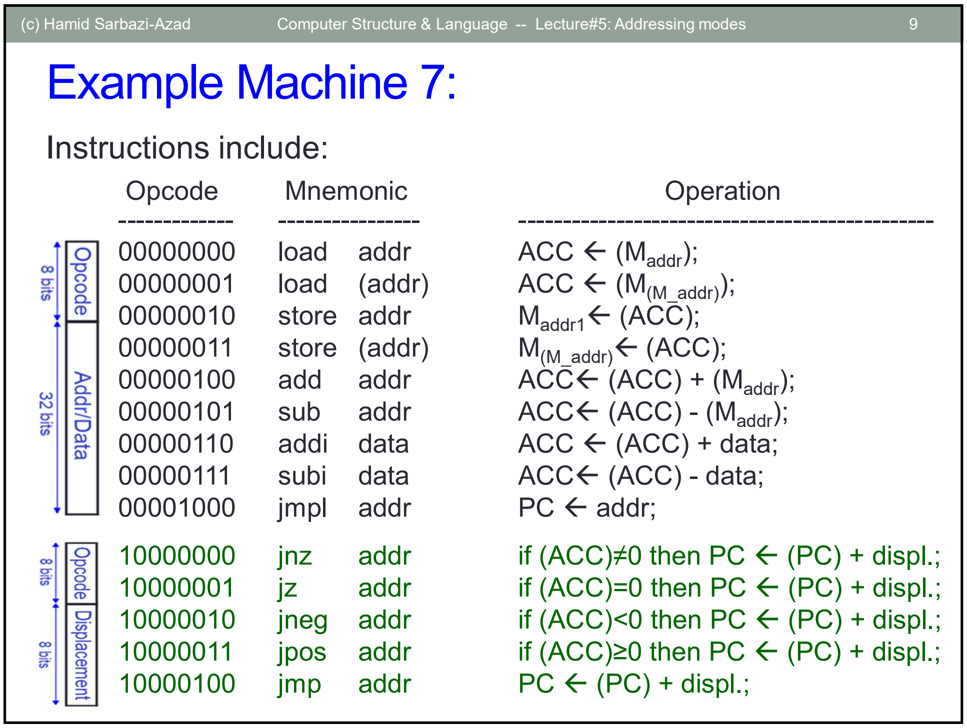
- Main memory size: 2^{32} addressable units (each 8 bits)
- Word size: 32 bits, unaligned, big endian,
- Addressing modes: Memory Direct and Indirect, Immediate, Relative
- 2s-complement representation
- Two instruction formats as shown below:

Format I

Opcode	Addr/Data
8 bits	32 bits

Format II

Opcode	Displacement
8 bits	8 bits



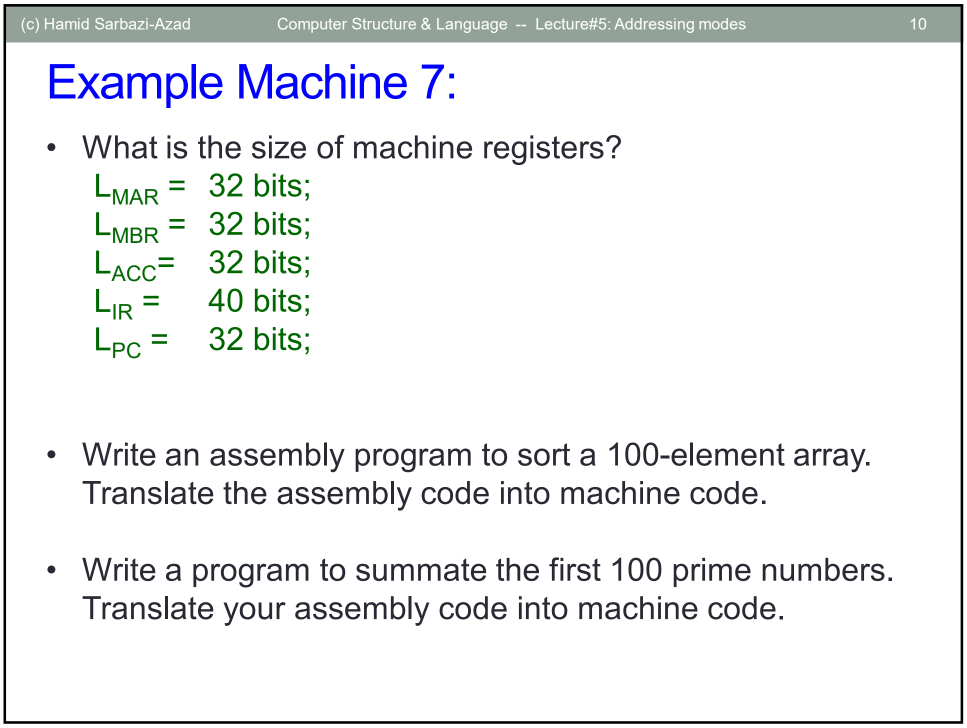
(c) Hamid Sarbazi-Azad Computer Structure & Language -- Lecture#5: Addressing modes 9

Example Machine 7:

Instructions include:

Opcode	Mnemonic		Operation
00000000	load	addr	ACC ← (M_{addr});
00000001	load	(addr)	ACC ← ($M_{(M\_addr)}$);
00000010	store	addr	M_{addr1} ← (ACC);
00000011	store	(addr)	$M_{(M\_addr)}$ ← (ACC);
00000100	add	addr	ACC← (ACC) + (M_{addr});
00000101	sub	addr	ACC← (ACC) - (M_{addr});
00000110	addi	data	ACC ← (ACC) + data;
00000111	subi	data	ACC← (ACC) - data;
00001000	jmpl	addr	PC ← addr;
10000000	jnz	addr	if (ACC)≠0 then PC ← (PC) + displ.;
10000001	jz	addr	if (ACC)=0 then PC ← (PC) + displ.;
10000010	jneg	addr	if (ACC)<0 then PC ← (PC) + displ.;
10000011	jpos	addr	if (ACC)≥0 then PC ← (PC) + displ.;
10000100	jmp	addr	PC ← (PC) + displ.;

