POP81 Processzor

Architektúraterv

Tartalom

[1. Változatási lista 3](#_Toc55423963)

[1.1. Tartalom változások 4](#_Toc55423964)

[1.2. Formázási változások 5](#_Toc55423965)

[2. Általános 6](#_Toc55423966)

[2.1. Alapvető tulajdonságok 7](#_Toc55423967)

[3. Hardver, gépi kód 8](#_Toc55423968)

[3.1. Gépi kódú utasítások 9](#_Toc55423969)

[3.2. Gépi kódú utasítások 10](#_Toc55423970)

[3.3. Regiszterek 12](#_Toc55423971)

[3.4. Zászlók 13](#_Toc55423972)

[4. Programozhatóság 14](#_Toc55423973)

[4.1. A POP81 assembly nyelv 15](#_Toc55423974)

[4.1.1. Cízmési jelölések 15](#_Toc55423975)

[4.1.2. Számjelölések 15](#_Toc55423976)

[4.1.3. Megjegyzések 16](#_Toc55423977)

[4.1.4. Meta-utasítások 16](#_Toc55423978)

[4.1.5. Címkék 16](#_Toc55423979)

[4.1.6. Fordítandó műveletek 18](#_Toc55423980)

[4.2. Programozási alaptételek popasm nyelven 23](#_Toc55423981)

[4.2.1. Elágazások 23](#_Toc55423982)

[4.2.2. Ciklusok 23](#_Toc55423983)

[4.2.3. Szubrutinok 24](#_Toc55423984)

[4.2.4. Alapvető algoritmusok 24](#_Toc55423985)

[4.3. A PopAsmWin assembler szoftver 28](#_Toc55423986)

[4.3.1. --verbose direktíva 28](#_Toc55423987)

# Változtatási lista

## Tartalom változások

* 2020. 11. 02.: Első kiadott verzó

## Formázási változások

* 2020. 11. 04.: Első kiadott verzió

# Általános

## Alapvető tulajdonságok

* Neumann-elv
* 8 bites adatbusz
* 16 bites címbusz
* 8 db általános 8 bites regiszter, 4 db 16 bites regiszter 2-2 8 btest összevonva
* nincs pipeline
* RISC felépítés
* kétoperandusos utasítások kivéve ahol nem értelmezhető
* adat- és kódverem hardveresen külön kezelve

# Hardver, gépi kód

## Gépi kódú utasítások

## Gépi kódú utasítások

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Típus** | **Művelet** | **Mnemonik** | **Variációk** | | | | | | | | | **Leírás** |
| **--** | **r** | **R** | **L** | **rr** | **RR** | **Rr** | **rL** | **RL** |
| "Filler" műveletek | nope | NOPE | X |  |  |  |  |  |  |  |  | Semmi |
| halt | HALT | X |  |  |  |  |  |  |  |  | Megállítja a CPU-t |
| Adatmozgatás | move | MOVE, JUMP, PUSH, PULL, CALL, RETURN |  |  |  |  | X | X | X | X | X | Adat másolása forrásból célba. |
| Memóriakezelés | load | --- | X |  |  |  |  |  |  |  |  | MD tartalma olvasás adatbuszról |
| store | --- | X |  |  |  |  |  |  |  |  | MD tartalom kiírásra adatbuszra |
| Elágazások | jz | JIFZ | X |  |  |  |  |  |  |  |  | Ugrás, ha nulla |
| jnz | JNOZ | X |  |  |  |  |  |  |  |  | Ugrás, ha nem nulla |
| jc | JIFC | X |  |  |  |  |  |  |  |  | Ugrás ha carry |
| jnc | JNOC | X |  |  |  |  |  |  |  |  | Ugrás, ha nem carry |
| jv | JIFV | X |  |  |  |  |  |  |  |  | Ugrás, ha overflow |
| jnv | JNOV | X |  |  |  |  |  |  |  |  | Ugrás, ha nem overflow |
| jg | JIFG | X |  |  |  |  |  |  |  |  | Ugrás, ha nagyobb |
| jng | JNOG | X |  |  |  |  |  |  |  |  | Ugrás, ha nem nagyobb |
| Aritmetika | add | ADD, INCR |  |  |  |  | X | X | X | X | X | Öszeadás |
| sub | SUB, DECR |  |  |  |  | X | X | X | X | X | Kivonás |
| mul | MUL |  |  |  |  | X | X | X | X | X | Szorzás |
| div | DIV |  |  |  |  | X | X | X | X | X | Osztás |
| Logika | and | AND |  |  |  |  | X |  |  | X |  | Bitenkénti ÉS |
| or | OR |  |  |  |  | X |  |  | X |  | Bitenkénti VAGY |
| not | NOT |  | X | X |  |  |  |  |  |  | Bitenkénti negálás |
| Összehasonlítás | cmp | COMP |  |  |  |  | X | X | X | X | X | Összehasonlít két értéket; beállítja Z és G zászlókat |
| Shift és forgás | lsh | LSFT |  |  |  |  | X |  | X | X | X | Regiszter balra shifitelése n-szer |
| rsh | RSFT |  |  |  |  | X |  | X | X | X | Regiszter jobbra shifitelése n-szer |
| lrot | LROT |  |  |  |  | X |  | X | X | X | Regiszter balra rotálása n-szer |
| rrot | RROT |  |  |  |  | X |  | X | X | X | Regiszter jobbra rotálása n-szer |

