香港中文大學
The Chinese University of Hong Kong

CSCI2510 Computer Organization

Tutorial 09:Associative mapping in MASM

Yuhong LIANG

yhliang@cse.cuhk.edu.hk

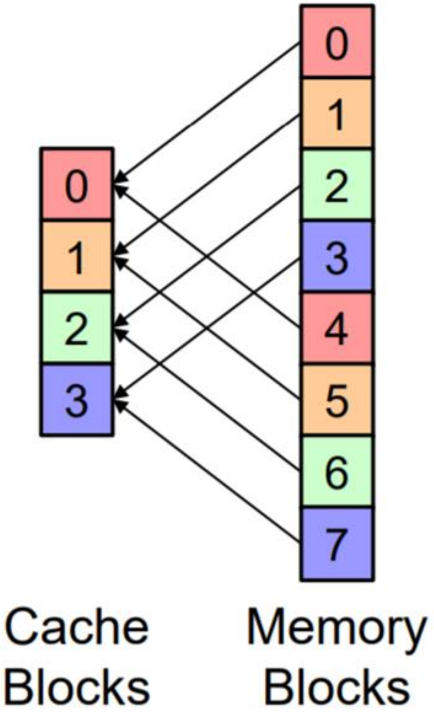
Outline

- LRU Algorithm
- First-In-First-Out Algorithm

LRU Algorithm

Direct

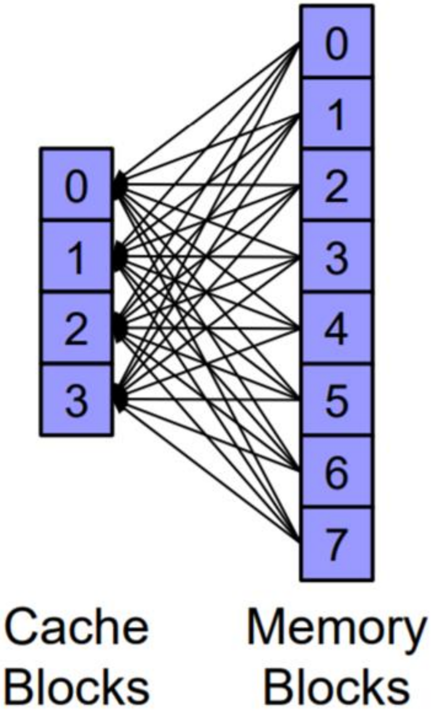
A Memory Block is **directly mapped** (%) to a Cache Block.



Associative

A Memory Block can be **mapped to any** Cache Block.

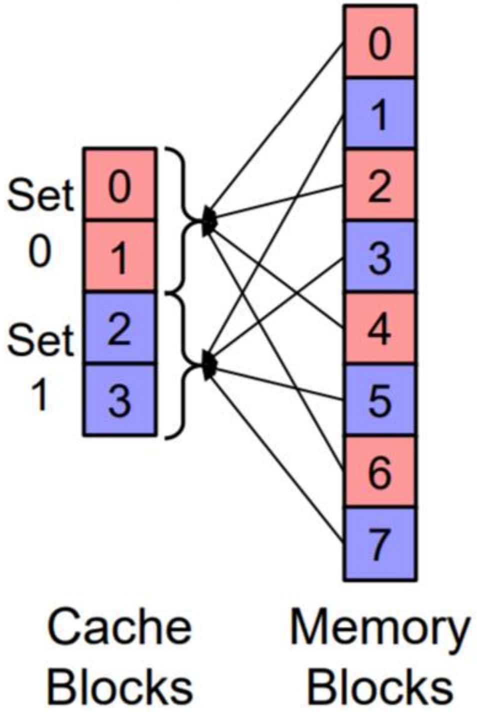
(First come first serve!)



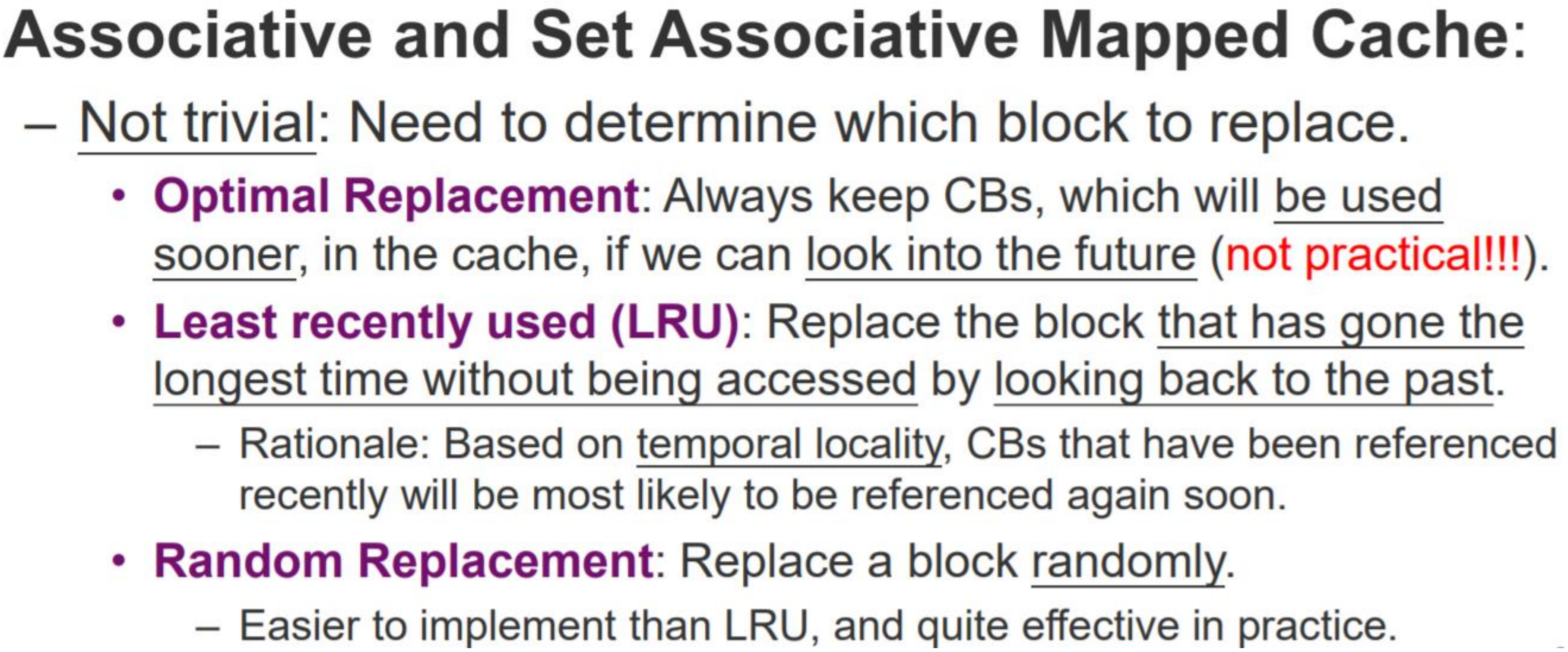
Set Associative

A Memory Block is **directly mapped** (%) to a **Cache Set**.

In a **Set**? **Associative**



LRU Algorithm



Associative and Set Associative Mapped Cache:

- Not trivial: Need to determine which block to replace.
  - **Optimal Replacement**: Always keep CBs, which will be used sooner, in the cache, if we can look into the future (not practical!!!).
  - **Least recently used (LRU)**: Replace the block that has gone the longest time without being accessed by looking back to the past.
    - Rationale: Based on temporal locality, CBs that have been referenced recently will be most likely to be referenced again soon.
  - **Random Replacement**: Replace a block randomly.
    - Easier to implement than LRU, and quite effective in practice.

LRU Algorithm

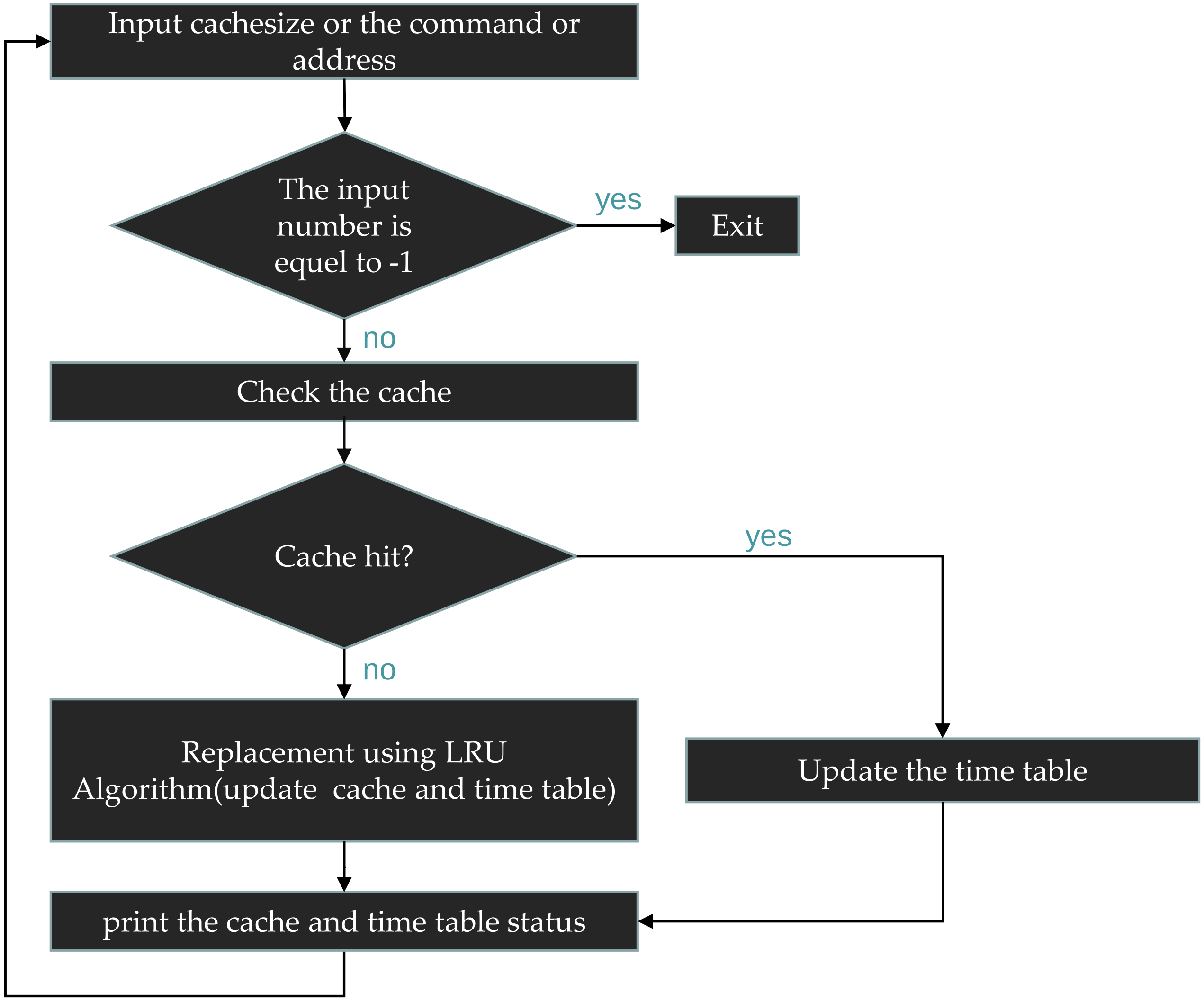
LRU Algorithm: Replace the CB that has not been used for the longest period of time (in the past).

	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
States of cache	7	7	7	7	7	3	3	3	3	3	3	3	3	3	3	3	3	7	7	7
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
			1	1	1	1	1	4	4	4	4	4	4	1	1	1	1	1	1	1
				2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2

States of count	1	1	1	1	1	6	6	6	6	10	10	12	12	12	12	12	12	18	18	18
		2	2	2	5	5	7	7	7	7	11	11	11	11	11	16	16	16	19	19
			3	3	3	3	3	8	8	8	8	8	8	14	14	14	17	17	17	20
				4	4	4	4	4	9	9	9	9	13	13	15	15	15	15	15	15

The cache block has not been used for the longest period of time!

LRU Algorithm



Input cachesize or the command or address
The input number is equel to -1
yes
Exit
no
Check the cache
Cache hit?
yes
no
Replacement using LRU Algorithm(update cache and time table)
Update the time table
print the cache and time table status

LRU Algorithm

.data:

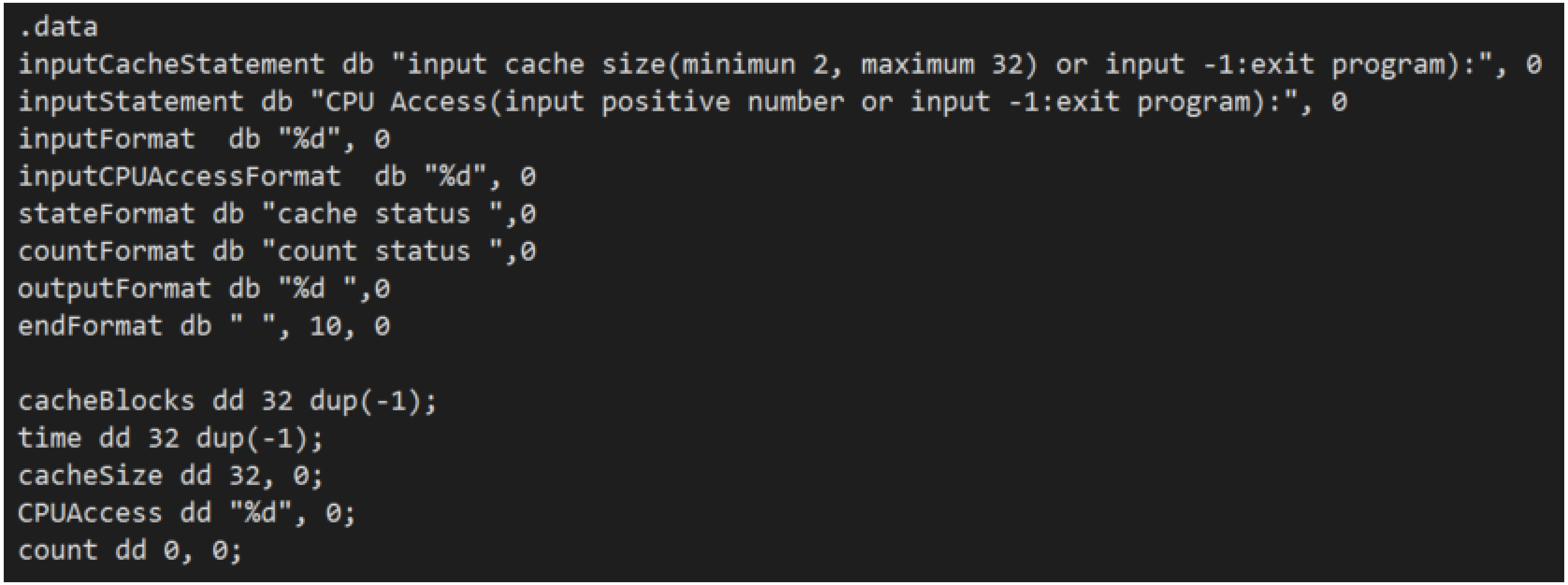
cacheBlocks dd 32 dup(-1); hold the address