8 bits
Opcode
32 bits
Addr/Data
8 bits
Opcode
8 bits
Displacement



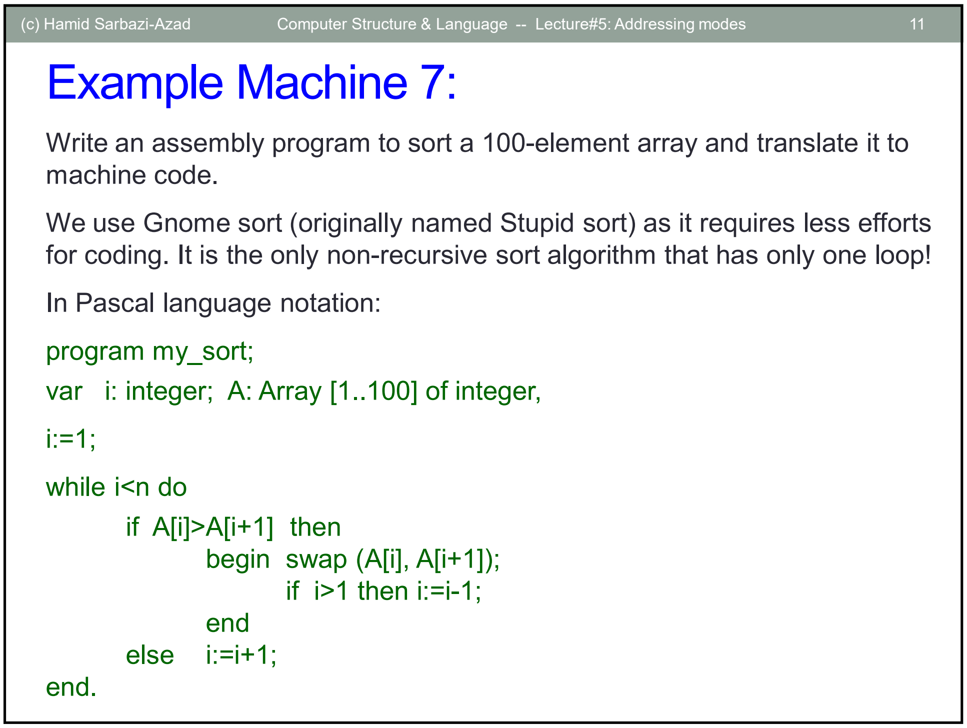
(c) Hamid Sarbazi-Azad Computer Structure & Language -- Lecture#5: Addressing modes 10

Example Machine 7:

- What is the size of machine registers?
  L_{MAR} = 32 bits;
  L_{MBR} = 32 bits;
  L_{ACC}= 32 bits;
  L_{IR} = 40 bits;
  L_{PC} = 32 bits;

- Write an assembly program to sort a 100-element array. Translate the assembly code into machine code.

- Write a program to summate the first 100 prime numbers. Translate your assembly code into machine code.



(c) Hamid Sarbazi-Azad Computer Structure & Language -- Lecture#5: Addressing modes 11

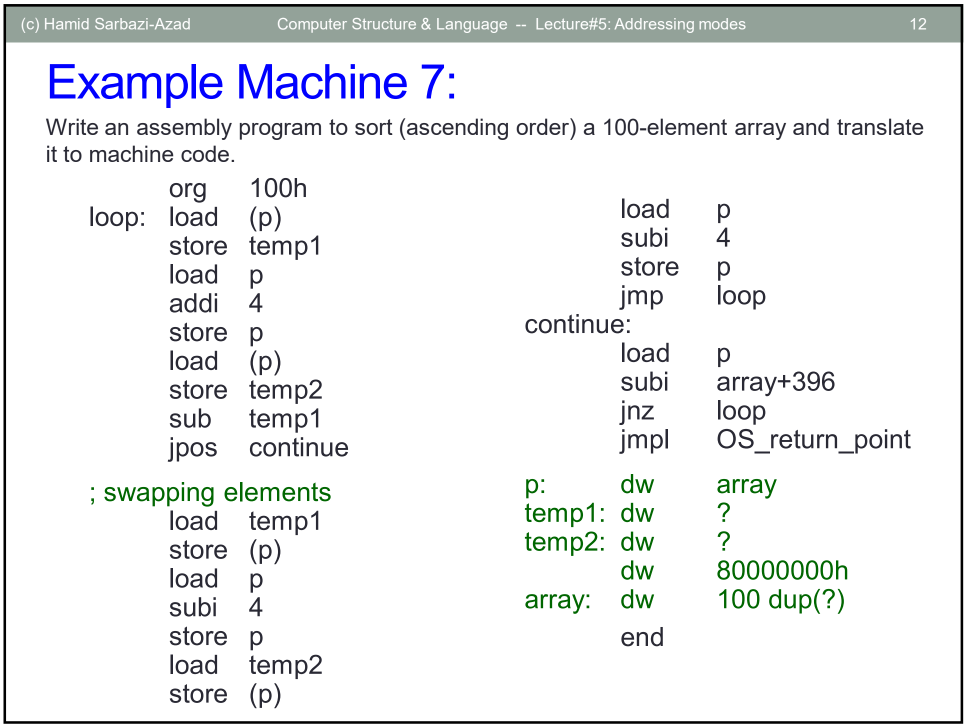
Example Machine 7:

Write an assembly program to sort a 100-element array and translate it to machine code.

We use Gnome sort (originally named Stupid sort) as it requires less efforts for coding. It is the only non-recursive sort algorithm that has only one loop!