## Regiszterek

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Típus** | **Név** | **Méret (bit)** | **Belső cím** | **Leírás** |
| Általános | R1 | 8 | 0b00000 | - |
| R2 | 8 | 0b00001 | - |
| R3 | 8 | 0b00010 | - |
| R4 | 8 | 0b00011 | - |
| R5 | 8 | 0b00100 | - |
| R6 | 8 | 0b00101 | - |
| R7 | 8 | 0b00110 | - |
| R8 | 8 | 0b00111 | - |
| RA | 16 | 0b10000 | Kombinált regiszter, R2-R1 |
| RB | 16 | 0b10001 | Kombinált regiszter, R4-R3 |
| RC | 16 | 0b10010 | Kombinált regiszter, R6-R5 |
| RD | 16 | 0b10011 | Kombinált regiszter, R8-R7 |
| Speciális | PC | 16 | 0b11000 | Program counter |
| DB | 16 | 0b11010 | Aadatverem kezdőcím mutató |
| DP | 16 |  | Adatverem mutató |
| DS | 16 |  | Adatverem méret regiszter |
| CB | 16 | 0b11100 | Kódverem kezdőcím mutató |
| CP | 16 |  | Kódverem mutató |
| CS | 16 |  | Kódverem méret mutató |
| MA | 16 | 0b11011 | Memóriacím mutató |
| MD | 8 | 0b01000 | Memória adat |
| FL | 8 | 0b01001 | Zászló regiszter |

## Zászlók

|  |  |  |
| --- | --- | --- |
| **Bit** | **Név** | **Leírás** |
| 0 | Zero | Set if instruction output is exactly zero |
| 1 | Carry | Set if there was a carry-over at the most significant bit |
| 2 | Sign | Set f the most significant bit is 1 |
| 3 | Parity | Set if the least significant bit is 1 |
| 4 | Half-Carry | Set if there was a cary-over in the n/2th bit (4 or 8) |
| 5 | Overflow |  |
| 6 | Greater | Set by the COMP instruction if register A is bigger than B |
| 7 |  |  |

# Programozhatóság

## A POP81 assembly nyelv

A POP81 assembly nyelve (továbbiakban: **popasm**) durván követi a 8086 szintaktikáját, utasításai pedig a legtöbb esetben 1:1 megfelelnek egy gépi utasításnak (az alól az indirekt címzés kivételt képez).

* Minden sor pontosan egy utasítást tartalmat
* Megjegyzések hozzáfűzésére van lehetőség
* Az assembly utasíátsok három csoportra oszthatók: fordítandó, meta és label

Jelentős eltérések vannak azonban a számjelölésekben, meta-utasításokkban és sok más egyéb téren.

