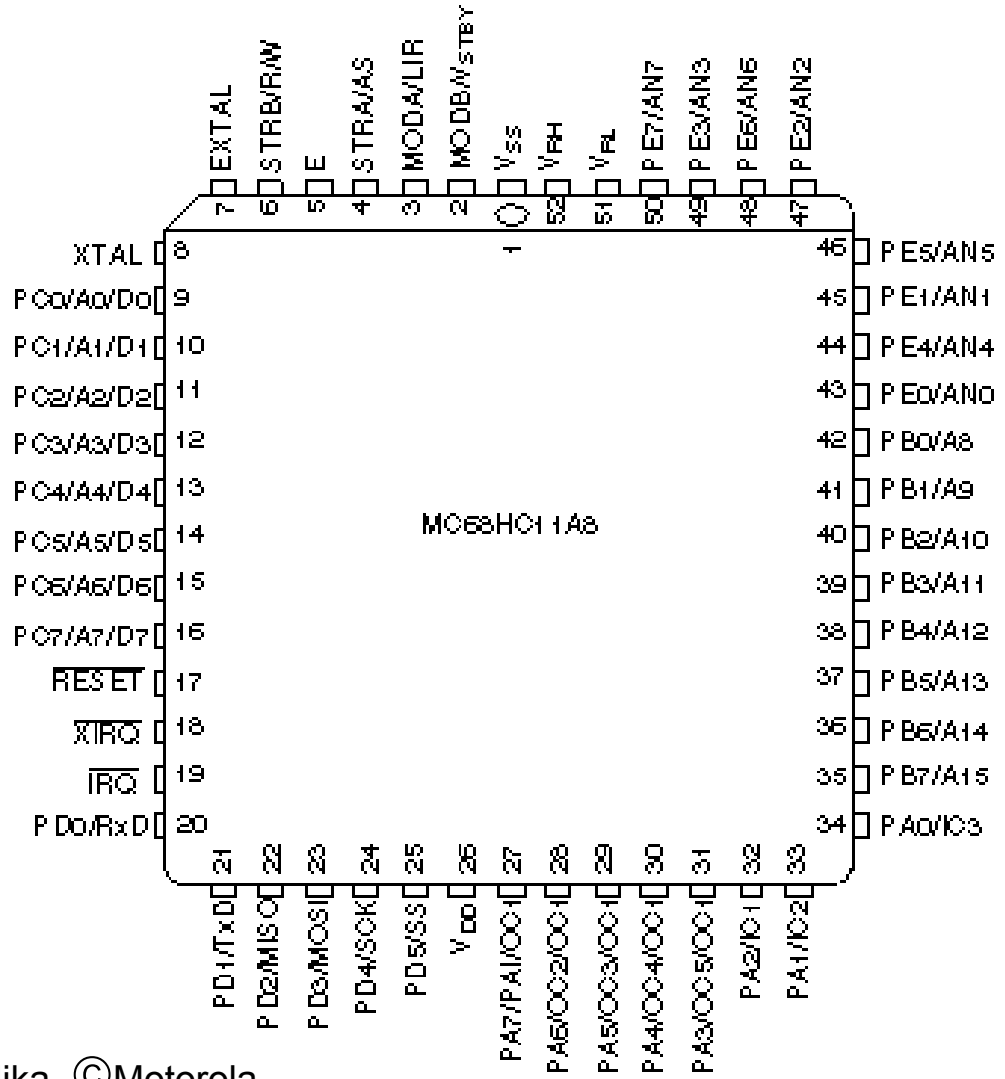
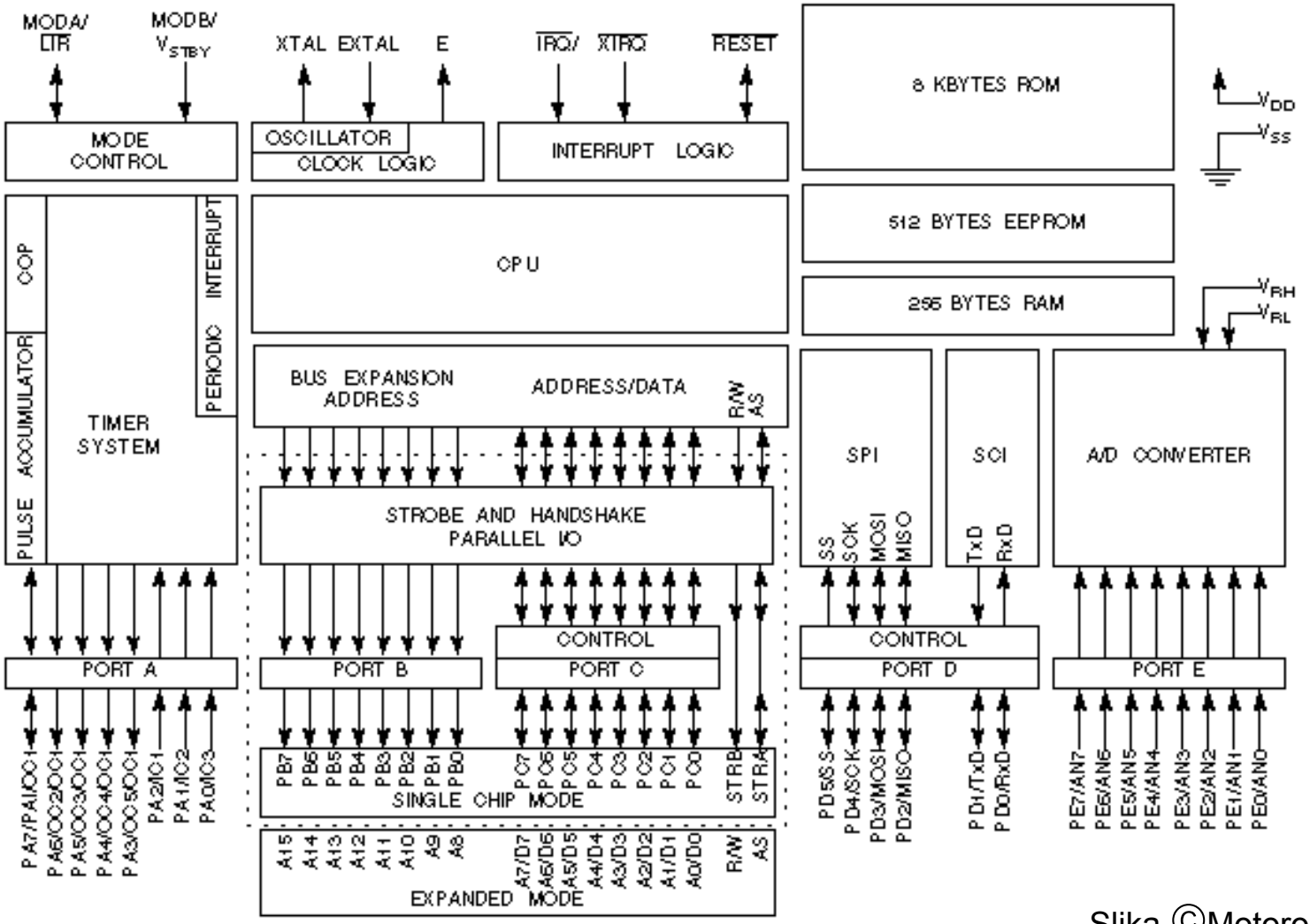
Mikrokrmilnik Motorola MC68HC11