time dd 32 dup(-1); hold the count

cacheSize dd 32, 0; the size of cache

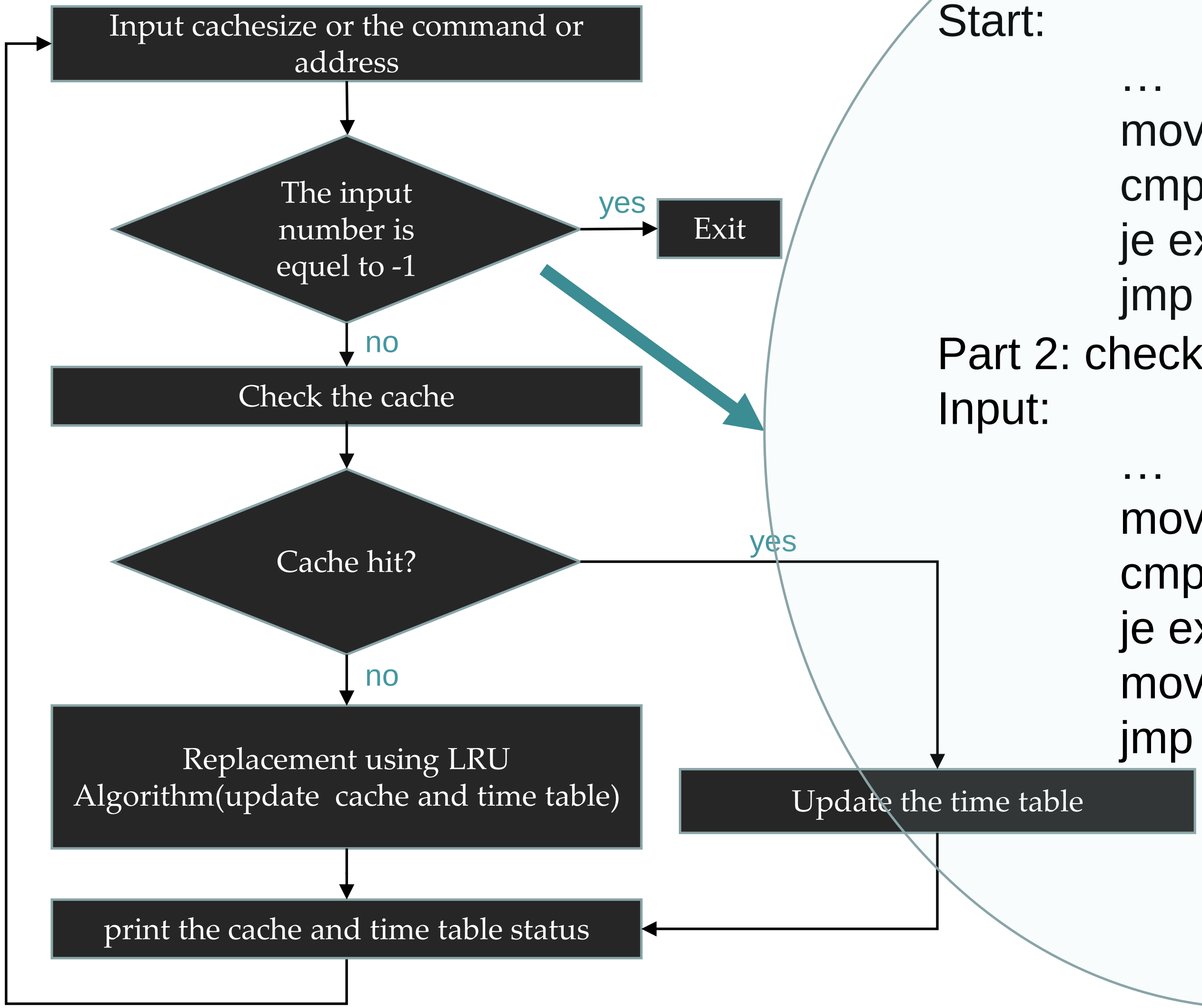
CPUAccess dd "%d", 0; the address that cpu access

count dd 0, 0; the current time



```
.data
inputCacheStatement db "input cache size(minimun 2, maximum 32) or input -1:exit program):", 0
inputStatement db "CPU Access(input positive number or input -1:exit program):", 0
inputFormat  db "%d", 0
inputCPUAccessFormat  db "%d", 0
stateFormat db "cache status ",0
countFormat db "count status ",0
outputFormat db "%d ",0
endFormat db " ", 10, 0

cacheBlocks dd 32 dup(-1);
time dd 32 dup(-1);
cacheSize dd 32, 0;
CPUAccess dd "%d", 0;
count dd 0, 0;
```

LRU Algorithm


Input cachesize or the command or address
The input number is equel to -1
yes
Exit
no
Check the cache
Cache hit?
yes
no
Replacement using LRU Algorithm(update cache and time table)
Update the time table
print the cache and time table status

Part 1: check the cachesize

Start:

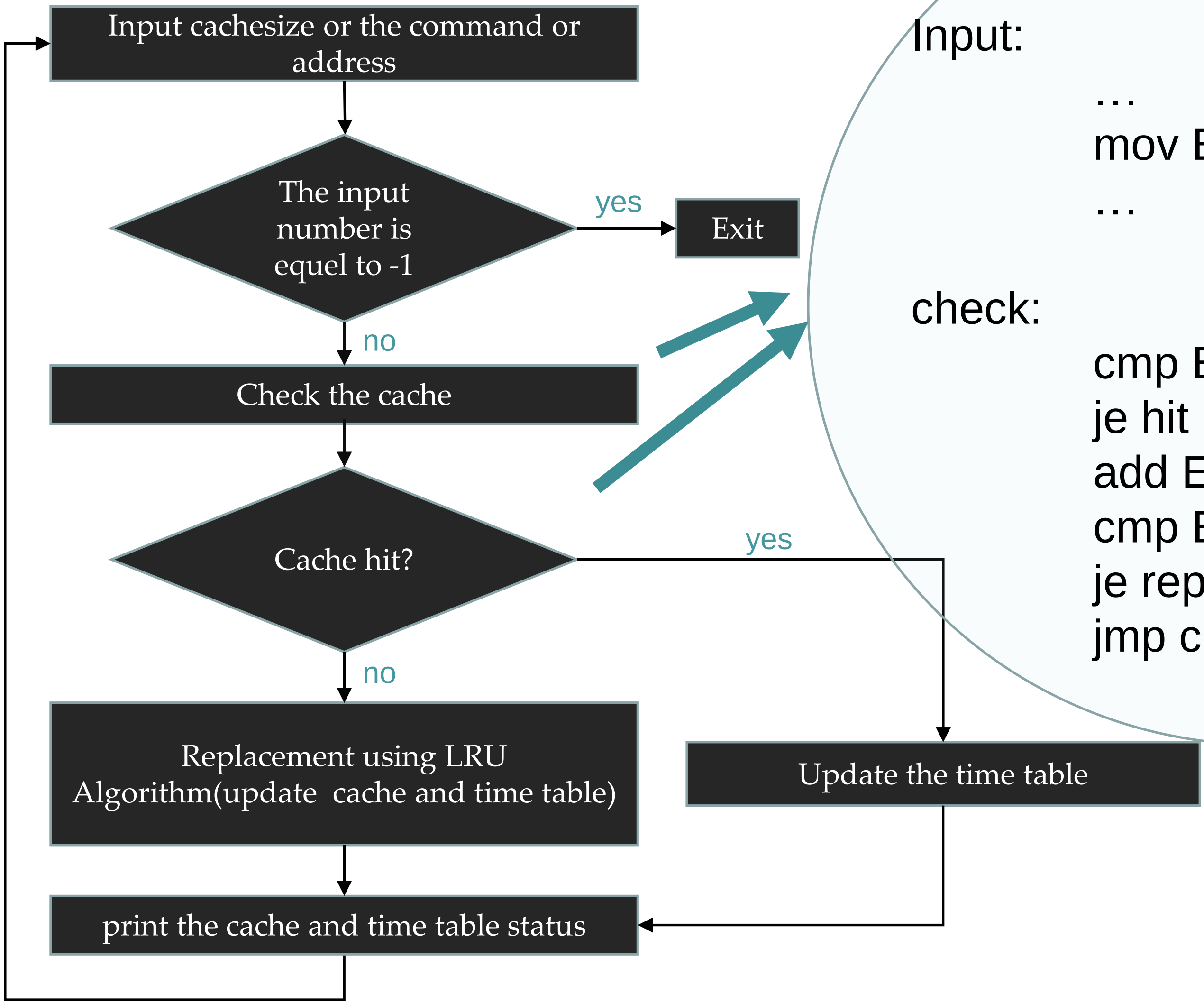
```
        …
        mov ECX, cacheSize
        cmp ECX, -1
        je exitprogram
        jmp input
```

Part 2: check the CPUAccess

Input:

```
        …
        mov ECX, CPUAccess
        cmp ECX, -1
        je exitprogram
        mov EAX, 0
        jmp check
```

LRU Algorithm

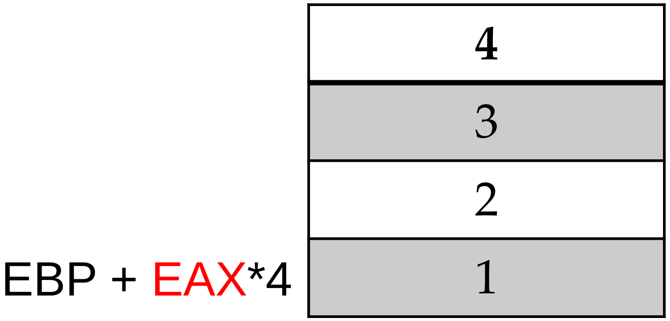

Input cachesize or the command or address
The input number is equel to -1
yes
Exit
no
Check the cache
Cache hit?
yes
no
Replacement using LRU Algorithm(update cache and time table)
Update the time table
print the cache and time table status

```
Input:
            …
            mov EAX, 0
            …

check:
            cmp ECX, [EBP + EAX*4]
            je hit
            add EAX,1
            cmp EAX, cacheSize
            je replace
            jmp check
```

LRU Algorithm

```
Input:
        …
        mov EAX, 0
        …

check:
        cmp ECX, [EBP + EAX*4]
        je hit
        add EAX,1
        cmp EAX, cacheSize
        je replace
        jmp check
```

cacheBlocks(cachesize = 4,cpu access(ECX) 2)



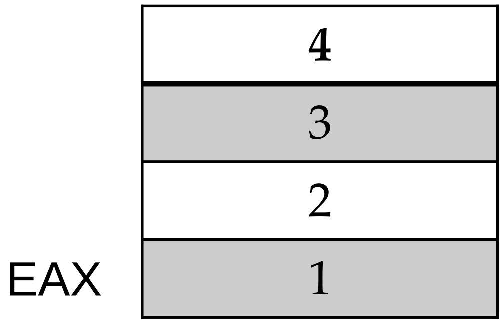
4
3
2
1

EBP + EAX*4

LRU Algorithm

```
Input:
            …
            mov EAX, 0
            …

check:
            cmp ECX, [EBP + EAX*4]
            je hit
            add EAX,1
            cmp EAX, cacheSize
            je replace
            jmp check
```

cacheBlocks(cachesize = 4,cpu access(ECX) 2)



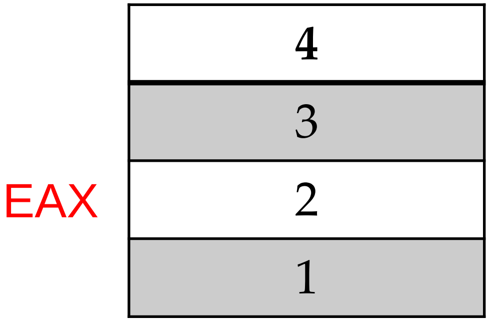
4
3
2
1

ECX != 1

LRU Algorithm

```
Input:
          …
          mov EAX, 0
          …

check:
          cmp ECX, [EBP + EAX*4]
          je hit
          add EAX,1
          cmp EAX, cacheSize
          je replace
          jmp check
```

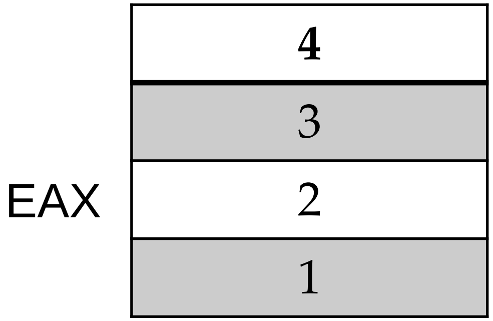
cacheBlocks(cachesize = 4,cpu access(ECX) 2)



LRU Algorithm

```
Input:
            …
            mov EAX, 0
            …

check:
            cmp ECX, [EBP + EAX*4]
            je hit
            add EAX,1
            cmp EAX, cacheSize
            je replace
            jmp check
```

cacheBlocks(cachesize = 4,cpu access(ECX) 2)



EAX

4

3

2

1

EAX!=4

LRU Algorithm

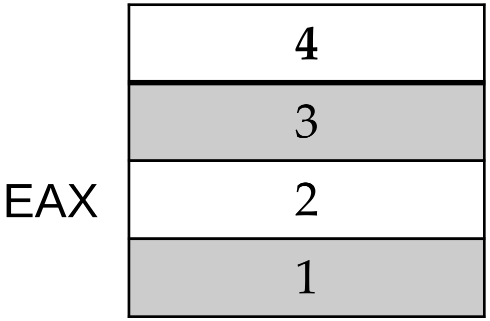
Input:

```
        …
        mov EAX, 0
        …
```

check:

```
        cmp ECX, [EBP + EAX*4]
        je hit
        add EAX,1
        cmp EAX, cacheSize
        je replace
        jmp check
```

cacheBlocks(cachesize = 4,cpu access(ECX) 2)



EAX

4

3

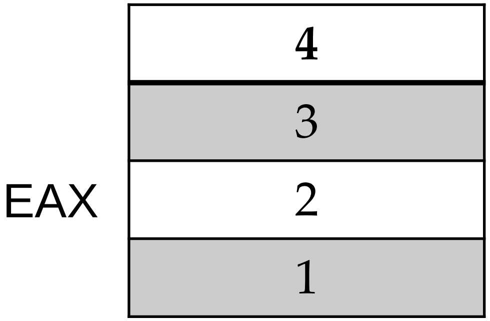
2

1

LRU Algorithm

```
Input:
        …
        mov EAX, 0
        …

check:
        cmp ECX, [EBP + EAX*4]
        je hit
        add EAX,1
        cmp EAX, cacheSize
        je replace
        jmp check
```

cacheBlocks(cachesize = 4,cpu access(ECX) 2)