### Cízmési jelölések

A leírás egyszerűsítése és általánosítása érdekében a különböző forrás- és célcímek típusait a következőképpen jelöljük a továbbiakban:

|  |  |
| --- | --- |
| **Jelölés** | **Leírás** |
| r | 8 bites regiszter |
| R | 16 bites regiszter |
| x | 8 bites konstant |
| X | 16 bites konstant |
| i | 8 bites indirekt érték (pointer) |
| I | 16 bites indirekt érték (pointer) |

Az ezekből következő kombinációk két operandus esetén:

|  |  |
| --- | --- |
| **Jelölés** | **Leírás** |
| rr | 8 bites regiszter |
| Rr |  |
| RR |  |
| rx | 16 bites regiszter |
| RX |  |
| rI |  |
| Ir |  |
|  |  |
| x | 8 bites konstant |
| X | 16 bites konstant |
| i | 8 bites indirekt érték (pointer) |
| I | 16 bites indirekt érték (pointer) |

### Számjelölések

A popasm nyevben többféle módon lehet számszerű értékeket jelölni attól függően, hogy milyen módon kívánjuk lfehazsnálni őket:

|  |  |  |
| --- | --- | --- |
| **Jelölésrendszer** | **Példa** | **Leírás** |
| Decimális jelölés | * 1 * 28 * 987 | Tízes számrendszerű jelölés, a közéletben használt jelölésnek megfelelő mód. |
| Hexadecimális jelölés (1) | * 12h * F8h * F120h * 0FFEh | 0-1 és A-F karakterek 0-15 értékek közötti jelölése, és kis „h” karakterrel zárása. kezdő nullák kiírása kötelező. |
| Hexadecimális jelölés (2) | * 0x45 * 0x08FF | Ld. előző jelölés, de itt a kezdés mindig „0x” karakterpáros. |
| Bináris jelölés (1) | * 00000000b * 11001010b * LLHLLHLLb | 0-1 ill. L-H karakterek bináris értékek jelölésére. Csak 8 bites használat engedélyezett. Záró kis „b” karakter. Kezdő nullák kiírása kötelező. |
| Bináris jelölés (2) | * 0b00010001 * 0bHHHHLLLL | Ld. előző jelölés, de itt a kezdés mindig „0b” karakterpáros. |
| „k” jelölés | * $32k * $8k16 | „$” karakterrel kezdődő és „k” karakterrel elválasztott számpáros. A „k” karakter előtt lévő szám 1024-el megszorzódik, tehát $32k51 esetén az érték (32 \* 1024) + 51. |

### Megjegyzések

Megjegyzést a ; karakter elhelyezésével lehet írni. Mindent, ami ez után a karakter után következik, csak a szöveges fájlban fog helyet foglalni, az assembler program figyelmen kívül hagyja.

**Példák:**

Teljes sor:

;Ez a sor teljes egésézben egy megjegyzés

Sorközbeni:

MOVE ra, rb ;Sorközbeni komment – átmozgatjuk rb-t ra-ba

### Címkék

A címkékkel a programkódban kijelölhetők különböző, megnevezett címek, tipikusan abból a vélból, hogy a programfutás ezekre a címekre ugorjon pl. ciklusoknál, elgazásoknál. és szubrutin hívásoknál.

Címkét definiálni lehet és hivatkozni rá. Deifiniáláskor a sor végére : karaktert kell rakni, míg hivatkozásnál a címet : karakterrel kell kezdeni. Fordítás során a címkék abszolút címekre kerülnek fordításra, de ezeket a címeket a [@base](#_@base) utasításban beállított értékkel ofszeteli a fordítás.