Slika ©Motorola

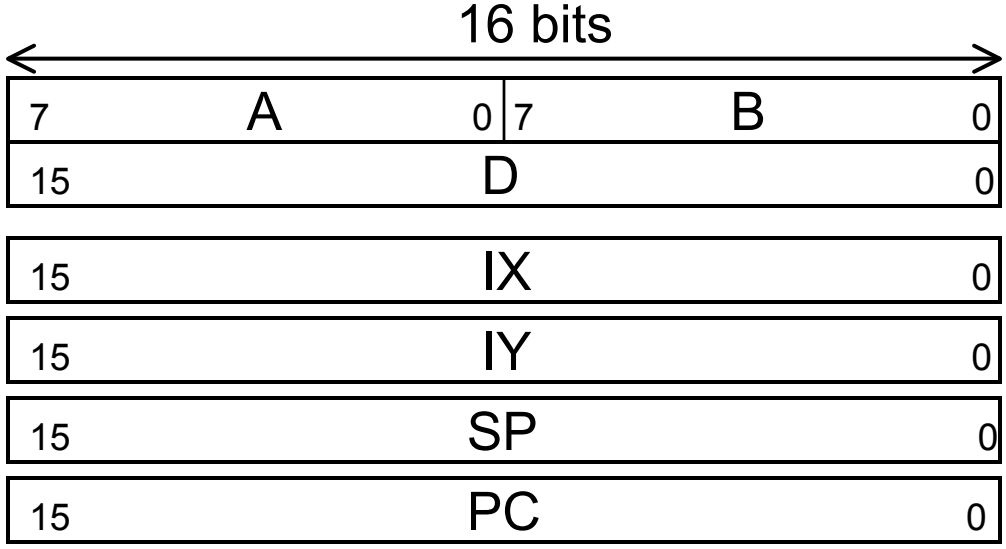
- **Običajno se nahaja v 52-pinskem PLCC ohišju (plastic leaded chip carrier)**
- **Poraba je 15-35 mA pri napajalni napetosti 5-voltov (največja poraba: 165 mW max)**
- **V posebnem režimu delovanja *sleep mode* porabi samo 250 μW , stanje se ohrani**

Shema 68HC11



Slika ©Motorola

Programirni model



Splošni registri:
Vidni kot 8-bitna “A” in “B”
ali kot 16-bitni “D”

Indeksna registra za naslavljanje pomnilnika

Kazalec na sklad

Programski števec

Splošnim registrom (A,B,D) pravimo *akumulatorji*.