ECX = 2

LRU Algorithm

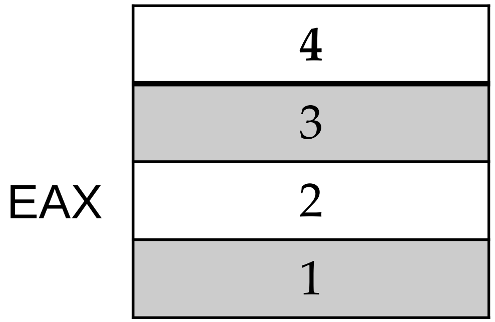
Input:

```
…
mov EAX, 0
…
```

check:

```
cmp ECX, [EBP + EAX*4]
je hit
add EAX,1
cmp EAX, cacheSize
je replace
jmp check
```

cacheBlocks(cachesize = 4,cpu access(ECX) 2)



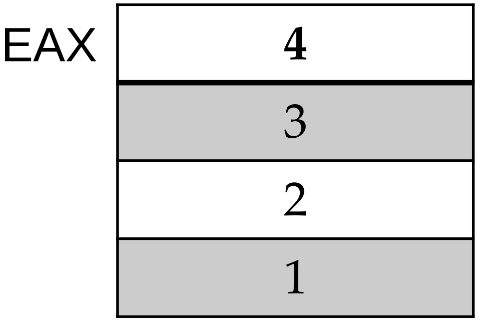
Jump to hit

LRU Algorithm

If CPU access(ECX) 5

```
Input:
            …
            mov EAX, 0
            …

check:
            cmp ECX, [EBP + EAX*4]
            je hit
            add EAX,1
            cmp EAX, cacheSize
            je replace
            jmp check
```

cacheBlocks(cachesize = 4,cpu access(ECX) 5)

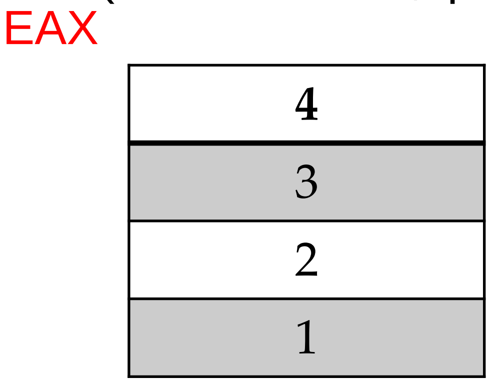


ECX != 4

LRU Algorithm

```
Input:
        …
        mov EAX, 0
        …

check:
        cmp ECX, [EBP + EAX*4]
        je hit
        add EAX,1
        cmp EAX, cacheSize
        je replace
        jmp check
```

cacheBlocks(cachesize = 4,cpu access(ECX) 5)



EAX

4
3
2
1

LRU Algorithm

Input:

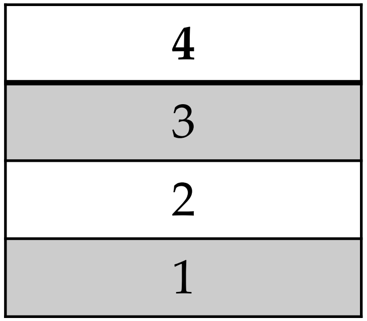
```
        …
        mov EAX, 0
        …
```

check:

```
        cmp ECX, [EBP + EAX*4]
        je hit
        add EAX,1
        cmp EAX, cacheSize
        je replace
        jmp check
```

cacheBlocks(cachesize = 4,cpu access(ECX) 5)

EAX



4

3

2

1

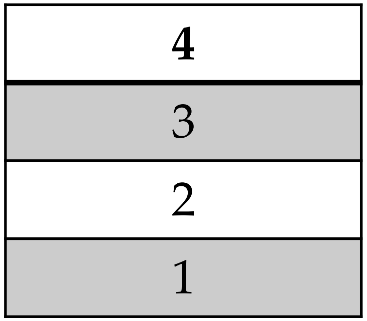
EAX = 4

LRU Algorithm

```
Input:
        …
        mov EAX, 0
        …

check:
        cmp ECX, [EBP + EAX*4]
        je hit
        add EAX,1
        cmp EAX, cacheSize
        je replace
        jmp check
```

cacheBlocks(cachesize = 4,cpu access(ECX) 5)

EAX



4

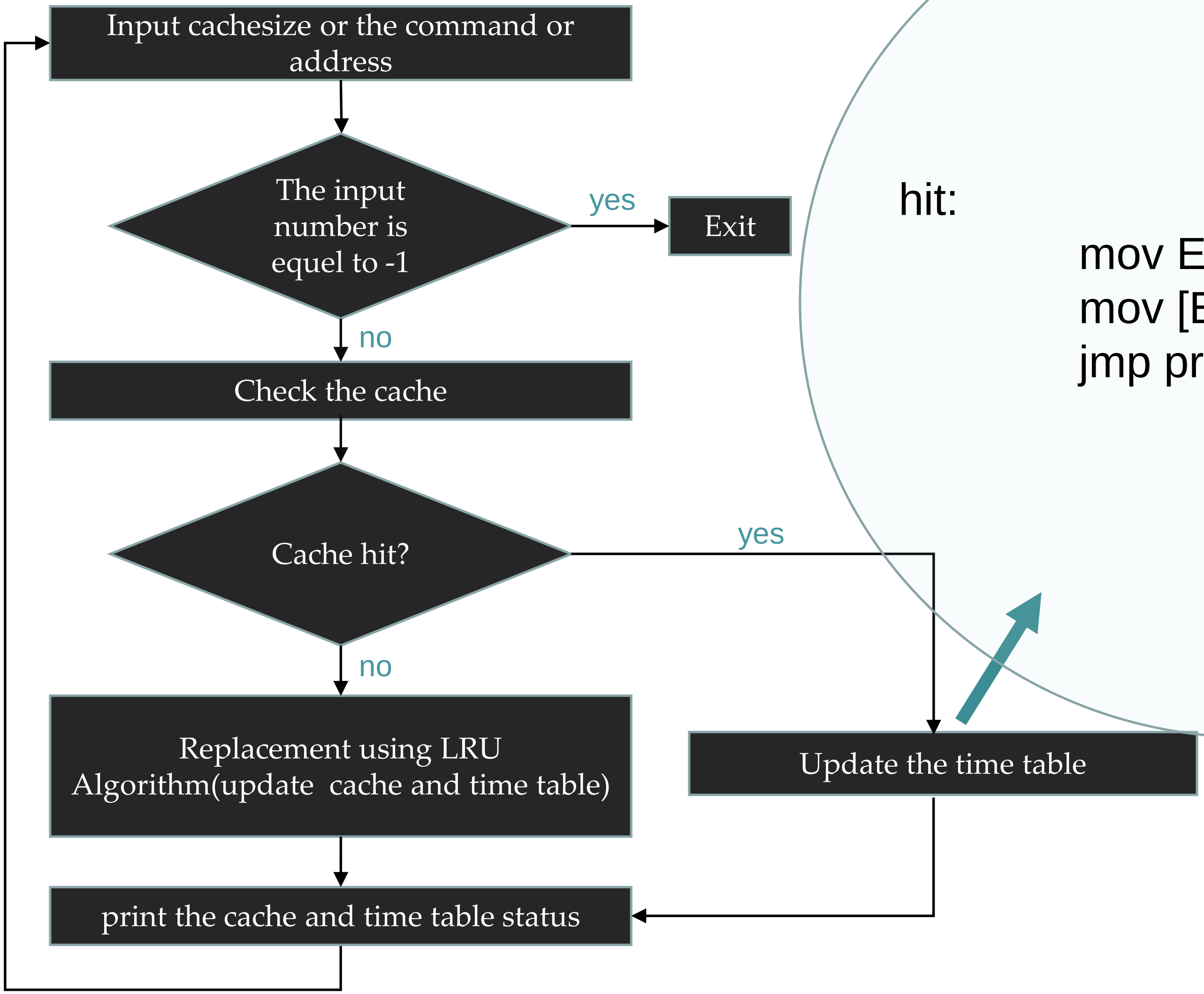
3

2

1

Jump to replace

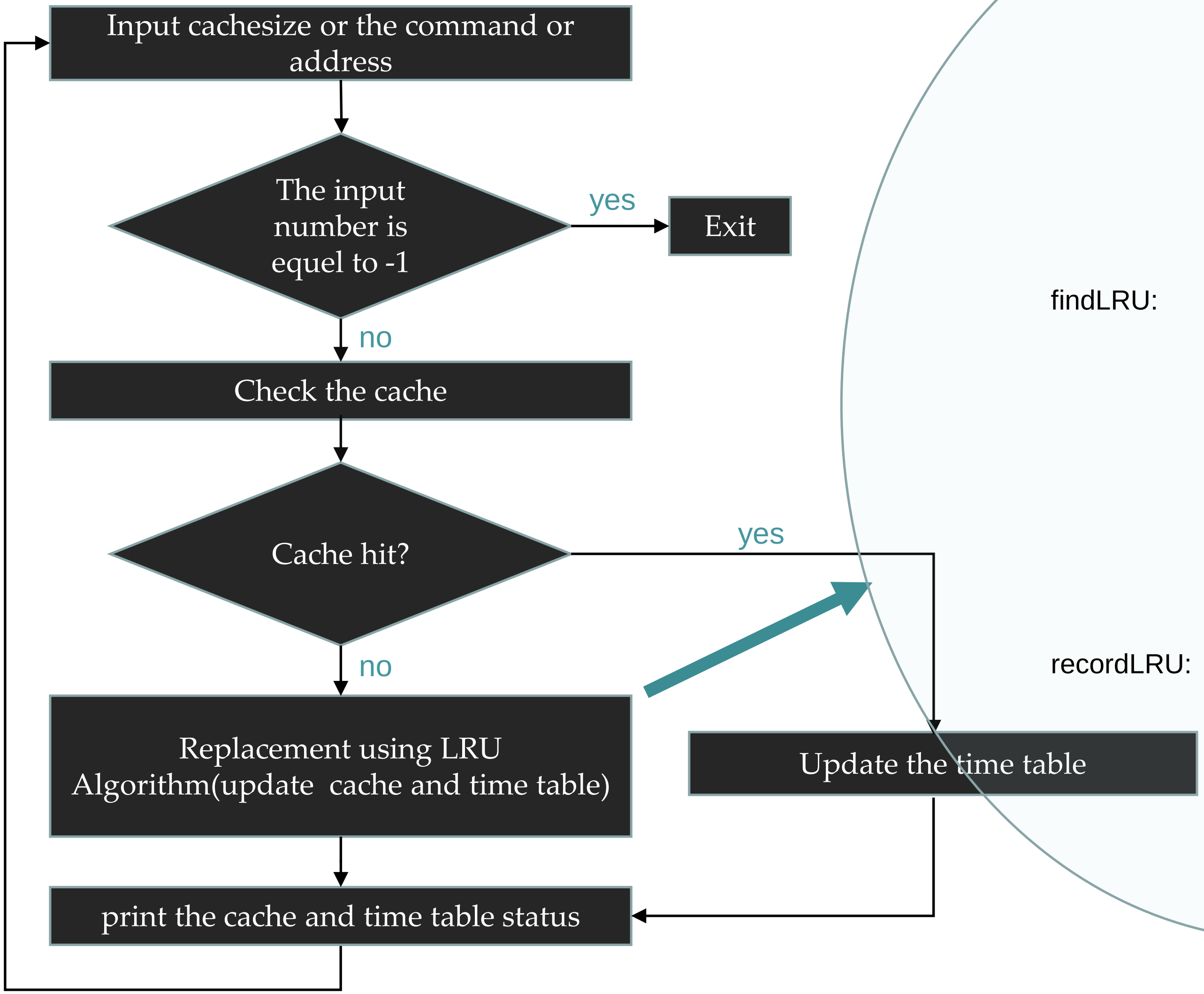
LRU Algorithm


Input cachesize or the command or address
The input number is equel to -1
yes
Exit
no
Check the cache
Cache hit?
yes
no
Replacement using LRU Algorithm(update cache and time table)
Update the time table
print the cache and time table status

hit:

```
mov EBX ,count
mov [ESI + EAX*4], EBX
jmp printcachestatus
```

LRU Algorithm



Input cachesize or the command or address

The input number is equel to -1

yes

Exit

no

Check the cache

Cache hit?

yes

no

Replacement using LRU Algorithm(update cache and time table)

Update the time table

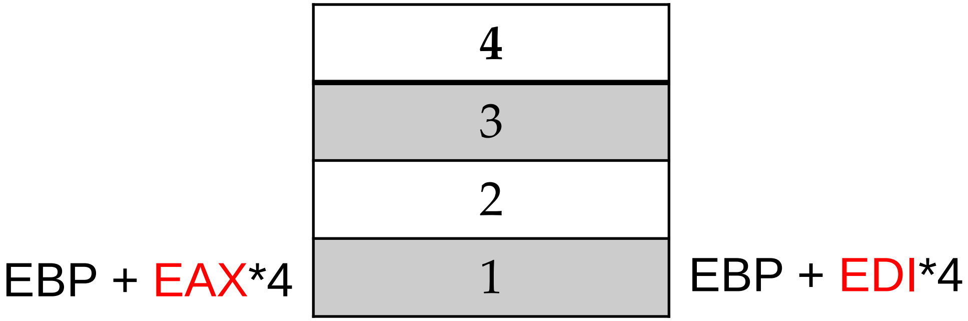
print the cache and time table status

```
replace:
                mov EAX, 0
                mov EDI, 0
                call findLRU
                mov EBX, CPUAccess
                mov [EBP + EDI*4], EBX
                mov EBX ,count
                mov [ESI + EDI*4], EBX
                jmp printcachestatus

findLRU:
                mov EBX, [ESI + EDI*4]
                add EAX, 1
                cmp EBX, [ESI + EAX*4]
                jg recordLRU
                mov EBX, cacheSize
                sub EBX, 1
                cmp EAX, EBX
                jne findLRU
                ret

recordLRU:
                mov EDI, EAX
                mov EBX, cacheSize
                dec EBX
                cmp EAX, EBX
                jne findLRU
                ret
```