**Példák:**

címke1: ;definiáljuk a ”címke1” nevű címkét

COMP ra, rb

JIFG :címke1 ;”címke1” címére ugrunk, ha ra > rb

…

CALL :szubrutin ;szubrutin hívás

…

HALT

szubrutin:

PUSH rb ;rb értékét letároljuk

MOVE rb, 0b1010101010101010 ;rb-ben a páros bites 1-ek

AND ra, rb ;rb masznak használjuk ra-ra

POP rb ;rb-be a meglevő értékét visszatötljük

RET ;vissaztérünk a hívási helyre

### Meta-utasítások

Ezek olyan utasítások amelyek nem kerülnek közvetlenül lefordításra gépi kódra, de befolyással vannak a további utasítások fordítási módjára. Nevezhetjük ezeket az utasításokat **előfodító direktívák**nak is. Minden meta-utasítás @ karakterrel kezdődik

#### @base

|  |  |
| --- | --- |
| A base meta-utasítás adja meg, hogy mi legyen a fordítás kezdőcíme. Ez a progrm betöltődésénél ill. az ugráscímeknél megkerülhetetlen fontosságú.  **Példák:**  ;PROGRAM 1  @base 0x0000 ;alapértelmezett, a báziscím 0  label1:  …  JUMP label1 ;az ugrás 0x0000 címre fog történni  ;PROGRAM 2  @base 0x1000 ;alapértelmezett, a báziscím 1k  label1:  …  JUMP label1 ;az ugrás 0x1000 címre fog történni | @base |

#### @def

|  |  |
| --- | --- |
| A def meta-utasítás egy konstans értéknek ad a továbbiakban egy nevet. Meg kell jegyezni, hogy ez a név csak forráskódszinten köti össze a különböző értékeket, fordítás során a forráskódba a konstant érték literálként kerül behelyettesítésre! | @def |
| **Példák:** |
| ;PROGRAM 1  @def TÖMB\_HOSSZ, 10  MOVE ra, TÖMB\_HOSSZ ; -> MOVE ra, 10 |

#### @var

|  |  |
| --- | --- |
| A var meta-utasítás egy fix memóriaterületnek ad a továbbiakban egy nevet. Meg kell jegyezni, hogy bármilyen utasítás ami a var által lérehozott nevesített memóriaterületet használja indirekt címzésű utasítássá lesz alakítva! | @var |
| **Példák:** |
| ;PROGRAM 1  @var TÖMB\_HOSSZ, 10  MOVE ra, TÖMB\_HOSSZ ; -> MOVE ra, [auto] – ra = 10  INCR TÖMB\_HOSSZ ; -> INCR [auto]  MOVE rb, TÖMB\_HOSSZ ; rb = 11 |

### Fordítandó műveletek

A fordítandó műveletek közé sorolunk minden olyan műveletet/utasítást, ami közvetlenül gépi utasítás(oka)t eredményez. Ezek a máveletek a legtöbb esetben egyben megfelelnek egy gépi utasítással – ez alól a legjelentősebben az indirekt címzésből származó műveletek számítanak kivételnek.

Minden fordítandó művelet egy mnemonikból áll és függően a művelettől, 0, 1 vagy 2 operandustól. Az utolsó operandus általánosságban bármilyen adat lehet (regiszter, literál, indirekt), az utolsó előtti opernadus legtöbbször csak regiszter és indirekt adat lehet.

#### NOPE

A NOPE utasítás olyan utasítás, ami semmilyen utasítást ad a processzornak, tehát semmiféle belső vagy külső állaotváltozást nem fog eredményezni a PC regiszter növekedésén kívül. Ez pl. kiszámítható időzítések programozására alkalmas.

**Bemenetek száma**: 0

**Címzési módok**: -

;PROGRAM 1

;várakozunk 5(+2) órajelciklusig

CALL :várakoz

HALT

várakoz:

NOPE

NOPE

NOPE

NOPE

NOPE

RET

#### HALT

A HALT utasítás megállítja a processzort és a további utasítások végrehajtását. A megállt állapotból csak és kizárólag egy hardveres reset művelettel lehet kilépni így használata csak végzetes hibák esetén ajánlott, de még ebben az esetben is inkább egy végtelen ciklus használata kedvezőbb.

|  |  |
| --- | --- |
| **Bemenetek száma**: | 0 |
| **Címzési módok**: | - |
| Gépi kód megfeleltetés : | halt |

;PROGRAM 1

...