In Pascal language notation:

```
program my_sort;
var   i: integer;  A: Array [1..100] of integer,
i:=1;
while i<n do
        if  A[i]>A[i+1]  then
                begin  swap (A[i], A[i+1]);
                       if  i>1 then i:=i-1;
                end
        else    i:=i+1;
end.
```



(c) Hamid Sarbazi-Azad Computer Structure & Language -- Lecture#5: Addressing modes 12

Example Machine 7:

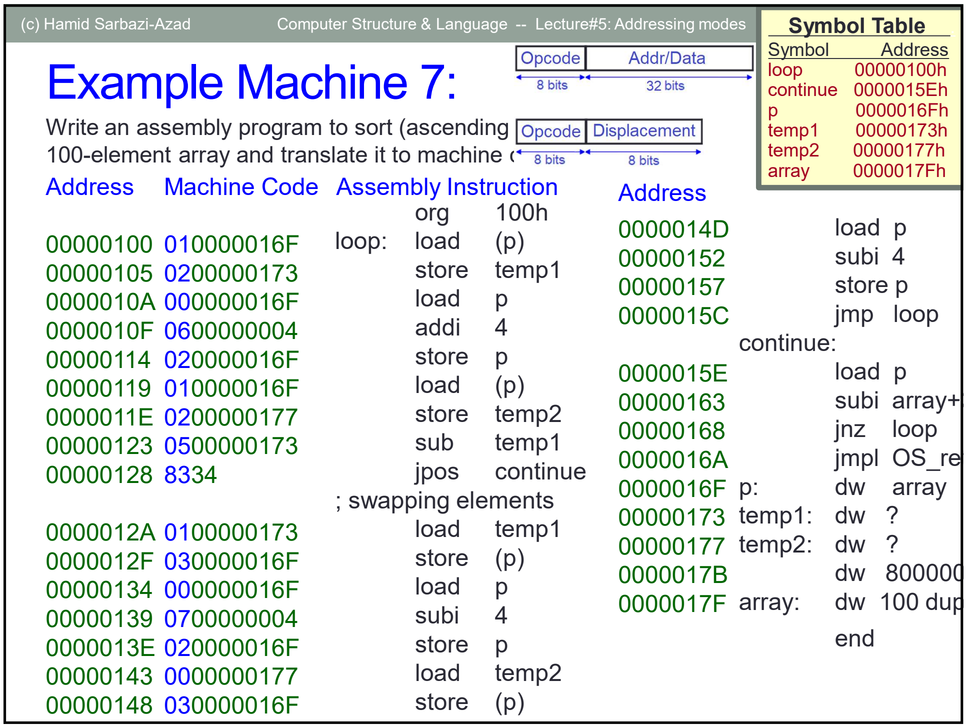
Write an assembly program to sort (ascending order) a 100-element array and translate it to machine code.

```
        org    100h
loop:   load   (p)
        store  temp1
        load   p
        addi   4
        store  p
        load   (p)
        store  temp2
        sub    temp1
        jpos   continue

; swapping elements
        load   temp1
        store  (p)
        load   p
        subi   4
        store  p
        load   temp2
        store  (p)
```

```
        load   p
        subi   4
        store  p
        jmp    loop
continue:
        load   p
        subi   array+396
        jnz    loop
        jmpl   OS_return_point

p:      dw     array
temp1:  dw     ?
temp2:  dw     ?
        dw     80000000h
array:  dw     100 dup(?)
        end
```



(c) Hamid Sarbazi-Azad Computer Structure & Language -- Lecture#5: Addressing modes

Symbol Table

Symbol	Address
loop	00000100h
continue	0000015Eh
p	0000016Fh
temp1	00000173h
temp2	00000177h
array	0000017Fh

Example Machine 7:

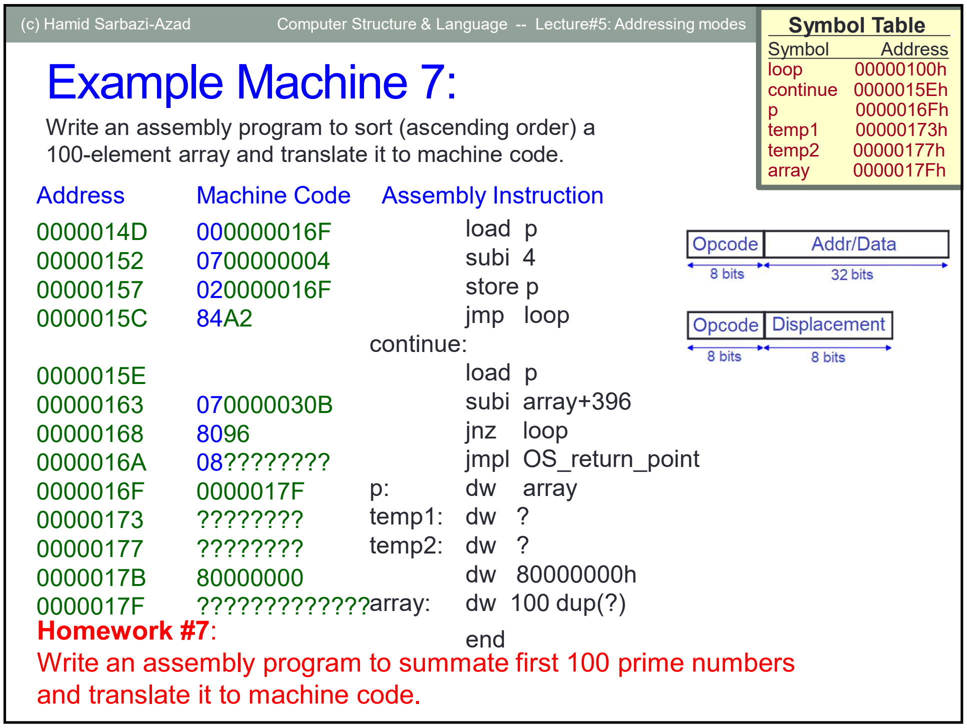
Opcode	Addr/Data
8 bits	32 bits

Opcode	Displacement
8 bits	8 bits

Write an assembly program to sort (ascending
100-element array and translate it to machine c

```
Address   Machine Code  Assembly Instruction
                              org    100h
00000100  010000016F    loop: load   (p)
00000105  0200000173          store  temp1
0000010A  000000016F          load   p
0000010F  0600000004          addi   4
00000114  020000016F          store  p
00000119  010000016F          load   (p)
0000011E  0200000177          store  temp2
00000123  0500000173          sub    temp1
00000128  8334                jpos   continue
                        ; swapping elements
0000012A  0100000173          load   temp1
0000012F  030000016F          store  (p)
00000134  000000016F          load   p
00000139  0700000004          subi   4
0000013E  020000016F          store  p
00000143  0000000177          load   temp2
00000148  030000016F          store  (p)
```

```
Address
0000014D              load  p
00000152              subi  4
00000157              store p
0000015C              jmp   loop
             continue:
0000015E              load  p
00000163              subi  array+
00000168              jnz   loop
0000016A              jmpl  OS_re
0000016F  p:          dw    array
00000173  temp1:      dw    ?
00000177  temp2:      dw    ?
0000017B              dw    800000
0000017F  array:      dw    100 dup
                      end
```