LRU Algorithm

```
replace:
                  mov EAX, 0
                  mov EDI, 0
                  call findLRU
                  mov EBX, CPUAccess
                  mov [EBP + EDI*4], EBX
                  mov EBX ,count
                  mov [ESI + EDI*4], EBX
                  jmp printcachestatus

findLRU:
                  mov EBX, [ESI + EDI*4]
                  add EAX, 1
                  cmp EBX, [ESI + EAX*4]
                  jg recordLRU
                  mov EBX, cacheSize
                  sub EBX, 1
                  cmp EAX, EBX
                  jne findLRU
                  ret

recordLRU:
                  mov EDI, EAX
                  mov EBX, cacheSize
                  dec EBX
                  cmp EAX, EBX
                  jne findLRU
                  ret
```

cacheBlocks, count = 64, cpu access 5



4

3

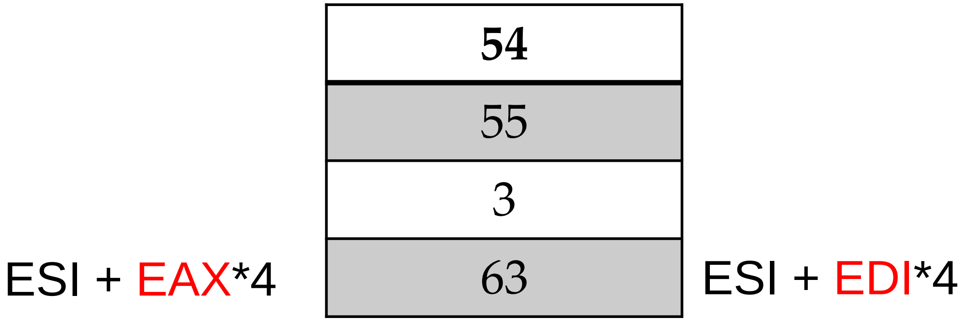
2

1

EBP + EAX*4

EBP + EDI*4

Time table, count = 64



54

55

3

63

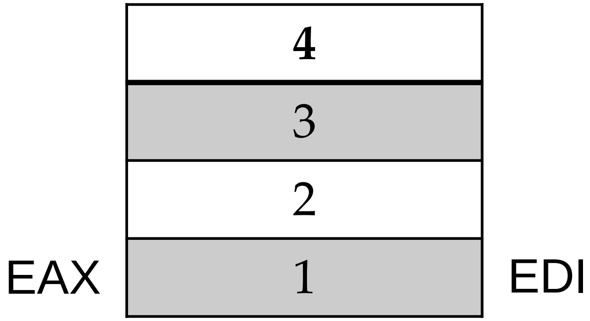
ESI + EAX*4

ESI + EDI*4

LRU Algorithm

```
replace:
        mov EAX, 0
        mov EDI, 0
        call findLRU
        mov EBX, CPUAccess
        mov [EBP + EDI*4], EBX
        mov EBX ,count
        mov [ESI + EDI*4], EBX
        jmp printcachestatus

findLRU:
        mov EBX, [ESI + EDI*4]
        add EAX, 1
        cmp EBX, [ESI + EAX*4]
        jg recordLRU
        mov EBX, cacheSize
        sub EBX, 1
        cmp EAX, EBX
        jne findLRU
        ret

recordLRU:
        mov EDI, EAX
        mov EBX, cacheSize
        dec EBX
        cmp EAX, EBX
        jne findLRU
        ret
```

cacheBlocks, count = 64, cpu access 5



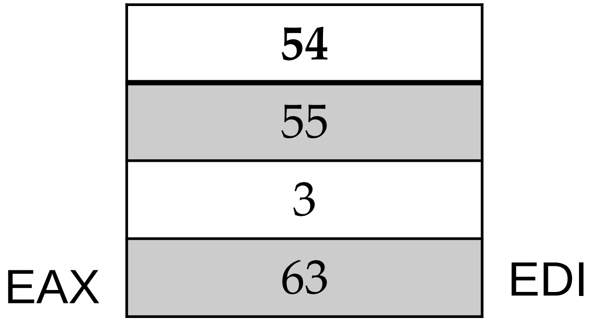
4

3

2

EAX 1 EDI

Time table, count = 64



54

55

3

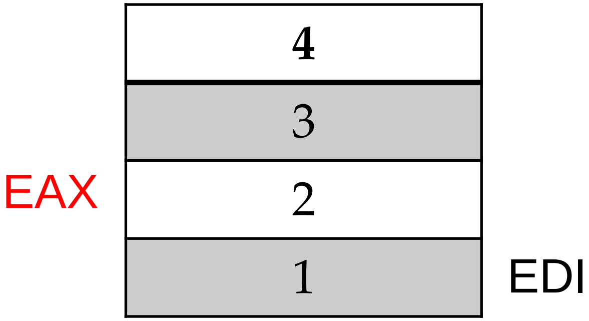
EAX 63 EDI

EBX = 63

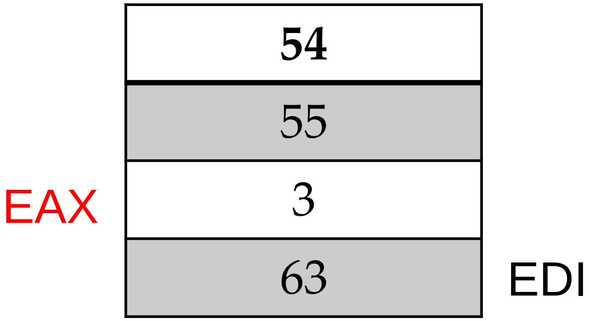
LRU Algorithm

```
replace:
                mov EAX, 0
                mov EDI, 0
                call findLRU
                mov EBX, CPUAccess
                mov [EBP + EDI*4], EBX
                mov EBX ,count
                mov [ESI + EDI*4], EBX
                jmp printcachestatus

findLRU:
                mov EBX, [ESI + EDI*4]
                add EAX, 1
                cmp EBX, [ESI + EAX*4]
                jg recordLRU
                mov EBX, cacheSize
                sub EBX, 1
                cmp EAX, EBX
                jne findLRU
                ret

recordLRU:
                mov EDI, EAX
                mov EBX, cacheSize
                dec EBX
                cmp EAX, EBX
                jne findLRU
                ret
```

cacheBlocks, count = 64, cpu access 5



Time table, count = 64

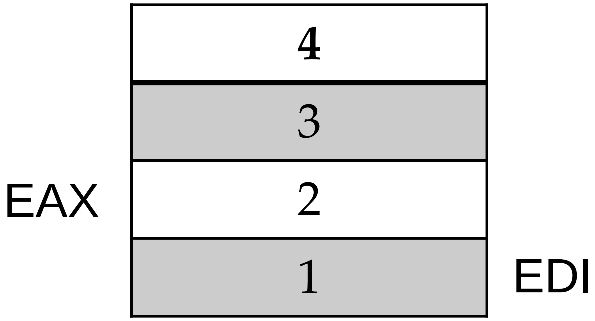


EBX = 63

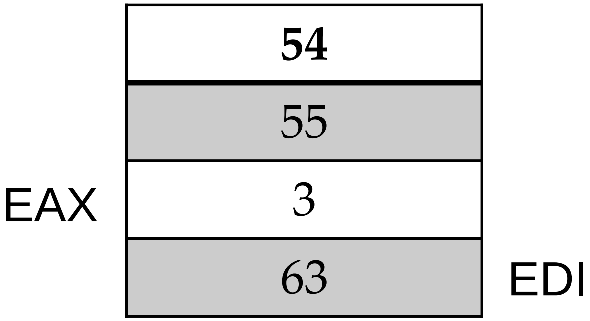
LRU Algorithm

```
replace:
        mov EAX, 0
        mov EDI, 0
        call findLRU
        mov EBX, CPUAccess
        mov [EBP + EDI*4], EBX
        mov EBX ,count
        mov [ESI + EDI*4], EBX
        jmp printcachestatus

findLRU:
        mov EBX, [ESI + EDI*4]
        add EAX, 1
        cmp EBX, [ESI + EAX*4]
        jg recordLRU
        mov EBX, cacheSize
        sub EBX, 1
        cmp EAX, EBX
        jne findLRU
        ret

recordLRU:
        mov EDI, EAX
        mov EBX, cacheSize
        dec EBX
        cmp EAX, EBX
        jne findLRU
        ret
```

cacheBlocks, count = 64, cpu access 5



Time table, count = 64



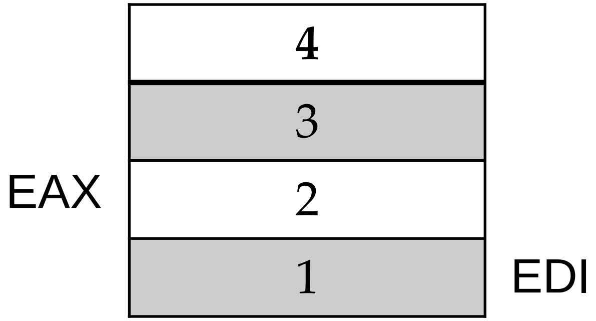
EBX = 63

EBX > 3

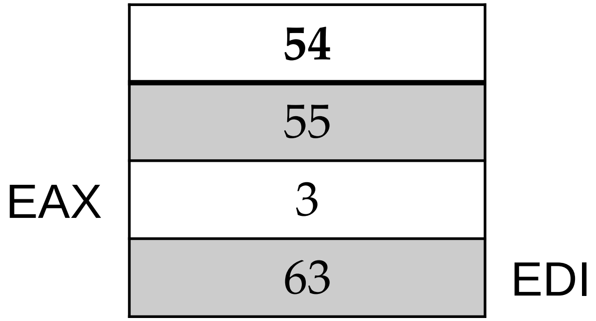
LRU Algorithm

```
replace:
            mov EAX, 0
            mov EDI, 0
            call findLRU
            mov EBX, CPUAccess
            mov [EBP + EDI*4], EBX
            mov EBX ,count
            mov [ESI + EDI*4], EBX
            jmp printcachestatus

findLRU:
            mov EBX, [ESI + EDI*4]
            add EAX, 1
            cmp EBX, [ESI + EAX*4]
            jg recordLRU
            mov EBX, cacheSize
            sub EBX, 1
            cmp EAX, EBX
            jne findLRU
            ret

recordLRU:
            mov EDI, EAX
            mov EBX, cacheSize
            dec EBX
            cmp EAX, EBX
            jne findLRU
            ret
```

cacheBlocks, count = 64, cpu access 5



Time table, count = 64

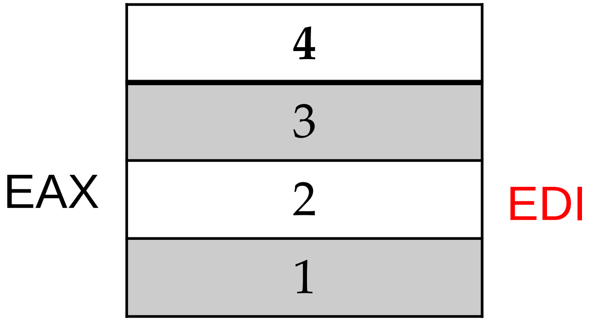


Jump to record LRU

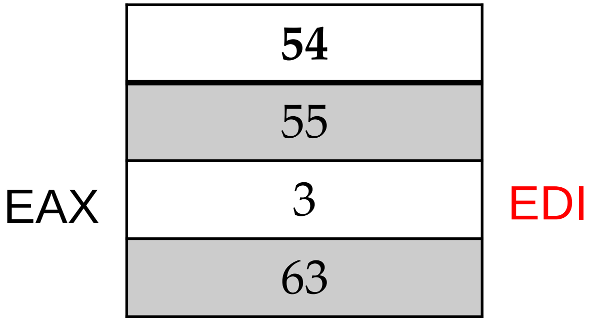
LRU Algorithm

```
replace:
                mov EAX, 0
                mov EDI, 0
                call findLRU
                mov EBX, CPUAccess
                mov [EBP + EDI*4], EBX
                mov EBX ,count
                mov [ESI + EDI*4], EBX
                jmp printcachestatus

findLRU:
                mov EBX, [ESI + EDI*4]
                add EAX, 1
                cmp EBX, [ESI + EAX*4]
                jg recordLRU
                mov EBX, cacheSize
                sub EBX, 1
                cmp EAX, EBX
                jne findLRU
                ret

recordLRU:
                mov EDI, EAX
                mov EBX, cacheSize
                dec EBX
                cmp EAX, EBX
                jne findLRU
                ret
```

cacheBlocks, count = 64, cpu access 5

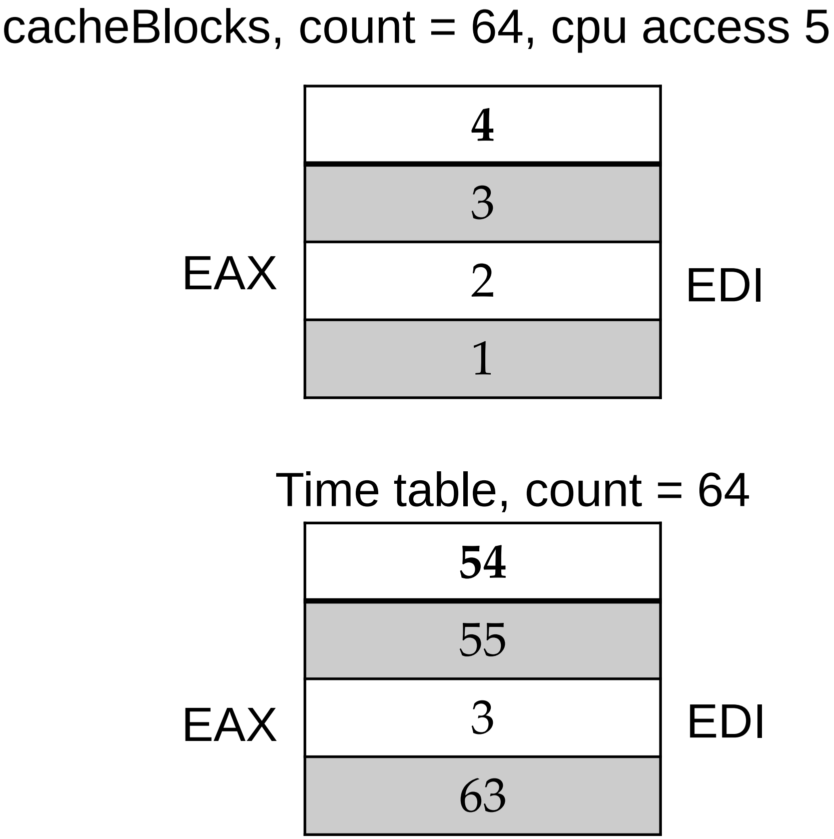


Time table, count = 64



LRU Algorithm

```
replace:
                mov EAX, 0
                mov EDI, 0
                call findLRU
                mov EBX, CPUAccess
                mov [EBP + EDI*4], EBX
                mov EBX ,count
                mov [ESI + EDI*4], EBX
                jmp printcachestatus

findLRU:
                mov EBX, [ESI + EDI*4]
                add EAX, 1
                cmp EBX, [ESI + EAX*4]
                jg recordLRU
                mov EBX, cacheSize
                sub EBX, 1
                cmp EAX, EBX
                jne findLRU
                ret

recordLRU:
                mov EDI, EAX
                mov EBX, cacheSize
                dec EBX
                cmp EAX, EBX
                jne findLRU
                ret
```



cacheBlocks, count = 64, cpu access 5

4
3
2
1

EAX

EDI

Time table, count = 64

54
55
3
63

EAX

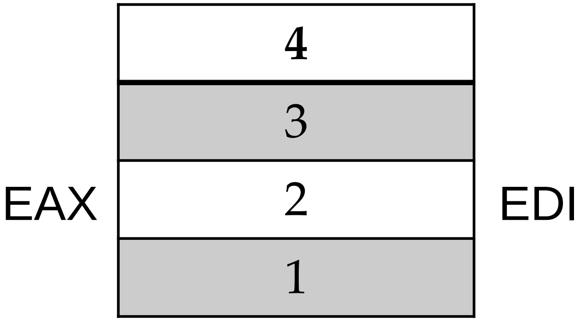
EDI

EBX = 4

LRU Algorithm

```
replace:
                mov EAX, 0
                mov EDI, 0
                call findLRU
                mov EBX, CPUAccess
                mov [EBP + EDI*4], EBX
                mov EBX ,count
                mov [ESI + EDI*4], EBX
                jmp printcachestatus

findLRU:
                mov EBX, [ESI + EDI*4]
                add EAX, 1
                cmp EBX, [ESI + EAX*4]
                jg recordLRU
                mov EBX, cacheSize
                sub EBX, 1
                cmp EAX, EBX
                jne findLRU
                ret

recordLRU:
                mov EDI, EAX
                mov EBX, cacheSize
                dec EBX
                cmp EAX, EBX
                jne findLRU
                ret
```