JIFZ :végzetes ;végzetes hiba történt, azonnal megállítjuk a

;processzort

...

végzetes:

#### MOVE

A MOVE valószínűleg a legsokoldalúbban használt utasítás, tetszőleges helyről teszőleges helyre való adatmozgatásra alkalmazható. Ide értendő a regiszterek közötti, regiszterbe konstants, és regiszterbe memóriacímről történő adamozgatás is.

Az indirekt címzés esetén alkalmazott MOVE műveletek makro-műveletnek számítanak így a végrehajtásuk is több órajelciklus!

**Bemenetek száma**: 2

**Címzési módok**: rr, Rr, rx, RX, rI, RI

;PROGRAM 1 (rx, rX)

MOVE r1, 1 ;r1 = 1

MOVE r2, 0x02 ;r2 = 2

MOVE r3, FFh ;r3 = 255

;PROGRAM 2 (RX)

MOVE ra, 5 ;ra = 1

MOVE rb, 0xFF00 ;rb = 65 280

MOVE rc, $32k16 ;rc = 32 784

;PROGRAM 3 (rr)

MOVE r1, 5 ;r1 = 1

INCR r1

MOVE r2, r1 ;r2 = 2

;PROGRAM5 (Ir)

MOVE [$32k1], r1 ;RAM<-r1 @ 32k1

MOVE [rb], 32 ;RAM<-32

;PROGRAM4 (rI)

MOVE r1, [$32k1] ;r1<-RAM @ 32k1

MOVE r2, [ra] ;r2<-RAM @ ra

;PROGRAM6 (II)

MOVE [0x0020], [ra] ;RAM<-RAM @ ra

#### JUMP

A JUMP utasítás a MOVE utasítás egy specializált változata, ebben az esetben a célregiszter minden esetben a PC (program counter). Leggyakoribb használata a címkés ugrás.

**Bemenetek száma**: 1

**Címzési módok**: X, R, I

;PROGRAM 1

JUMP :start

NOPE ;nem hajtódik végre

HALT ;nem hajtódik végre

start:

#### POP

|  |  |
| --- | --- |
| A POP utasítás az adatveremből vesz ki egy elemet és írja be a megadott regiszterbe. A verem utolsó elemének címét mindig a DSI regiszter tartalmazza. Ha a velem üres (DSB és DSI értéke egyenlő) a művelet kiszámíthatatlan ereményekhez vezethet! | POP |
| **Bemenetek száma**: 1  **Címzési módok**: r, R |
| **Példa:** |
| ;PROGRAM 1  POP r1 ;r1-be |

#### PUSH

#### CALL

#### RET

#### JIFZ

#### JNOZ

#### JIFC

#### JNOC

#### JIFV

#### JNOV

#### JIFG

#### JNOG

#### ADD, INC

#### SUB, DEC

#### MUL

#### DIV

#### AND

#### OR

#### NOT

#### COMP

#### LSFT

#### RSFT

#### LROT

#### RROT

## Programozási alaptételek popasm nyelven

TODO

### Elágazások

#### Egyszerű ha-kifejezés - egyenlőségvizsgálat

|  |  |
| --- | --- |
| *HA RA = RB AKKOR*  *<ÁG1>* | COMP ra, rb  JNOZ :ágvége  <ÁG1>  ágvége: |

#### Egyszerű ha-kifejezés – relációvizsgálat

|  |  |
| --- | --- |
| *HA RA > RB AKKOR*  *<ÁG1>* | COMP ra, rb  JNOG :ágvége  <ÁG1>  ágvége: |

#### Egyszerű elágazás

|  |  |
| --- | --- |
| *HA RA = RB AKKOR*  *<ÁG1>*  *KÜLÖNBEN*  *<ÁG2>* | COMP ra, rb  JNOZ :ág2  <ÁG1>  JUMP :havége  <ÁG2>  havége: |

### Ciklusok

#### Egyszerű ciklus – örökké futás

|  |  |
| --- | --- |
| *AMÍG IGAZ*  *<ÁG>* | ciklus:  <ÁG>  JUMP :ciklus |

#### Előltesztelő ciklus

|  |  |
| --- | --- |
| *AMÍG RA = RB*  *<ÁG>* | ciklus:  COMP ra, rb  JNOZ :vége  <ÁG>  JUMP :ciklus  vége: |

