FII Iasi
- Arhitectura calculatoarelor si sisteme de operare
- Probabilități și Statistică

Orar

Sitemap

Laborator 9

* Compare: **CMP** *source1, source2*
Face diferenta intre *source1* si *source2.* Se seteaza flagurile de status corespunzator. (CF, OF, SF, ZF, AF, PF)
Instructiunea **CMP** este folosita, de obicei, impreuna cu o instructiune de salt conditionat (Jcc)
* Salturi conditionate: **Jcc** *label*
- testarea egalitatii:
Jump if equal/zero **JE/JZ** *label*
Jump of not equal/zero: **JNE/JNZ** *label*
-fara semn
Jump if above/not below or equal: **JA/JNBE** *label*
Jump if above or equal/not below: **JAE/JNB** *label*
Jump if below/not above or equal: **JB/JNAE** *label*
Jump if below or qual/not above: **JBE/JNA** *label*
- cu semn
Jump if greater/not lower or equal: **JG/JNLE** *label*
Jump if greater or equal/not lower: **JGE/JNL** *label*
Jump if lower/not greater or equal: **JL/JNGE** *label*
Jump if lower or qual/not greater: **JLE/JNG** *label*
- testarea flagurilor
Jump if carry: **JC** *label*
Jump if not carry: **JNC** *label*
Jump if overflow : **JO** *label*
Jump if not overflow: **JNO** *label*
Jump if sign(negative): **JS** *label*
Jump if not sign(non-neg):**JNS** *label*
Jump if parity odd: **JPO/JNP** *label*

Jump if parity even: **JPE/JP** *label*

- urmatoarele instructiuni nu intra in programa si sunt optionale:

Jump when register *ECX* is zero: **JECXZ** *label*

Jump when register *CX* is zero: **JCXZ**

* Perform a loop operation using *ECX* or *CX* as a counter. Each time the *LOOP* instruction is executed, the count register is decremented and then checked for 0. If the count is 0, the loop terminates:

Decrement count and jump if count <> 0: **LOOP** *label*
Decrement count and jump if count <> 0 and *ZF=1*: **LOOPE** *label*
Decrement count and jump if count <> 0 and *ZF=1*: **LOOPZ** *label*
Decrement count and jump if count <> 0 and *ZF=0*: **LOOPNE** *label*
Decrement count and jump if count <> 0 and *ZF=0*: **LOOPNZ** *label*

RDTSC - Read TimeStamp Counter (EDX:EAX)
CLD - Clear Direction Flag (operatiile cu stringuri incrementeaza registrii index ESI/EDI)
STD - Set Direction Flag (operatiile cu stringuri decrementeaza registrii index ESI/EDI)

LODS/B/W/D = Load String
MOV AL/AX/EAX, [ESI]
ADD/SUB ESI,1/2/4
REP LODS/B/W/D = SE REPETA INSTRUCTIUNEA LODS/B/W/D DE ECX ORI

STOS/B/W/D = Store string
MOV [EDI], AL/AX/EAX
ADD/SUB ESI,1/2/4
REP STOS/B/W/D = SE REPETA INSTRUCTIUNEA LODS/B/W/D DE ECX ORI

MOVS/B/W/D = Move data from String to String
MOV TEMP, BYTE/WORD/DWORD PTR [ESI]
ADD/SUB ESI, 1/2/4
MOV BYTE/WORD/DWORD PTR [EDI], TEMP
ADD/SUB EDI, 1/2/4
REP MOVS/B/W/D = SE REPETA INSTRUCTIUNEA MOVS/B/W/D DE ECX ORI

SCAS/B/W/D = Scan String
CMP AL/AX/EAX, [EDI]
PUSHFD
ADD/SUB EDI, 1/2/4
POPFD

REPE/REPNE SCAS/B/W/D = SE REPETA INSTRUCTIUNEA SCAS/B/W/D DE MAXIM ECX ORI, CAT TIMP ZF=1 (REPE)/ZF=0(REPNE)

CMPS/B/W/D = Compare String Operands
MOV TEMP, BYTE/WORD/DWORD PTR [ESI]
CMP TEMP, BYTE/WORD/DWORD PTR [EDI]
PUSHFD
ADD/SUB ESI, 1/2/4
ADD/SUB EDI, 1/2/4
POPFD
REPE/REPNE CMPS/B/W/D = SE REPETA INSTRUCTIUNEA CMPS/B/W/D DE MAXIM ECX ORI, CAT TIMP ZF=1 (REPE)/ZF=0(REPNE)

* Salturi neconditionate:
  Jump: **JMP** label

* Implementarea intructiunii *If-else*:

```
int a,b;
if( a > b){
   //instructiuni-1
}else{
   //instructiuni-2
}

--------------------------------------------------------------------------------
;a si b sunt signed deci folosim instructiuni de salt contitionat - signed
MOV eax,a
MOV ebx,b
CMP eax,ebx
JLE _else
   //instructiuni-1
   JMP _end_if
_else:
   //instructiuni-2
_end_if
```

* Implementarea intructiunii *While:*

```
unsigned a,b;
while(a<b){
//instructiuni

}


--------------------------------------------------------------------------------
;a si b sunt unsigned deci folosim instructiuni de salt contitionat - unsigned
MOV eax,a
MOV ebx,b
_while:
CMP eax,ebx
JAE _end_while

//instructiuni

JMP _while

_end_while:
```

* Implementarea intructiunii *do-while:*

```
unsigned a,b;
do{
//instructiuni
}while(a<b);


--------------------------------------------------------------------------------
;a si b sunt unsigned deci folosim instructiuni de salt contitionat - unsigned
MOV eax,a
MOV ebx,b
_do_while:
   //instructiuni
    CMP eax,ebx
    JB _do_while
```

* Implementarea intructiunii *for*

```
short limit;
for(short i=0;i<limit;i++){
   //instructiuni
}

--------------------------------------------------------------------------------
;i si limit sunt signed deci folosim instructiuni de salt contitionat - signed

MOV dx ,limit
XOR cx, cx
_for:
   CMP cx,dx
   JGE _end_for
   //instructiuni
   INC cx
   JMP _for
_end_for:
```

Ex 1.

```
#include <stdio.h>

//Completati exemplul urmator astfel incat functia max sa returneze maximun dintre a si b

int max(int a, int b){
 int maxim;
 _asm{
 //completati
 }
 return maxim;
}

void main(){
 int a,b;
 printf("a = ");
 scanf("%d",&a);
 printf("b = ");
```

```
 scanf("%d",&b);
 printf("MAX(a,b) = %d",max(a,b));
}
```

Ex 2

```
#include <stdio.h>

//Sa se scrie codul in limbaj de asamblare care oglindeste bitii  unui numar

void main(){
 unsigned int number;
 number = 140;

 _asm{
 /* Completati */
 }
 if( number != 822083584)
 printf("Failed! Your result is %d\n",number);
 else
 printf("OK!");
}
```

l9p1r.cpp (0k)	Alexandru Baetu, 10 dec. 2013, 23:34	v.1	
l9p2r.cpp (0k)	Alexandru Baetu, 4 dec. 2014, 02:09	v.3	