cacheBlocks, count = 64, cpu access 5

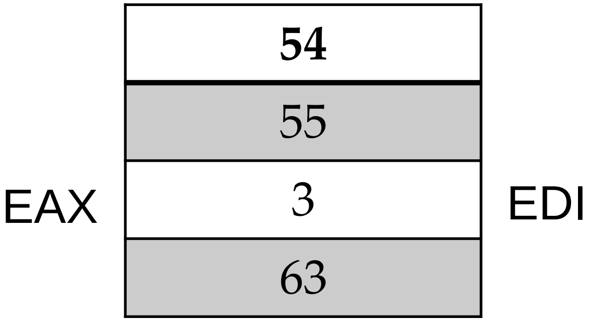


EAX

4
3
2
1

EDI

Time table, count = 64



EAX

54
55
3
63

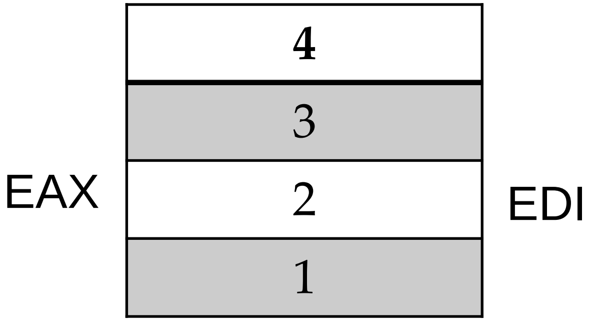
EDI

EBX = 3

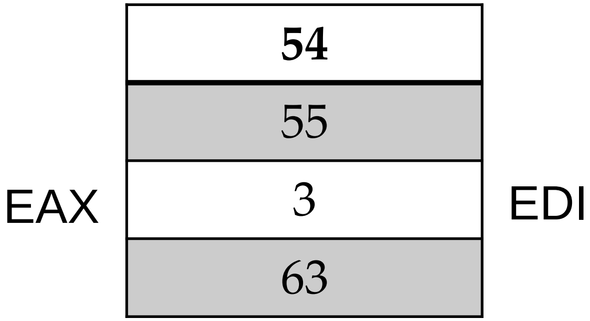
LRU Algorithm

```
replace:
        mov EAX, 0
        mov EDI, 0
        call findLRU
        mov EBX, CPUAccess
        mov [EBP + EDI*4], EBX
        mov EBX ,count
        mov [ESI + EDI*4], EBX
        jmp printcachestatus

findLRU:
        mov EBX, [ESI + EDI*4]
        add EAX, 1
        cmp EBX, [ESI + EAX*4]
        jg recordLRU
        mov EBX, cacheSize
        sub EBX, 1
        cmp EAX, EBX
        jne findLRU
        ret

recordLRU:
        mov EDI, EAX
        mov EBX, cacheSize
        dec EBX
        cmp EAX, EBX
        jne findLRU
        ret
```

cacheBlocks, count = 64, cpu access 5



Time table, count = 64



EBX = 3

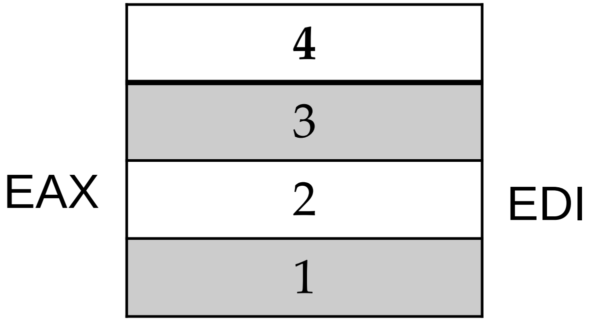
EAX = 1

EBX != 1

LRU Algorithm

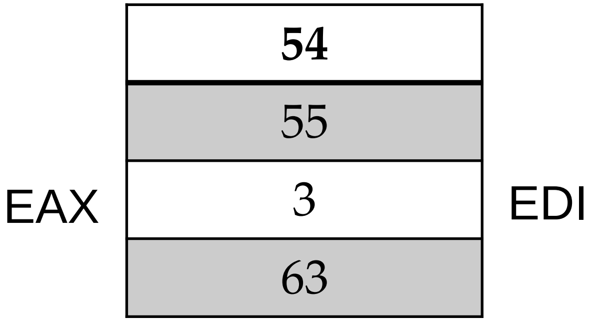
```
replace:
            mov EAX, 0
            mov EDI, 0
            call findLRU
            mov EBX, CPUAccess
            mov [EBP + EDI*4], EBX
            mov EBX ,count
            mov [ESI + EDI*4], EBX
            jmp printcachestatus

findLRU:
            mov EBX, [ESI + EDI*4]
            add EAX, 1
            cmp EBX, [ESI + EAX*4]
            jg recordLRU
            mov EBX, cacheSize
            sub EBX, 1
            cmp EAX, EBX
            jne findLRU
            ret

recordLRU:
            mov EDI, EAX
            mov EBX, cacheSize
            dec EBX
            cmp EAX, EBX
            jne findLRU
            ret
```

cacheBlocks, count = 64, cpu access 5



4
3
2
1

Time table, count = 64



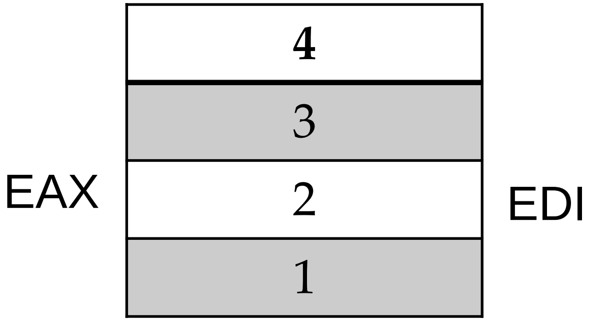
54
55
3
63

Jump to findLRU

LRU Algorithm

```
replace:
        mov EAX, 0
        mov EDI, 0
        call findLRU
        mov EBX, CPUAccess
        mov [EBP + EDI*4], EBX
        mov EBX ,count
        mov [ESI + EDI*4], EBX
        jmp printcachestatus

findLRU:
        mov EBX, [ESI + EDI*4]
        add EAX, 1
        cmp EBX, [ESI + EAX*4]
        jg recordLRU
        mov EBX, cacheSize
        sub EBX, 1
        cmp EAX, EBX
        jne findLRU
        ret

recordLRU:
        mov EDI, EAX
        mov EBX, cacheSize
        dec EBX
        cmp EAX, EBX
        jne findLRU
        ret
```

cacheBlocks, count = 64, cpu access 5



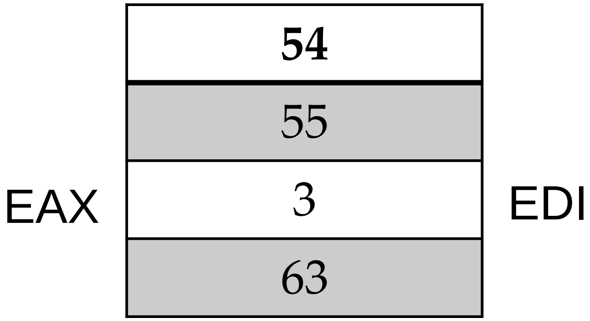
4

3

2

1

Time table, count = 64



54

55

3

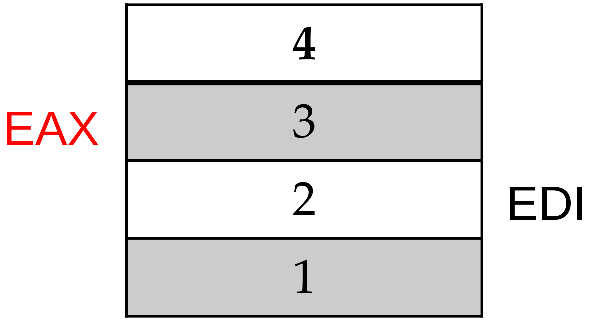
63

EBX = 3

LRU Algorithm

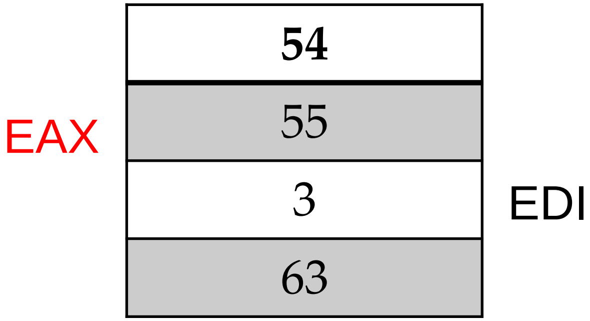
```
replace:
                mov EAX, 0
                mov EDI, 0
                call findLRU
                mov EBX, CPUAccess
                mov [EBP + EDI*4], EBX
                mov EBX ,count
                mov [ESI + EDI*4], EBX
                jmp printcachestatus

findLRU:
                mov EBX, [ESI + EDI*4]
                add EAX, 1
                cmp EBX, [ESI + EAX*4]
                jg recordLRU
                mov EBX, cacheSize
                sub EBX, 1
                cmp EAX, EBX
                jne findLRU
                ret

recordLRU:
                mov EDI, EAX
                mov EBX, cacheSize
                dec EBX
                cmp EAX, EBX
                jne findLRU
                ret
```

cacheBlocks, count = 64, cpu access 5



4
3
2
1

Time table, count = 64



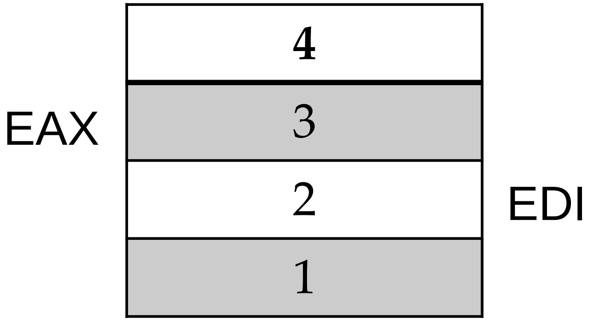
54
55
3
63

EBX = 3

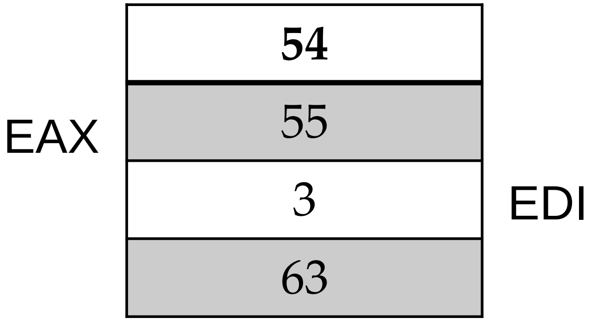
LRU Algorithm

```
replace:
            mov EAX, 0
            mov EDI, 0
            call findLRU
            mov EBX, CPUAccess
            mov [EBP + EDI*4], EBX
            mov EBX ,count
            mov [ESI + EDI*4], EBX
            jmp printcachestatus

findLRU:
            mov EBX, [ESI + EDI*4]
            add EAX, 1
            cmp EBX, [ESI + EAX*4]
            jg recordLRU
            mov EBX, cacheSize
            sub EBX, 1
            cmp EAX, EBX
            jne findLRU
            ret

recordLRU:
            mov EDI, EAX
            mov EBX, cacheSize
            dec EBX
            cmp EAX, EBX
            jne findLRU
            ret
```

cacheBlocks, count = 64, cpu access 5



Time table, count = 64

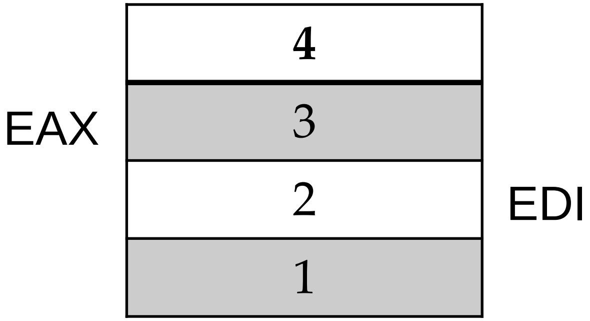


EBX = 3 THEN EBX < 55
Not jmp

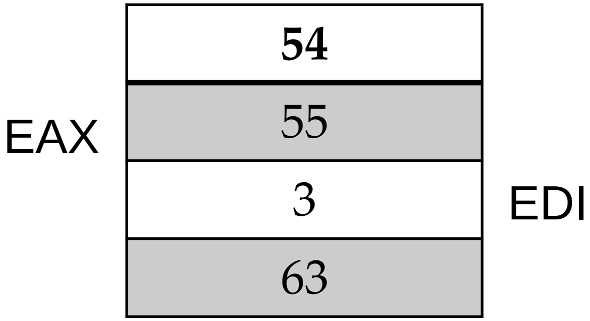
LRU Algorithm

```
replace:
            mov EAX, 0
            mov EDI, 0
            call findLRU
            mov EBX, CPUAccess
            mov [EBP + EDI*4], EBX
            mov EBX ,count
            mov [ESI + EDI*4], EBX
            jmp printcachestatus

findLRU:
            mov EBX, [ESI + EDI*4]
            add EAX, 1
            cmp EBX, [ESI + EAX*4]
            jg recordLRU
            mov EBX, cacheSize
            sub EBX, 1
            cmp EAX, EBX
            jne findLRU
            ret

recordLRU:
            mov EDI, EAX
            mov EBX, cacheSize
            dec EBX
            cmp EAX, EBX
            jne findLRU
            ret
```

cacheBlocks, count = 64, cpu access 5



Time table, count = 64

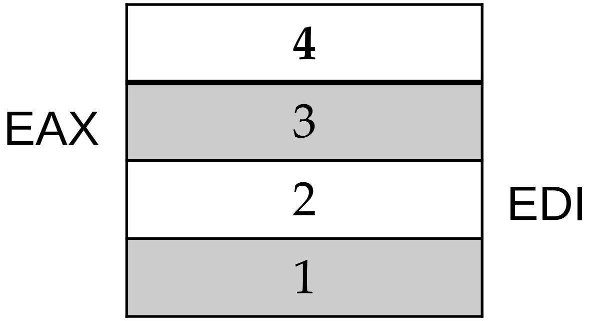


EBX = 4

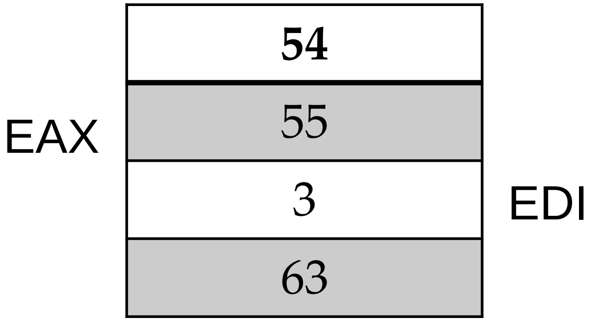
LRU Algorithm

```
replace:
                    mov EAX, 0
                    mov EDI, 0
                    call findLRU
                    mov EBX, CPUAccess
                    mov [EBP + EDI*4], EBX
                    mov EBX ,count
                    mov [ESI + EDI*4], EBX
                    jmp printcachestatus

findLRU:
                    mov EBX, [ESI + EDI*4]
                    add EAX, 1
                    cmp EBX, [ESI + EAX*4]
                    jg recordLRU
                    mov EBX, cacheSize
                    sub EBX, 1
                    cmp EAX, EBX
                    jne findLRU
                    ret

recordLRU:
                    mov EDI, EAX
                    mov EBX, cacheSize
                    dec EBX
                    cmp EAX, EBX
                    jne findLRU
                    ret
```