(c) Hamid Sarbazi-Azad Computer Structure & Language -- Lecture#5: Addressing modes

Symbol Table

Symbol	Address
loop	00000100h
continue	0000015Eh
p	0000016Fh
temp1	00000173h
temp2	00000177h
array	0000017Fh

Example Machine 7:

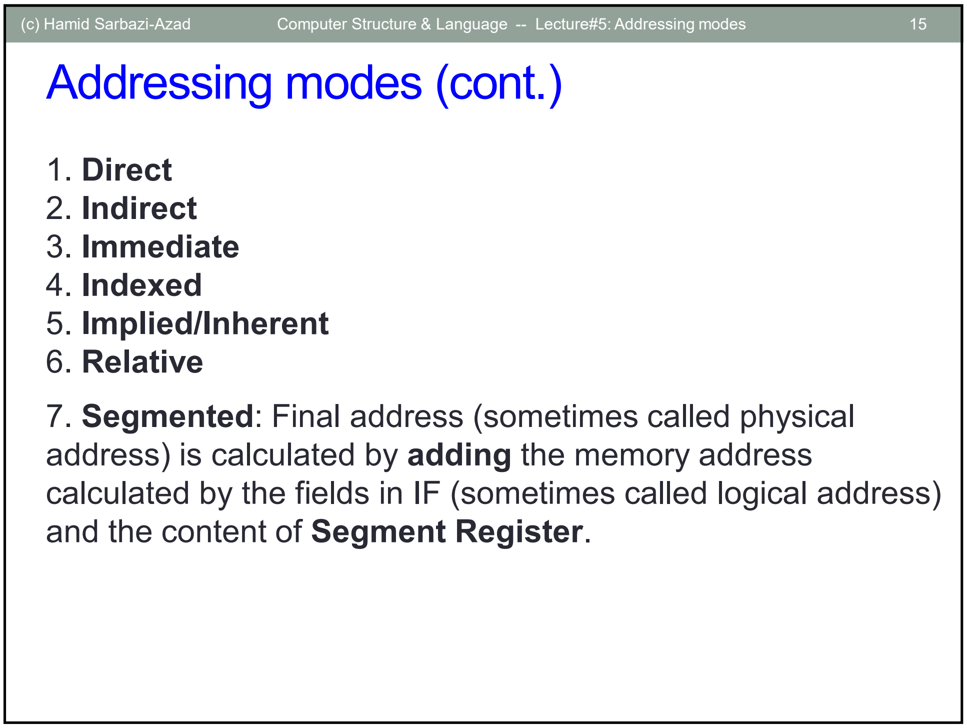
Write an assembly program to sort (ascending order) a
100-element array and translate it to machine code.

Opcode	Addr/Data
8 bits	32 bits

Opcode	Displacement
8 bits	8 bits

```
Address     Machine Code   Assembly Instruction
0000014D    000000016F              load  p
00000152    0700000004              subi  4
00000157    020000016F              store p
0000015C    84A2                    jmp   loop
                           continue:
0000015E                            load  p
00000163    070000030B              subi  array+396
00000168    8096                    jnz   loop
0000016A    08????????              jmpl  OS_return_point
0000016F    0000017F       p:       dw    array
00000173    ????????       temp1:   dw    ?
00000177    ????????       temp2:   dw    ?
0000017B    80000000                dw    80000000h
0000017F    ?????????????array:     dw    100 dup(?)
                                    end
```

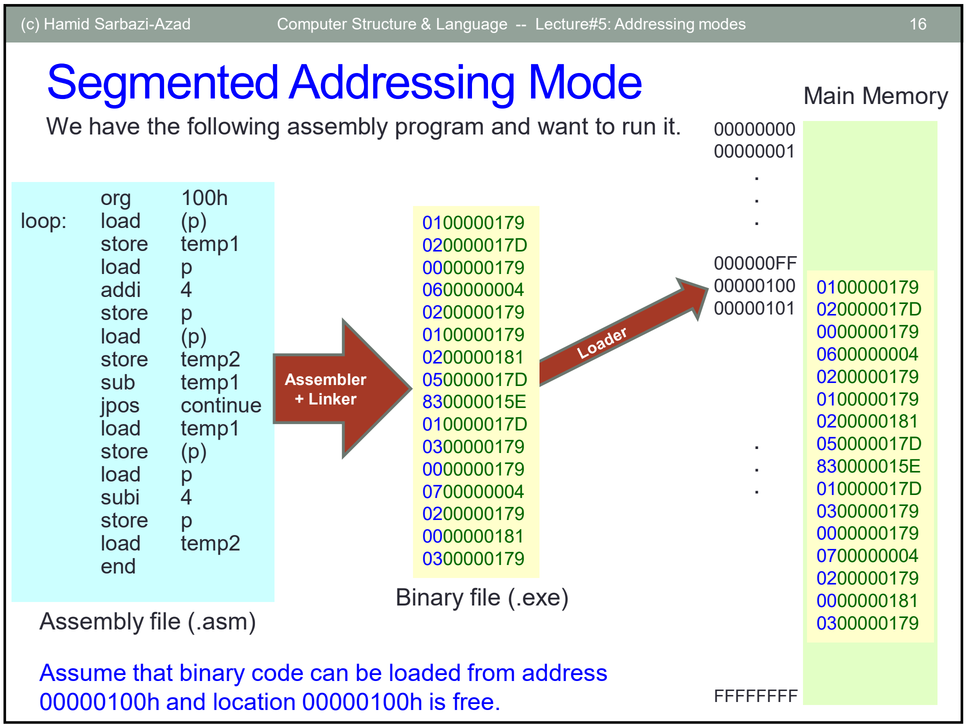
Homework #7:
Write an assembly program to summate first 100 prime numbers and translate it to machine code.