#### Hátultesztelő ciklus

|  |  |
| --- | --- |
| *CSINÁLD*  *<ÁG>*  *AMÍG RA = RB* | ciklus:  <ÁG>  COMP ra, rb  JIFZ :ciklus |

#### For ciklus

|  |  |
| --- | --- |
| *I := 0*  *AMÍG I < 10*  *<ÁG>*  *I := I+1* | MOVE ra, 0  ciklus:  COMP ra, 10  JNOZ :vége  <ÁG>  INC ra  vége: |

### Szubrutinok

#### Szubrutin hívás visszatérési érték nélkül

|  |  |
| --- | --- |
| *A()*  *…*  *FÜGGÉNY A*  *<ÁG>*  VÉGE | CALL :A  ...  A:  <ÁG>  RET |

#### Szubrutin hívás visszatérési értékkel

|  |  |
| --- | --- |
| *RA = B()*  *…*  *FÜGGÉNY B*  *<ÁG>*  VÉGE RB | CALL :B  MOVE ra, rb  ...  B:  <ÁG>  RET |

#### Szubrutin hívás paraméterekkel

|  |  |
| --- | --- |
| *RA = C(20)*  *…*  *FÜGGÉNY C*  *<ÁG>*  VÉGE RB | MOVE rd, 20  CALL :C  MOVE ra, rb  ...  C: ;RD itt bemenet  <ÁG>  RET |

### Alapvető algoritmusok

#### Összegzés

Összeadja egy tömb elemeit.

**Pszeudokód:**

*osszeg = 0*

*ciklus i = 0 .. n -1*

*osszeg = osszeg + t[i]*

*ciklus vége*

*ki osszeg*

**Assembly:**

MOVE rb, 0x1234 ;tömb memóriacíme

MOVE r7, 10 ;elemek száma

...

ÖSSZEGZÉS:

MOVE ra, 0 ;eredmény

MOVE r8, 0 ;iterátor

CIKLUS:

COMP r6, r5 ;ha rc nagyobb, mint rb, G = 1

JIFG :VÉGE ;ha rb > rc

MOVE r10, [rb]

ADD ra, r10

INCR r6

INCR rb

JUMP :CIKLUS

VÉGE:

RET

#### Megszámolás

Adott feltételek alapján a tömb bizonyos elemeit megszámolom.

Pl.: Megszámoljuk mennyi negatív szám van a tömbben

**Pszeudokód:**

szamlalo = 0

ciklus i = 0 .. n - 1

ha t[i] < 0 akkor

szamlalo = szamlalo + 1

ha vége

ciklus vége

ki szamlalo

**Assembly:**

MOVE rb, 0x1234 ;tömb memóriacíme

MOVE r7, 10 ;elemek száma

...

MEGSZÁMOLÁS:

MOVE ra, 0 ;eredmény

MOVE r8, 0 ;iterátor

CIKLUS:

COMP r6, r5 ;ha rc nagyobb, mint rb, G = 1

JIFG :VÉGE

MOVE r10, [rb] ; }

COMP r10, 0 ; |

JIFG :TOVÁBB ; |

INCR ra ; > Ciklus törzs

TOVÁBB: ; |

INCR r6 ; |

INCR rb ; }

JUMP :ciklus

VÉGE:

RET

#### Eldöntés

Szeretnénk tudni, hogy egy érték megtalálható-e egy tömbben.

**Pszeudokód:**

*i = 0*

*ciklus amíg i<n és t[i]<> ker*

*i=i+1*

*ciklus vége*

**Assembly:**

MOVE rb, 0x1234 ;tömb memóriacíme

MOVE r7, 10 ;elemek száma

MOVE r9, 5 ;keresett érték

...