cacheBlocks, count = 64, cpu access 5



Time table, count = 64

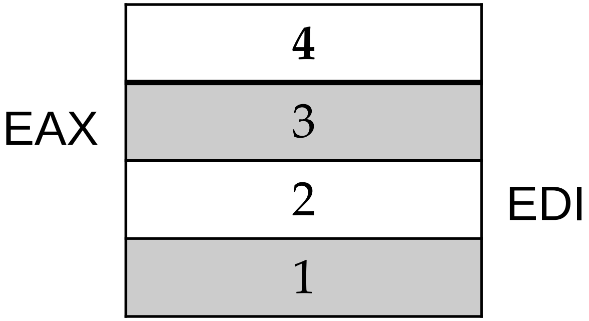


EBX = 3

LRU Algorithm

```
replace:
        mov EAX, 0
        mov EDI, 0
        call findLRU
        mov EBX, CPUAccess
        mov [EBP + EDI*4], EBX
        mov EBX ,count
        mov [ESI + EDI*4], EBX
        jmp printcachestatus

findLRU:
        mov EBX, [ESI + EDI*4]
        add EAX, 1
        cmp EBX, [ESI + EAX*4]
        jg recordLRU
        mov EBX, cacheSize
        sub EBX, 1
        cmp EAX, EBX
        jne findLRU
        ret

recordLRU:
        mov EDI, EAX
        mov EBX, cacheSize
        dec EBX
        cmp EAX, EBX
        jne findLRU
        ret
```

cacheBlocks, count = 64, cpu access 5



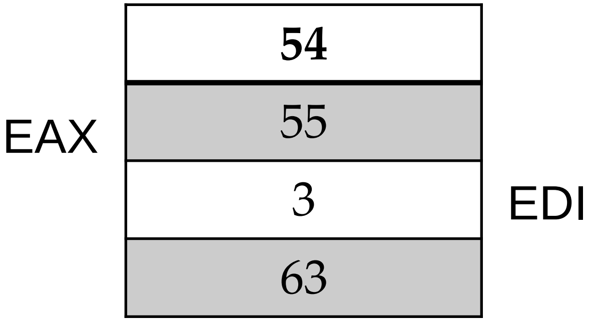
4

3

2

1

Time table, count = 64



54

55

3

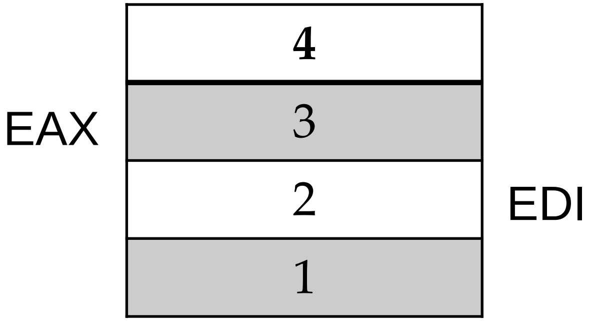
63

EBX = 3 EAX=2 SO
EBX != EAX

LRU Algorithm

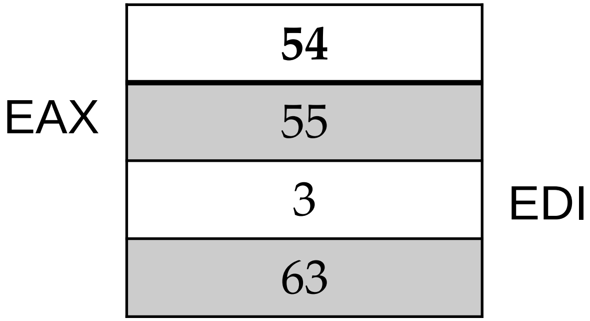
```
replace:
                mov EAX, 0
                mov EDI, 0
                call findLRU
                mov EBX, CPUAccess
                mov [EBP + EDI*4], EBX
                mov EBX ,count
                mov [ESI + EDI*4], EBX
                jmp printcachestatus

findLRU:
                mov EBX, [ESI + EDI*4]
                add EAX, 1
                cmp EBX, [ESI + EAX*4]
                jg recordLRU
                mov EBX, cacheSize
                sub EBX, 1
                cmp EAX, EBX
                jne findLRU
                ret

recordLRU:
                mov EDI, EAX
                mov EBX, cacheSize
                dec EBX
                cmp EAX, EBX
                jne findLRU
                ret
```

cacheBlocks, count = 64, cpu access 5



	4	
EAX	3	
	2	EDI
	1	

Time table, count = 64



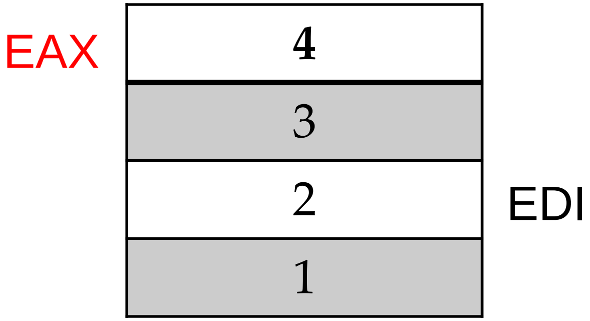
	54	
EAX	55	
	3	EDI
	63	

EBX = 3

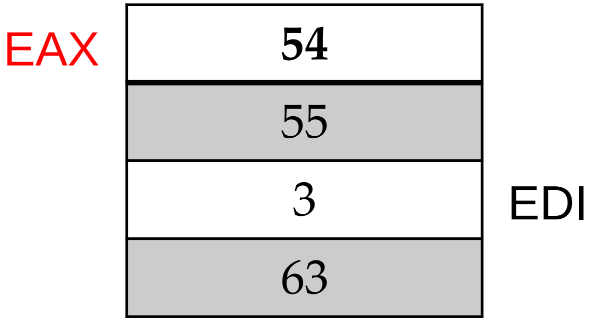
LRU Algorithm

```
replace:
                mov EAX, 0
                mov EDI, 0
                call findLRU
                mov EBX, CPUAccess
                mov [EBP + EDI*4], EBX
                mov EBX ,count
                mov [ESI + EDI*4], EBX
                jmp printcachestatus

findLRU:
                mov EBX, [ESI + EDI*4]
                add EAX, 1
                cmp EBX, [ESI + EAX*4]
                jg recordLRU
                mov EBX, cacheSize
                sub EBX, 1
                cmp EAX, EBX
                jne findLRU
                ret

recordLRU:
                mov EDI, EAX
                mov EBX, cacheSize
                dec EBX
                cmp EAX, EBX
                jne findLRU
                ret
```

cacheBlocks, count = 64, cpu access 5



Time table, count = 64

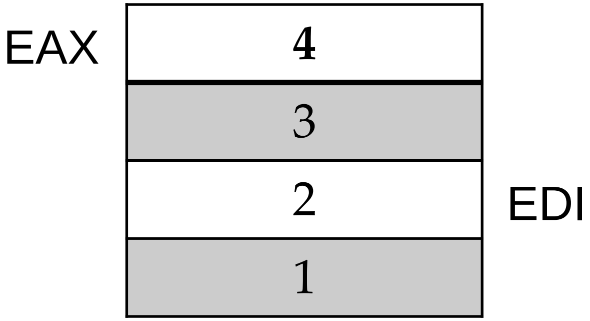


EBX = 3

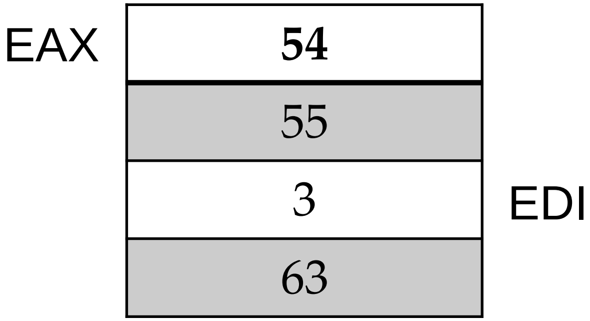
LRU Algorithm

```
replace:
                mov EAX, 0
                mov EDI, 0
                call findLRU
                mov EBX, CPUAccess
                mov [EBP + EDI*4], EBX
                mov EBX ,count
                mov [ESI + EDI*4], EBX
                jmp printcachestatus

findLRU:
                mov EBX, [ESI + EDI*4]
                add EAX, 1
                cmp EBX, [ESI + EAX*4]
                jg recordLRU
                mov EBX, cacheSize
                sub EBX, 1
                cmp EAX, EBX
                jne findLRU
                ret

recordLRU:
                mov EDI, EAX
                mov EBX, cacheSize
                dec EBX
                cmp EAX, EBX
                jne findLRU
                ret
```

cacheBlocks, count = 64, cpu access 5



Time table, count = 64



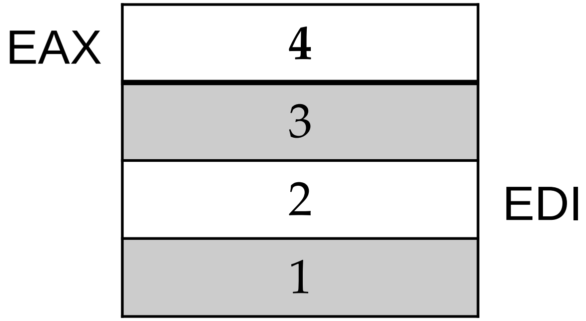
EBX = 3 THEN EBX < 54

Not jmp

LRU Algorithm

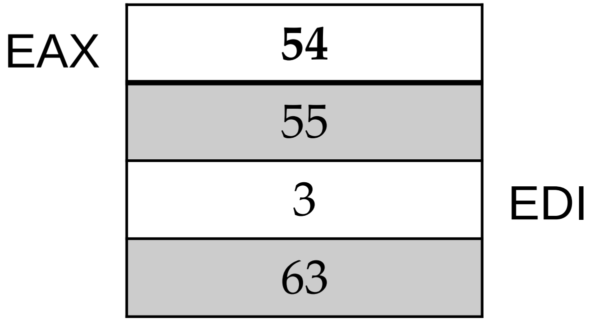
```
replace:
            mov EAX, 0
            mov EDI, 0
            call findLRU
            mov EBX, CPUAccess
            mov [EBP + EDI*4], EBX
            mov EBX ,count
            mov [ESI + EDI*4], EBX
            jmp printcachestatus

findLRU:
            mov EBX, [ESI + EDI*4]
            add EAX, 1
            cmp EBX, [ESI + EAX*4]
            jg recordLRU
            mov EBX, cacheSize
            sub EBX, 1
            cmp EAX, EBX
            jne findLRU
            ret

recordLRU:
            mov EDI, EAX
            mov EBX, cacheSize
            dec EBX
            cmp EAX, EBX
            jne findLRU
            ret
```

cacheBlocks, count = 64, cpu access 5



EAX	**4**	
	3	
	2	EDI
	1	

Time table, count = 64



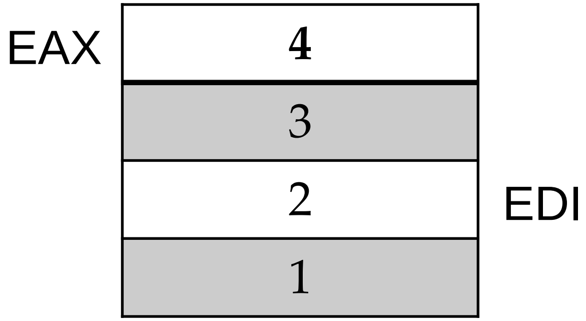
EAX	**54**	
	55	
	3	EDI
	63	

EBX = 4

LRU Algorithm

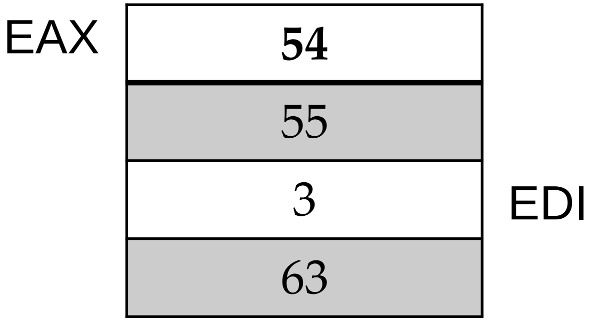
```
replace:
            mov EAX, 0
            mov EDI, 0
            call findLRU
            mov EBX, CPUAccess
            mov [EBP + EDI*4], EBX
            mov EBX ,count
            mov [ESI + EDI*4], EBX
            jmp printcachestatus

findLRU:
            mov EBX, [ESI + EDI*4]
            add EAX, 1
            cmp EBX, [ESI + EAX*4]
            jg recordLRU
            mov EBX, cacheSize
            sub EBX, 1
            cmp EAX, EBX
            jne findLRU
            ret

recordLRU:
            mov EDI, EAX
            mov EBX, cacheSize
            dec EBX
            cmp EAX, EBX
            jne findLRU
            ret
```

cacheBlocks, count = 64, cpu access 5



EAX	**4**	
	3	
	2	EDI
	1	

Time table, count = 64



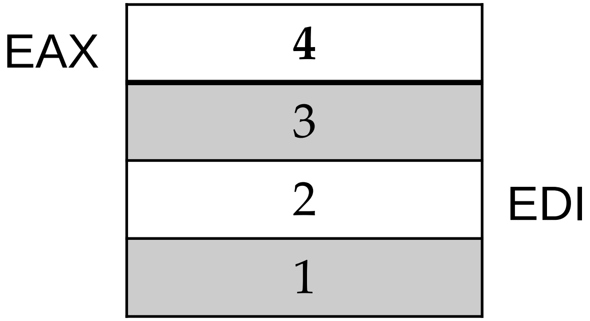
EAX	**54**	
	55	
	3	EDI
	63	

EBX = 3

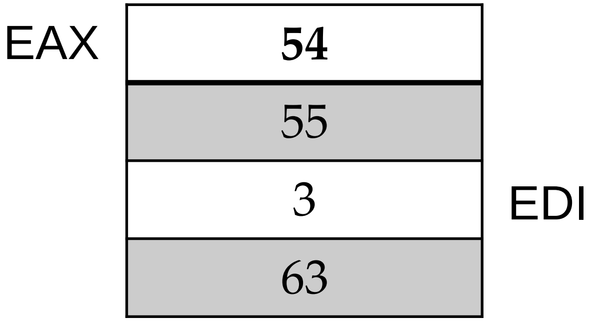
LRU Algorithm

```
replace:
                mov EAX, 0
                mov EDI, 0
                call findLRU
                mov EBX, CPUAccess
                mov [EBP + EDI*4], EBX
                mov EBX ,count
                mov [ESI + EDI*4], EBX
                jmp printcachestatus

findLRU:
                mov EBX, [ESI + EDI*4]
                add EAX, 1
                cmp EBX, [ESI + EAX*4]
                jg recordLRU
                mov EBX, cacheSize
                sub EBX, 1
                cmp EAX, EBX
                jne findLRU
                ret

recordLRU:
                mov EDI, EAX
                mov EBX, cacheSize
                dec EBX
                cmp EAX, EBX
                jne findLRU
                ret
```