Addressing modes (cont.)

1. **Direct**
2. **Indirect**
3. **Immediate**
4. **Indexed**
5. **Implied/Inherent**
6. **Relative**

7. **Segmented**: Final address (sometimes called physical address) is calculated by **adding** the memory address calculated by the fields in IF (sometimes called logical address) and the content of **Segment Register**.



Segmented Addressing Mode

We have the following assembly program and want to run it.

```
        org     100h
loop:   load    (p)
        store   temp1
        load    p
        addi    4
        store   p
        load    (p)
        store   temp2
        sub     temp1
        jpos    continue
        load    temp1
        store   (p)
        load    p
        subi    4
        store   p
        load    temp2
        end
```

Assembly file (.asm)

```
0100000179
020000017D
0000000179
0600000004
0200000179
0100000179
0200000181
050000017D
830000015E
010000017D
0300000179
0000000179
0700000004
0200000179
0000000181
0300000179
```

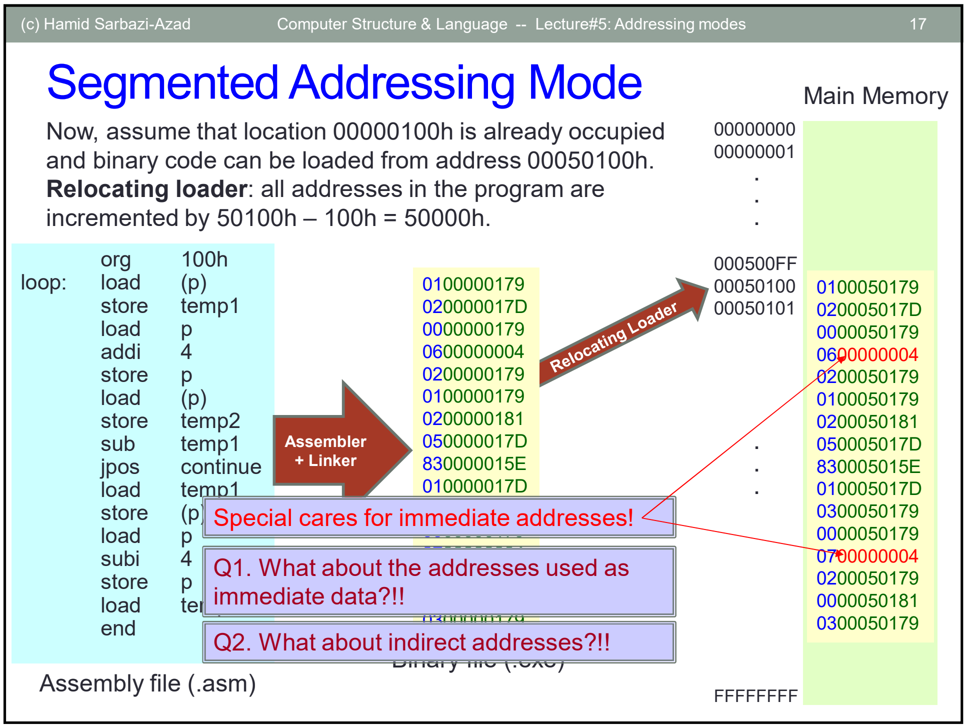
Binary file (.exe)

Main Memory

```
00000000
00000001
   .
   .
   .
000000FF
00000100   0100000179
00000101   020000017D
           0000000179
           0600000004
           0200000179
           0100000179
           0200000181
           050000017D
           830000015E
           010000017D
   .       0300000179
   .       0000000179
   .       0700000004
           0200000179
           0000000181
           0300000179

FFFFFFFF
```

Assume that binary code can be loaded from address 00000100h and location 00000100h is free.



Segmented Addressing Mode

Now, assume that location 00000100h is already occupied and binary code can be loaded from address 00050100h. **Relocating loader**: all addresses in the program are incremented by 50100h – 100h = 50000h.

```
        org     100h
loop:   load    (p)
        store   temp1
        load    p
        addi    4
        store   p
        load    (p)
        store   temp2
        sub     temp1
        jpos    continue
        load    temp1
        store   (p
        load    p
        subi    4
        store   p
        load    ter
        end
```

Assembly file (.asm)

```
0100000179
020000017D
0000000179
0600000004
0200000179
0100000179
0200000181
050000017D
830000015E
010000017D
```

Binary file (.exe)

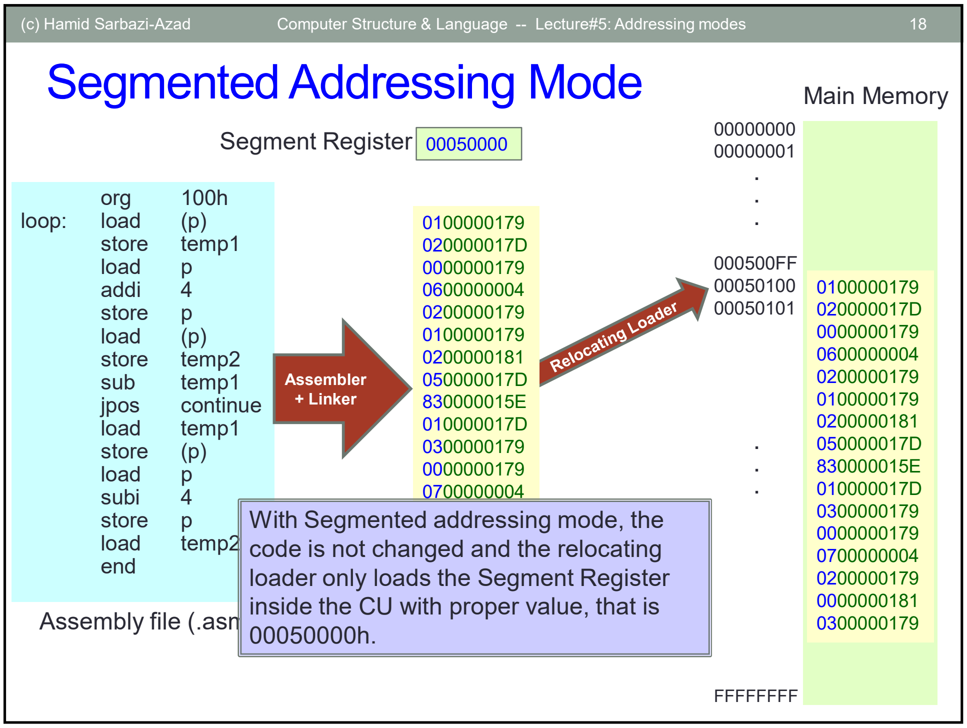
Main Memory

```
00000000
00000001
.
.
.
000500FF
00050100  0100050179
00050101  020005017D
          0000050179
          0600000004
          0200050179
          0100050179
          0200050181
          050005017D
   .      830005015E
   .      010005017D
   .      0300050179
          0000050179
          0700000004
          0200050179
          0000050181
          0300050179
FFFFFFFF
```

Special cares for immediate addresses!