ELDÖNTÉS:

MOVE ra, -1 ;eredmény

MOVE r8, 0 ;iterátor

CIKLUS:

COMP r6, r5 ;ha rc nagyobb, mint rb, G = 1

JIFG :VÉGE

MOVE r10, [rb] ; }

COMP r10, r9 ; |

JIFZ :TOVÁBB ; |

MOVE ra, r8 ; > Ciklus törzs

JUMP :VÉGE ; |

TOVÁBB: ; |

INCR r6 ; |

INCR rb ; }

JUMP :ciklus

VÉGE:

RET

#### Kiválasztás

#### Keresés

#### Másolás

#### Kiválogatás

#### Szétválogatás

#### Metszet

#### Unió

#### Maximum kiválasztás

#### Minimum kiválasztás

## A PopAsmWin assembler szoftver

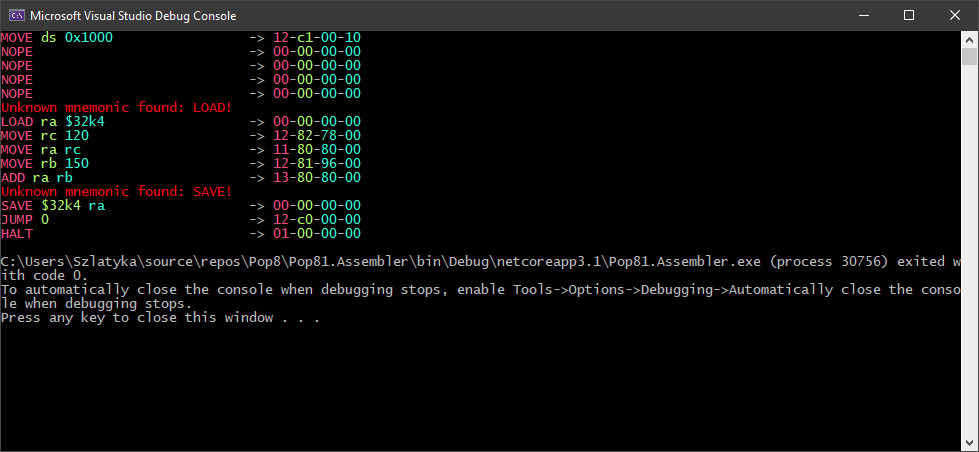
A POP81 processzorhoz elérhető egy platformfüggetlen (Windows, Linux és Mac rendszereken működő) keresztassemlber. Ez a szoftver a popasmwin, ami popasm nyelven írt forráskódból képes futtatható bináris állományt előállítani.

A popasmwin egy parancssori alkalmazás. A parancsori parancs teljes paraméterezhetősége a következő:

popasmwin {input.asm} [--verbose] [--output:{out.bin}]

### --verbose direktíva

A verbose direkítva megadása esetén a popasmwin a lehető legpontosabban, lépésről lépésre a standard outputra írja a fordítási folyamat részleteit. Itt az is megtekinthető, hogy a címkék hol és milyen címre kerülnek beregisztrálásra, illetve, hogy az egyes assembly utasításokból milyen gépi kódú utasíás(ok) álltak elő.



### --output direktíva

Alapértelmezetten a popasmwin assembler egy \*.asm fájlból egy \*.asm.bin kimenetet hoz létre. Értelemszerűen ez nem minden esetben megfelelő, ezért az output direktívávak megadható egy abszolút vagy relatív útvonal, ahova a kimenet lérejöjjön.

## A popvm virtuális gép

A POP81 rendelkezik egy, a processzor gépi kódját végrehajtani képes, platformfüggetlen virtuális géppel. A popasmwin-nel együtt így a teljes fejlesztési folyamat Windows, Linux és Mac rendszereken elvégezhető fizikai processzor nélkül.

**Figyelem!**

A popvm ***nem*** emulátor, a processzor utasításait csak eredményeikben, nem teljes mélységükben emulálja! Tehát alapvetően a létrejött program tesztelésére és nem magának a hardver tesztelésének az eszköze!

### A popvm alapvető tulajdonságai

A popvm egy egyszerű, virtuális és POP81 processzort használó számítógépet valósít meg a következő paraméterekkel:

* 32kB RAM
* közös grafikus RAM, 80\*25 karakteres kimenettel
* saját hardver soros port