cacheBlocks, count = 64,cpu access 5



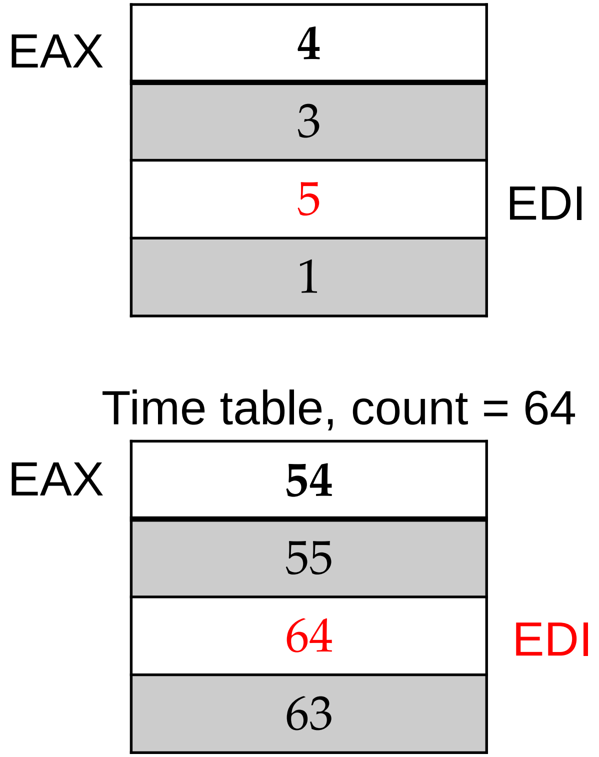
Time table, count = 64



EBX = 3 EAX=3 SO RET

LRU Algorithm

```
replace:
            mov EAX, 0
            mov EDI, 0
            call findLRU
            mov EBX, CPUAccess
            mov [EBP + EDI*4], EBX
            mov EBX ,count
            mov [ESI + EDI*4], EBX
            jmp printcachestatus

findLRU:
            mov EBX, [ESI + EDI*4]
            add EAX, 1
            cmp EBX, [ESI + EAX*4]
            jg recordLRU
            mov EBX, cacheSize
            sub EBX, 1
            cmp EAX, EBX
            jne findLRU
            ret

recordLRU:
            mov EDI, EAX
            mov EBX, cacheSize
            dec EBX
            cmp EAX, EBX
            jne findLRU
            ret
```



cacheBlocks, count = 64, cpu access 5

EAX	4	
	3	
	5	EDI
	1	

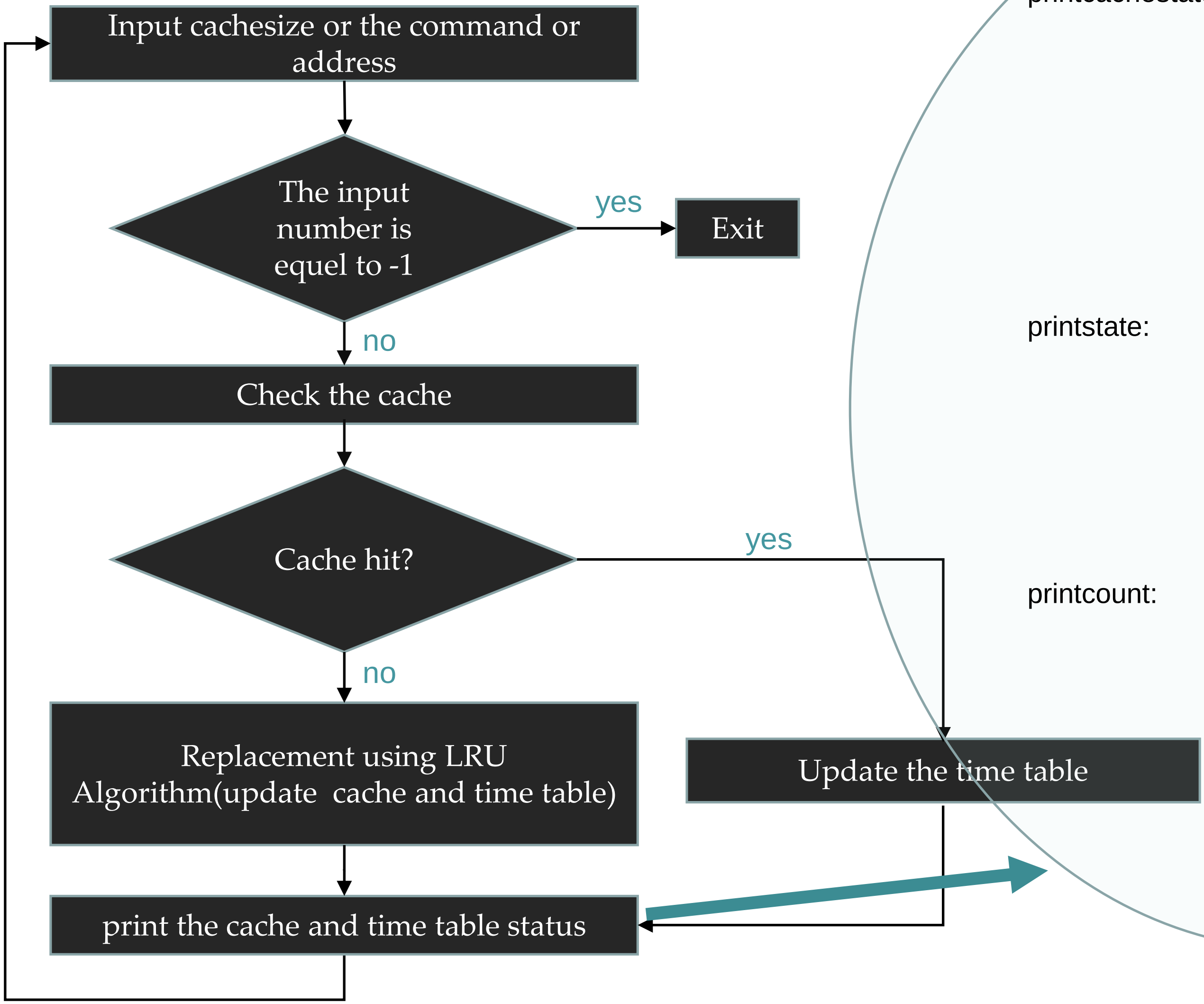
Time table, count = 64

EAX	54	
	55	
	64	EDI
	63	

EBX = 5

EBX = 64

LRU Algorithm



Input cachesize or the command or address

The input number is equel to -1

yes

Exit

no

Check the cache

Cache hit?

yes

no

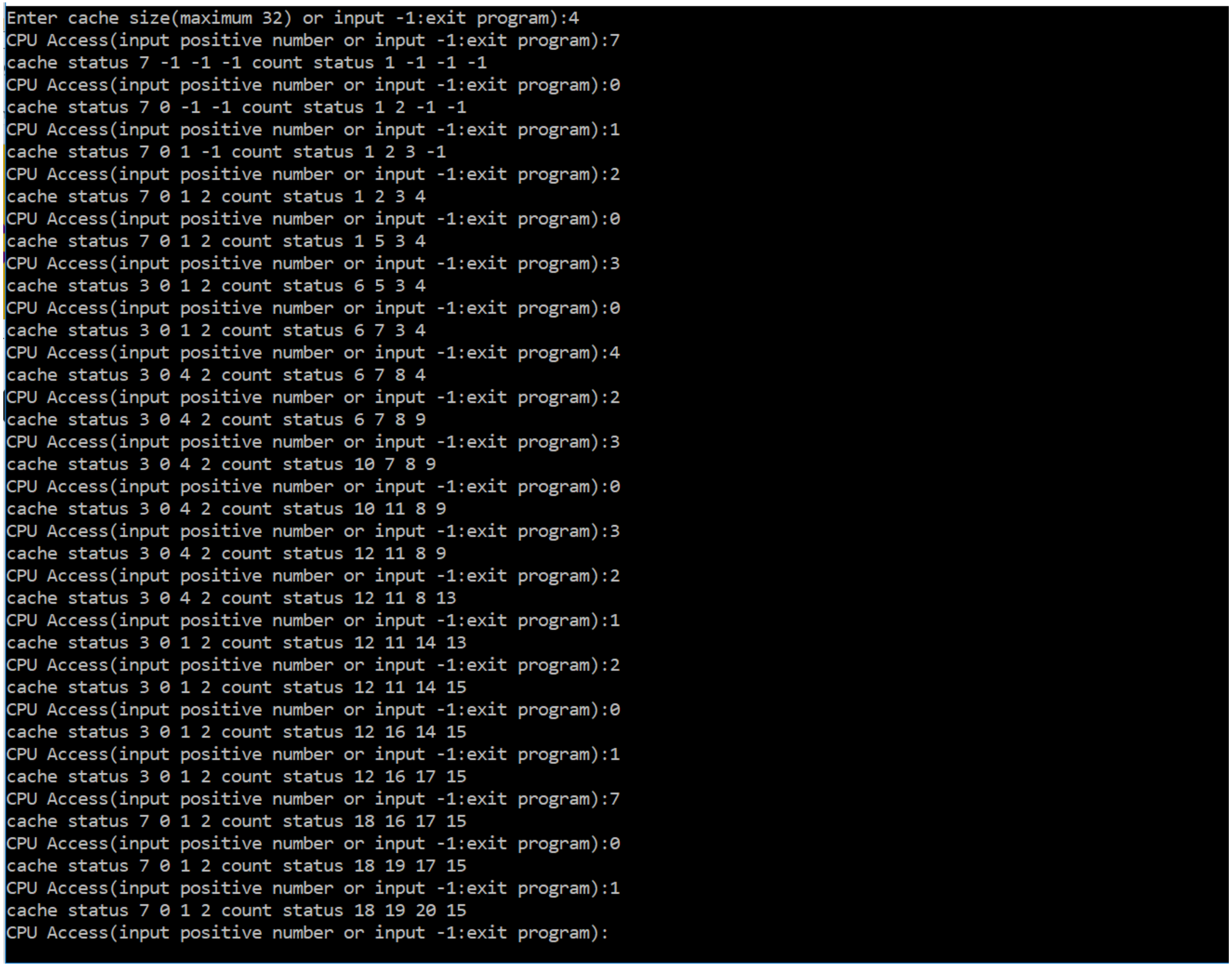
Replacement using LRU Algorithm(update cache and time table)

Update the time table

print the cache and time table status

```
printcachestatus:
        mov EDI, 0
        invoke crt_printf, addr stateFormat
        call printstate
        invoke crt_printf, addr countFormat
        mov EDI, 0
        call printcount
        invoke crt_printf, addr endFormat
        jmp input

printstate:
        mov EAX, [EBP + EDI*4]
        invoke crt_printf, addr outputFormat, EAX
        add EDI, 1
        cmp EDI, cacheSize
        jne printstate
        ret

printcount:
        mov EAX, [ESI + EDI*4]
        invoke crt_printf, addr outputFormat, EAX
        add EDI, 1
        cmp EDI, cacheSize
        jne printcount
        ret
```

Associative mapping implementation

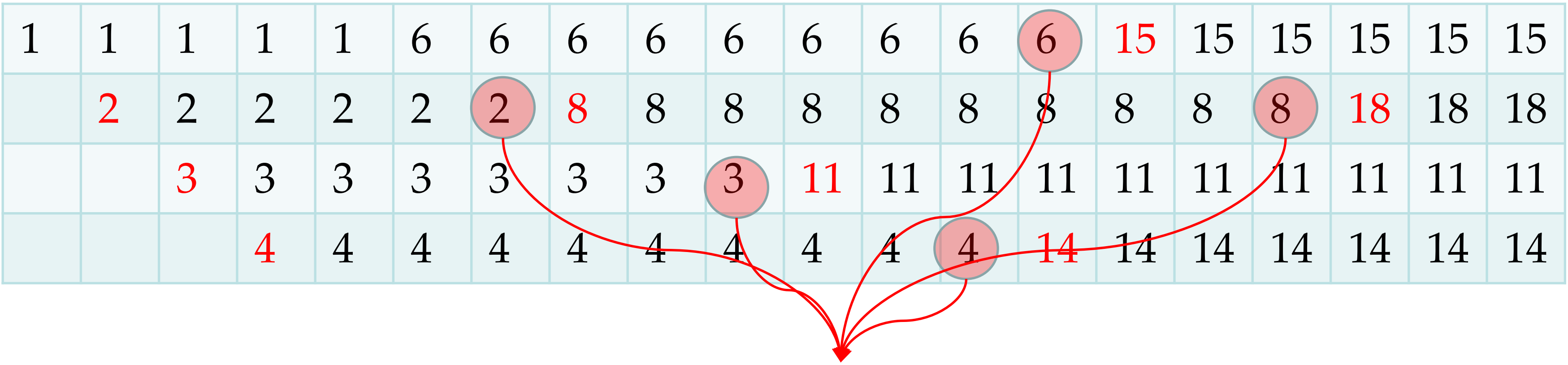


```
Enter cache size(maximum 32) or input -1:exit program):4
CPU Access(input positive number or input -1:exit program):7
cache status 7 -1 -1 -1 count status 1 -1 -1 -1
CPU Access(input positive number or input -1:exit program):0
cache status 7 0 -1 -1 count status 1 2 -1 -1
CPU Access(input positive number or input -1:exit program):1
cache status 7 0 1 -1 count status 1 2 3 -1
CPU Access(input positive number or input -1:exit program):2
cache status 7 0 1 2 count status 1 2 3 4
CPU Access(input positive number or input -1:exit program):0
cache status 7 0 1 2 count status 1 5 3 4
CPU Access(input positive number or input -1:exit program):3
cache status 3 0 1 2 count status 6 5 3 4
CPU Access(input positive number or input -1:exit program):0
cache status 3 0 1 2 count status 6 7 3 4
CPU Access(input positive number or input -1:exit program):4
cache status 3 0 4 2 count status 6 7 8 4
CPU Access(input positive number or input -1:exit program):2
cache status 3 0 4 2 count status 6 7 8 9
CPU Access(input positive number or input -1:exit program):3
cache status 3 0 4 2 count status 10 7 8 9
CPU Access(input positive number or input -1:exit program):0
cache status 3 0 4 2 count status 10 11 8 9
CPU Access(input positive number or input -1:exit program):3
cache status 3 0 4 2 count status 12 11 8 9
CPU Access(input positive number or input -1:exit program):2
cache status 3 0 4 2 count status 12 11 8 13
CPU Access(input positive number or input -1:exit program):1
cache status 3 0 1 2 count status 12 11 14 13
CPU Access(input positive number or input -1:exit program):2
cache status 3 0 1 2 count status 12 11 14 15
CPU Access(input positive number or input -1:exit program):0
cache status 3 0 1 2 count status 12 16 14 15
CPU Access(input positive number or input -1:exit program):1
cache status 3 0 1 2 count status 12 16 17 15
CPU Access(input positive number or input -1:exit program):7
cache status 7 0 1 2 count status 18 16 17 15
CPU Access(input positive number or input -1:exit program):0
cache status 7 0 1 2 count status 18 19 17 15
CPU Access(input positive number or input -1:exit program):1
cache status 7 0 1 2 count status 18 19 20 15
CPU Access(input positive number or input -1:exit program):
```