Q1. What about the addresses used as immediate data?!!

Q2. What about indirect addresses?!!



Segmented Addressing Mode

Segment Register 00050000

```
        org     100h
loop:   load    (p)
        store   temp1
        load    p
        addi    4
        store   p
        load    (p)
        store   temp2
        sub     temp1
        jpos    continue
        load    temp1
        store   (p)
        load    p
        subi    4
        store   p
        load    temp2
        end
```

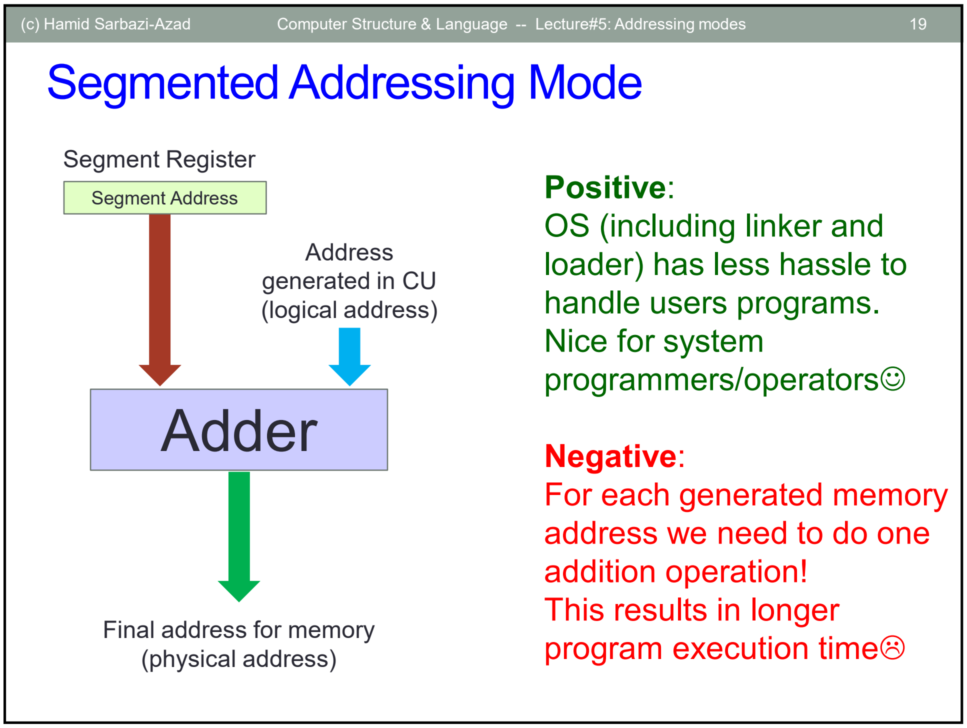
Assembly file (.asm

```
0100000179
020000017D
0000000179
0600000004
0200000179
0100000179
0200000181
050000017D
830000015E
010000017D
0300000179
0000000179
0700000004
```

Main Memory

```
00000000
00000001
.
.
.
000500FF
00050100  0100000179
00050101  020000017D
          0000000179
          0600000004
          0200000179
          0100000179
          0200000181
          050000017D
   .      830000015E
   .      010000017D
   .      0300000179
          0000000179
          0700000004
          0200000179
          0000000181
          0300000179
FFFFFFFF
```

With Segmented addressing mode, the code is not changed and the relocating loader only loads the Segment Register inside the CU with proper value, that is 00050000h.



Segmented Addressing Mode

Segment Register

Segment Address

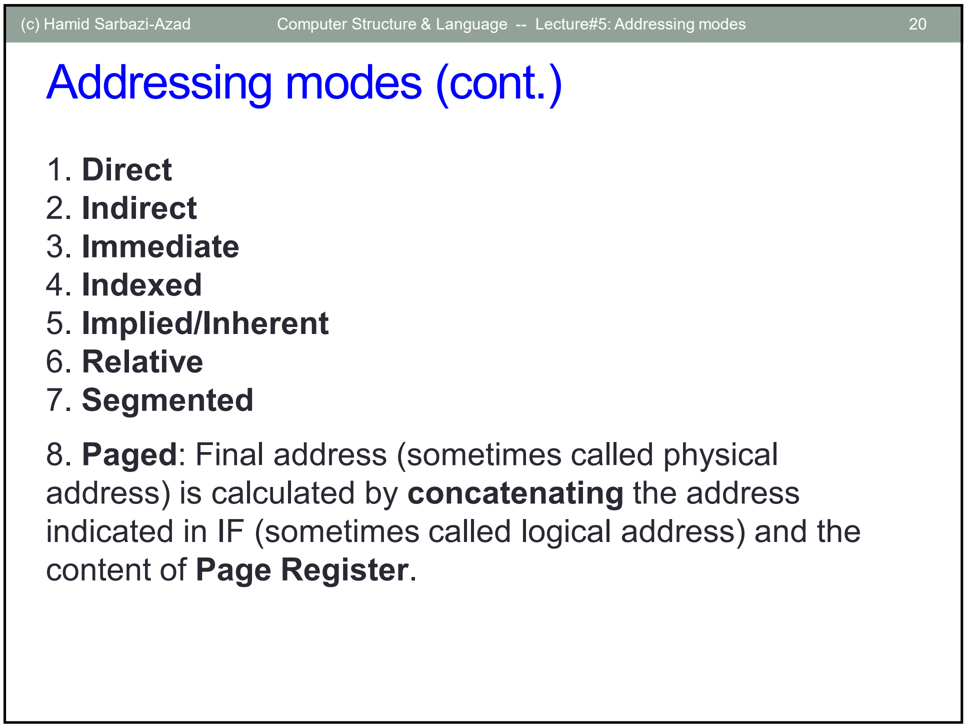
Address generated in CU (logical address)

Adder

Final address for memory (physical address)

Positive:
OS (including linker and loader) has less hassle to handle users programs.
Nice for system programmers/operators☺

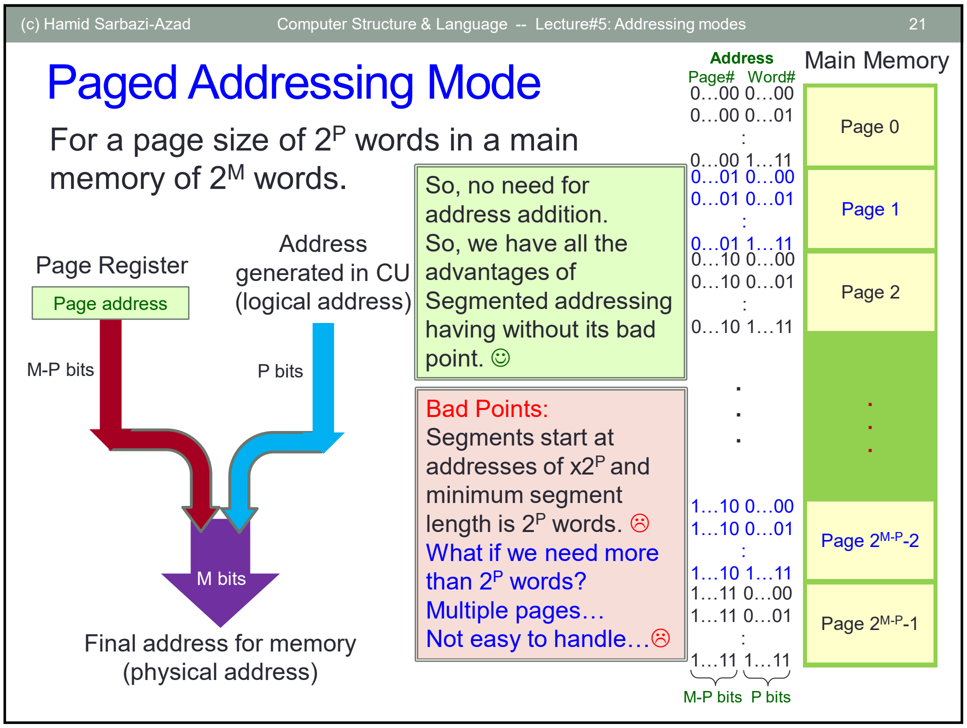
Negative:
For each generated memory address we need to do one addition operation!
This results in longer program execution time☹



Addressing modes (cont.)

1. **Direct**
2. **Indirect**
3. **Immediate**
4. **Indexed**
5. **Implied/Inherent**
6. **Relative**
7. **Segmented**

8. **Paged**: Final address (sometimes called physical address) is calculated by **concatenating** the address indicated in IF (sometimes called logical address) and the content of **Page Register**.



Paged Addressing Mode

For a page size of 2^P words in a main memory of 2^M words.

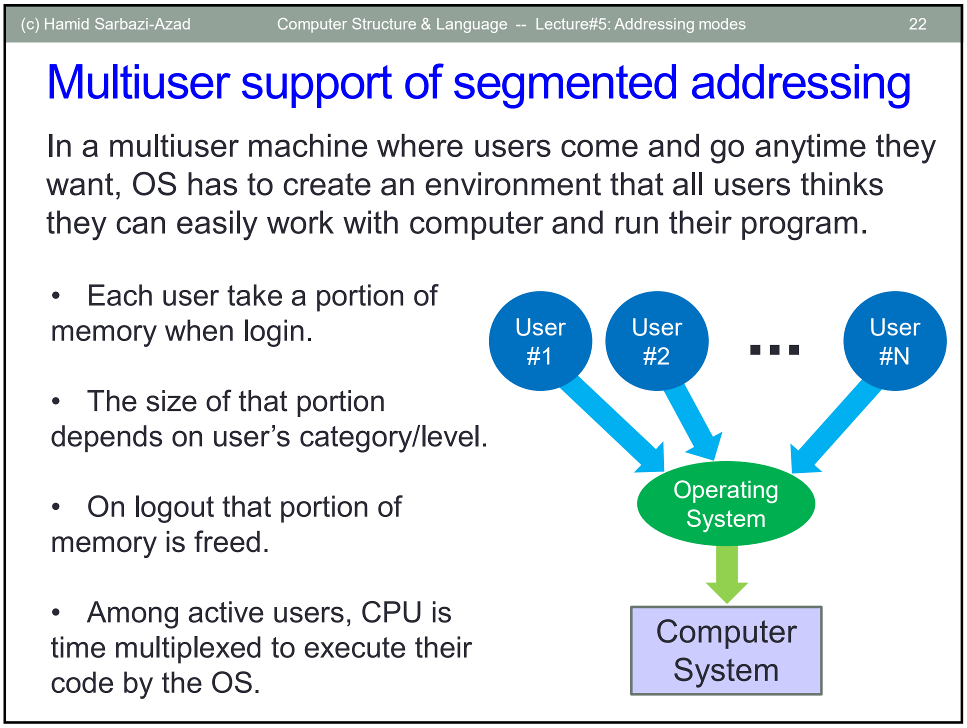
Final address for memory (physical address)

So, no need for address addition. So, we have all the advantages of Segmented addressing having without its bad point. ☺

Bad Points:
Segments start at addresses of $x2^P$ and minimum segment length is 2^P words. ☹
What if we need more than 2^P words?
Multiple pages…
Not easy to handle…☹