First-In-First-Out Algorithm

First-In-First-Out Algorithm: Replace the CB that has arrived for the longest period of time (in the past)

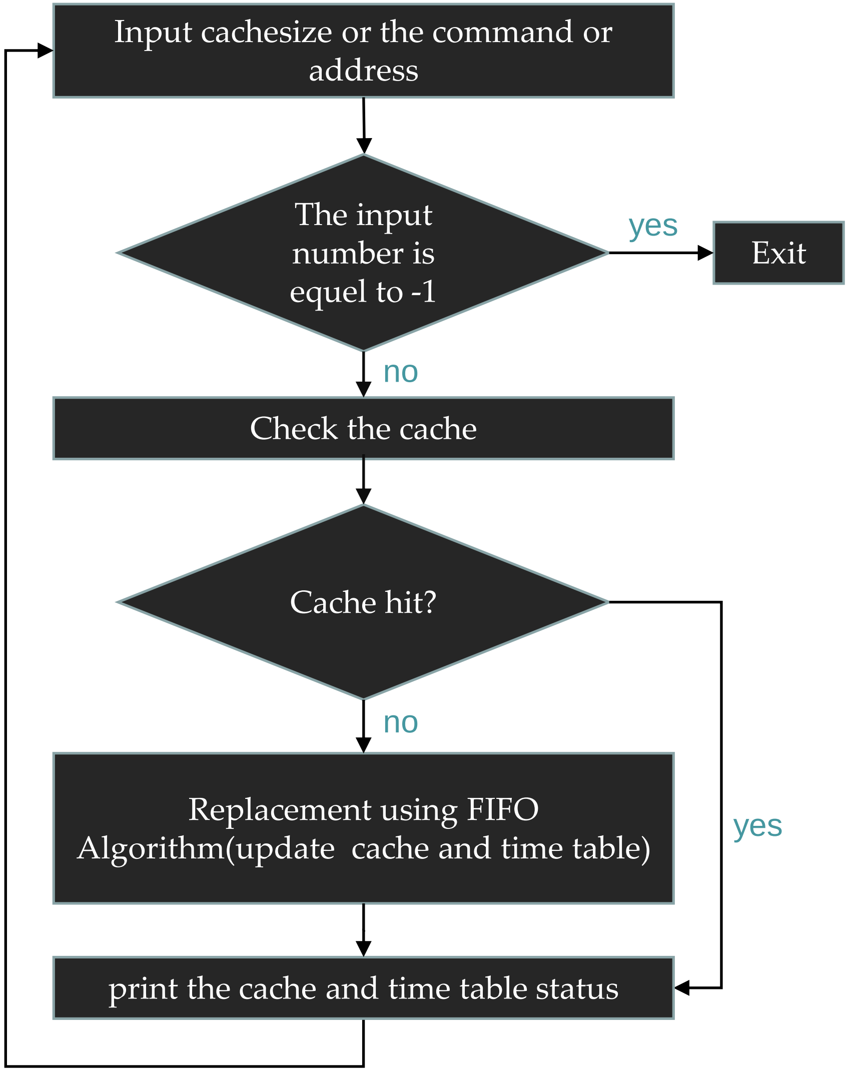
	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
States of cache	7	7	7	7	7	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2
		0	0	0	0	0	0	4	4	4	4	4	4	4	4	4	4	7	7	7
			1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
				2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1



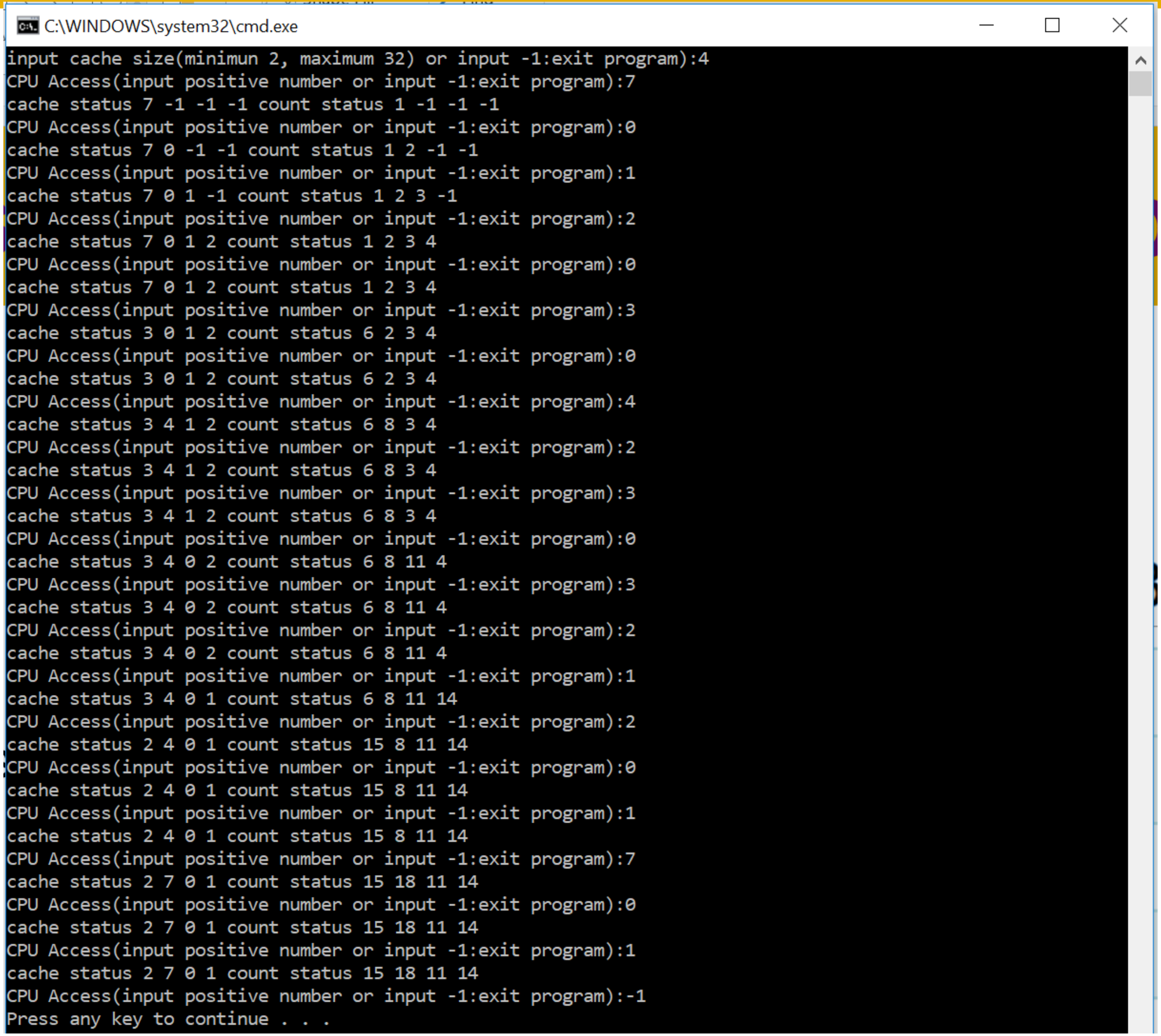
States of count	1	1	1	1	1	6	6	6	6	6	6	6	6	6	15	15	15	15	15	15
		2	2	2	2	2	2	8	8	8	8	8	8	8	8	8	8	18	18	18
			3	3	3	3	3	3	3	3	11	11	11	11	11	11	11	11	11	11
				4	4	4	4	4	4	4	4	4	4	14	14	14	14	14	14	14

The cache block has arrived for the longest period of time!

First-In-First-Out Algorithm


Input cachesize or the command or address
The input number is equel to -1
yes
Exit
no
Check the cache
Cache hit?
no
Replacement using FIFO Algorithm(update cache and time table)
yes
print the cache and time table status

First-In-First-Out Algorithm



```
input cache size(minimun 2, maximum 32) or input -1:exit program):4
CPU Access(input positive number or input -1:exit program):7
cache status 7 -1 -1 -1 count status 1 -1 -1 -1
CPU Access(input positive number or input -1:exit program):0
cache status 7 0 -1 -1 count status 1 2 -1 -1
CPU Access(input positive number or input -1:exit program):1
cache status 7 0 1 -1 count status 1 2 3 -1
CPU Access(input positive number or input -1:exit program):2
cache status 7 0 1 2 count status 1 2 3 4
CPU Access(input positive number or input -1:exit program):0
cache status 7 0 1 2 count status 1 2 3 4
CPU Access(input positive number or input -1:exit program):3
cache status 3 0 1 2 count status 6 2 3 4
CPU Access(input positive number or input -1:exit program):0
cache status 3 0 1 2 count status 6 2 3 4
CPU Access(input positive number or input -1:exit program):4
cache status 3 4 1 2 count status 6 8 3 4
CPU Access(input positive number or input -1:exit program):2
cache status 3 4 1 2 count status 6 8 3 4
CPU Access(input positive number or input -1:exit program):3
cache status 3 4 1 2 count status 6 8 3 4
CPU Access(input positive number or input -1:exit program):0
cache status 3 4 0 2 count status 6 8 11 4
CPU Access(input positive number or input -1:exit program):3
cache status 3 4 0 2 count status 6 8 11 4
CPU Access(input positive number or input -1:exit program):2
cache status 3 4 0 2 count status 6 8 11 4
CPU Access(input positive number or input -1:exit program):1
cache status 3 4 0 1 count status 6 8 11 14
CPU Access(input positive number or input -1:exit program):2
cache status 2 4 0 1 count status 15 8 11 14
CPU Access(input positive number or input -1:exit program):0
cache status 2 4 0 1 count status 15 8 11 14
CPU Access(input positive number or input -1:exit program):1
cache status 2 4 0 1 count status 15 8 11 14
CPU Access(input positive number or input -1:exit program):7
cache status 2 7 0 1 count status 15 18 11 14
CPU Access(input positive number or input -1:exit program):0
cache status 2 7 0 1 count status 15 18 11 14
CPU Access(input positive number or input -1:exit program):1
cache status 2 7 0 1 count status 15 18 11 14
CPU Access(input positive number or input -1:exit program):-1
Press any key to continue . . .
```

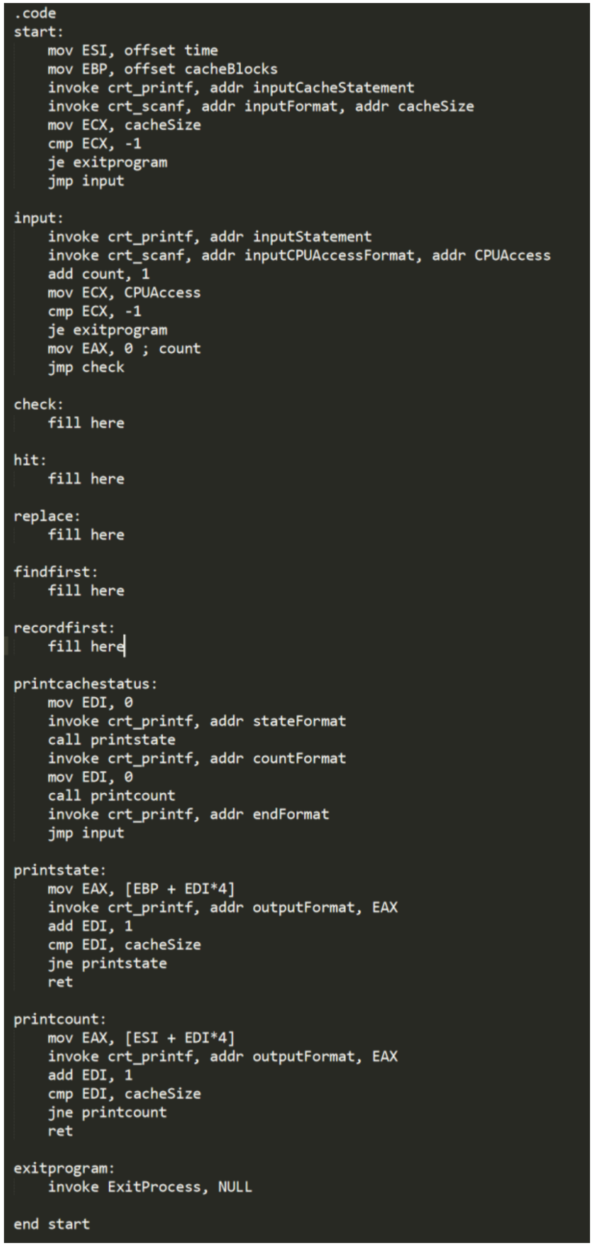
Assignment3

- Implement the FIFO Algorithm

- Fill the five code segments:

check, hit, replace, find first, recordfirst

- Don’t change the start, input and Output(printstate/printcachestatus/printcount) code segments!!!



```
.code
start:
    mov ESI, offset time
    mov EBP, offset cacheBlocks
    invoke crt_printf, addr inputCacheStatement
    invoke crt_scanf, addr inputFormat, addr cacheSize
    mov ECX, cacheSize
    cmp ECX, -1
    je exitprogram
    jmp input

input:
    invoke crt_printf, addr inputStatement
    invoke crt_scanf, addr inputCPUAccessFormat, addr CPUAccess
    add count, 1
    mov ECX, CPUAccess
    cmp ECX, -1
    je exitprogram
    mov EAX, 0 ; count
    jmp check

check:
    fill here

hit:
    fill here

replace:
    fill here

findfirst:
    fill here

recordfirst:
    fill here

printcachestatus:
    mov EDI, 0
    invoke crt_printf, addr stateFormat
    call printstate
    invoke crt_printf, addr countFormat
    mov EDI, 0
    call printcount
    invoke crt_printf, addr endFormat
    jmp input

printstate:
    mov EAX, [EBP + EDI*4]
    invoke crt_printf, addr outputFormat, EAX
    add EDI, 1
    cmp EDI, cacheSize
    jne printstate
    ret

printcount:
    mov EAX, [ESI + EDI*4]
    invoke crt_printf, addr outputFormat, EAX
    add EDI, 1
    cmp EDI, cacheSize
    jne printcount
    ret

exitprogram:
    invoke ExitProcess, NULL

end start
```

Summary

- LRU Algorithm
- First-In-First-Out Algorithm