Address

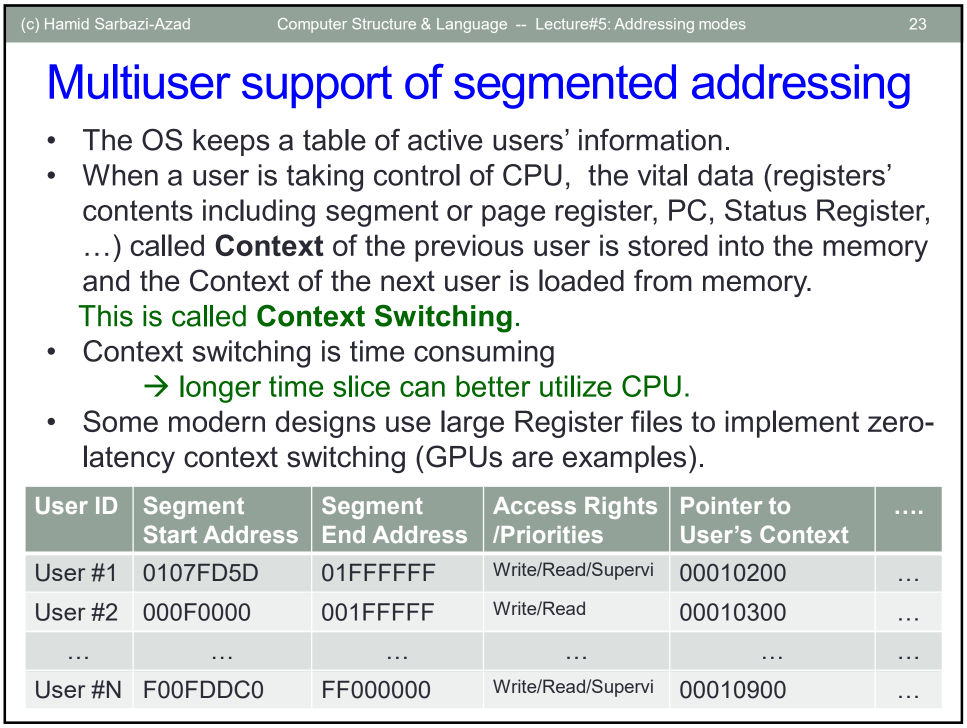
Page# Word#	Main Memory
0…00 0…00 0…00 0…01 : 0…00 1…11	Page 0
0…01 0…00 0…01 0…01 : 0…01 1…11	Page 1
0…10 0…00 0…10 0…01 : 0…10 1…11	Page 2
. . .	. . .
1…10 0…00 1…10 0…01 : 1…10 1…11	Page 2^{M-P}-2
1…11 0…00 1…11 0…01 : 1…11 1…11	Page 2^{M-P}-1
M-P bits P bits	



Multiuser support of segmented addressing

In a multiuser machine where users come and go anytime they want, OS has to create an environment that all users thinks they can easily work with computer and run their program.

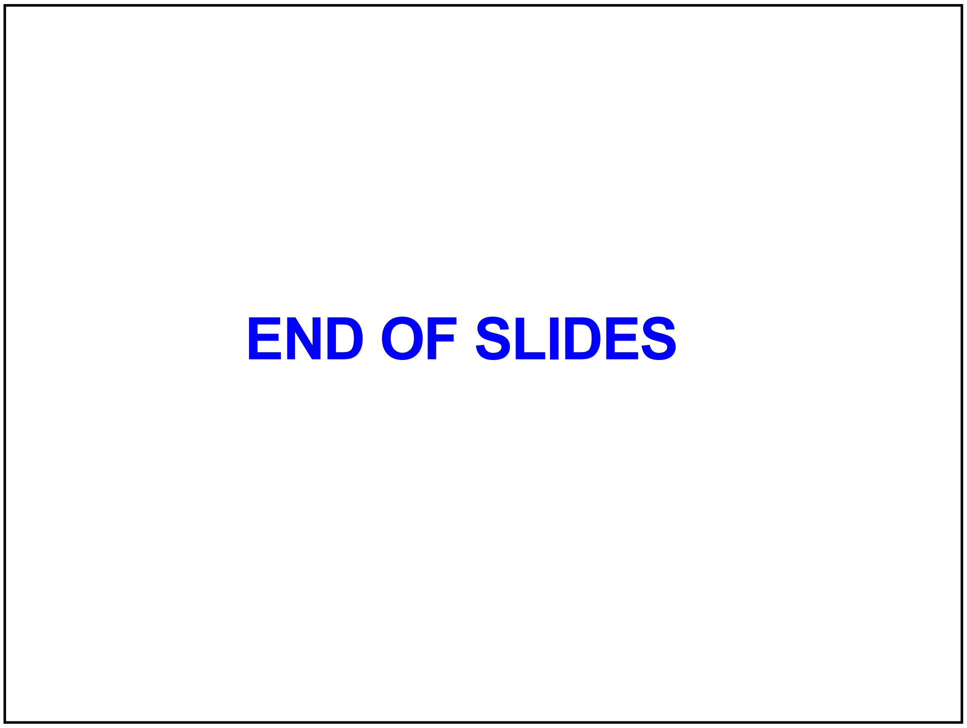
- Each user take a portion of memory when login.
- The size of that portion depends on user's category/level.
- On logout that portion of memory is freed.
- Among active users, CPU is time multiplexed to execute their code by the OS.



Multiuser support of segmented addressing

- The OS keeps a table of active users' information.
- When a user is taking control of CPU, the vital data (registers' contents including segment or page register, PC, Status Register, …) called **Context** of the previous user is stored into the memory and the Context of the next user is loaded from memory.
  This is called **Context Switching**.
- Context switching is time consuming
  → longer time slice can better utilize CPU.
- Some modern designs use large Register files to implement zero-latency context switching (GPUs are examples).

User ID	Segment Start Address	Segment End Address	Access Rights /Priorities	Pointer to User's Context	….
User #1	0107FD5D	01FFFFFF	Write/Read/Supervi	00010200	…
User #2	000F0000	001FFFFF	Write/Read	00010300	…
…	…	…	…	…	…
User #N	F00FDDC0	FF000000	Write/Read/Supervi	00010900	…



END OF SLIDES