Register pogojnih kod

Ukazi 68HC11

- **Ukaz sestoji iz:**
  - Operacijske kode - Določa vrsto ukaza
  - Operandov - 0 do 3 parametri za ukaz
    - *0*: **`ABA`** - add acc. B to acc. A
    - *1*: **`LDAA $34`** - load the value from mem[$34] into acc. A
    - *2*: **`BSET $02, #5`** - set bits 0 and 2 of mem[2]
    - *3*: **`BRCLR $82, 4, LOOP`** - Branch to LOOP if bit 2 of mem[$82] is zero

Format strojnih ukazov

Ukaz (zbirnik)	Strojni jezik
`LDAA #45`	`86 2d`
`LDAB 15,Y`	`18 e6 0f`
`ABA`	`1b`
`STAB $4232`	`f7 42 32`
`SUBB 3,X`	`e0 03`

Vsi ukazi niso enako dolgi

Predpostavimo, da je LDAA na pomnilniškem naslovu $2000. V pomnilniku vidimo naslednje:

$2000	86
$2001	2d
$2002	18
$2003	e6
$2004	0f
$2005	1b

$2006	f7
$2007	42
$2008	32
$2009	e0
$200A	03
$200B	??

Kaj mora narediti procesor?

Ukaz	Strojni jezik
`LDAB 15,Y`	**`18 e6 0f`**

$2002	18
$2003	e6
$2004	0f

- **Najprej mora procesor ukaz prebrati iz pomnilnika.**
  - To traja 3 cikle, ker v enem ciklu lahko prebere le en bajt
  - Read mem[$2002], mem[$2003], mem[$2004]
- **CPE mora izračunati dejanski naslov, ki je vsota odmika 15 in vrednosti indeksnega registra Y**
  - To traja en cikel
  - Na vodilu ni prenosa - slepi cikel
- **CPE mora prebrati pomnilniško lokacijo mem [15 + Y]**
  - To traja še en cikel
  - Read mem[15 + Y]
- **Torej: Ukaz traja pet ciklov**

Programiranje v zbirniku

- **Vsaka vrstica programa v zbirniku je običajno en ukaz**
- **Format je naslednji:**

Labela	Ukaz	Operandi	;komentar
ADDIT	ADDA	#10	; Add 10 to count
	STAA	Total	; store total value

- **Običajno polja ločimo s tabulatorji, dovoljeni so tudi presledki**
  - Če v vrstici ni labele, je še vedno potreben tabulator pred ukazom.

Labele

- **Labela je ime določene vrstice**
  - To ime lahko uporabimo npr. pri skokih na vrstico,...
- **Labele uporabljamo zaradi dveh razlogov**
  - S poimenovanjem pomnilniških lokacij dobimo spremenljivke
  - Za poimenovanje ukazov. Na labele se sklicujemo pri skokih.

```
LOOP        LDAB        #99    ; beginning of loop
            BRA         LOOP
```

Labele se morajo začeti v prvem stolpcu. Lahko se končajo z ':'

Psevdoukazi - ukazi prevajalniku

Predpostavimo, da je spremenljivka *Total* na pomnilniškem naslovu $2500.

* Ta program sešteje vrednosti akumulatorjev A in B

```
                ORG     $2400
BEGIN   2400    LDAA    #0          ;Zero out total
 = 2400 2402    STAA    $2500       ;Zero out total
        2405    ABA                 ;Add to total
Fudge:  2406    SUBA    #1          ;Subtract fudge factor
 = 2406 2408    STAA    $2500       ;Write to total
        240B
                END
```

Program se začenja na lokaciji $2400

Konec programa

Samo labele se začenjajo v stolpcu 1

Deklariranje konstant

Za deklariranje konstant uporabljamo psevdoukaz 'EQU'.

```
PI    EQU    31        ;PI gets the value of PI, sort-of
      LDAA   $2500     ;load diameter into acc. A
      LDAB   #PI       ;load PI into acc. B
      MUL              ;Find circumference
      STD    $2502     ;Store result in memory
```

To je labela - ime konstante

Obvezna uporaba '#'
- Naloži vrednost 31 v ak. B

~~LDAB PI
- Naloži mem[31] v ak. B~~

Rezervacija pomnilnika za spremenljivke

Za deklariranje spremenljivk moramo rezervirati za njih določen prostor v pomnilniku.

```
            ORG    $2000
RADIUS      RMB    2; reserve 2 bytes for radius
```

Labela - ime spremenljivke

Reserve Memory Byte

Potrebujemo 2 bajta

```
            STD    RADIUS        ; store acc. D in memory
```

Assembler bo 'RADIUS' nadomestil z ustrezno lokacijo ($2000) - Neposredno pomnilniško naslavljanje.

Rezervacija prostora v pomnilniku

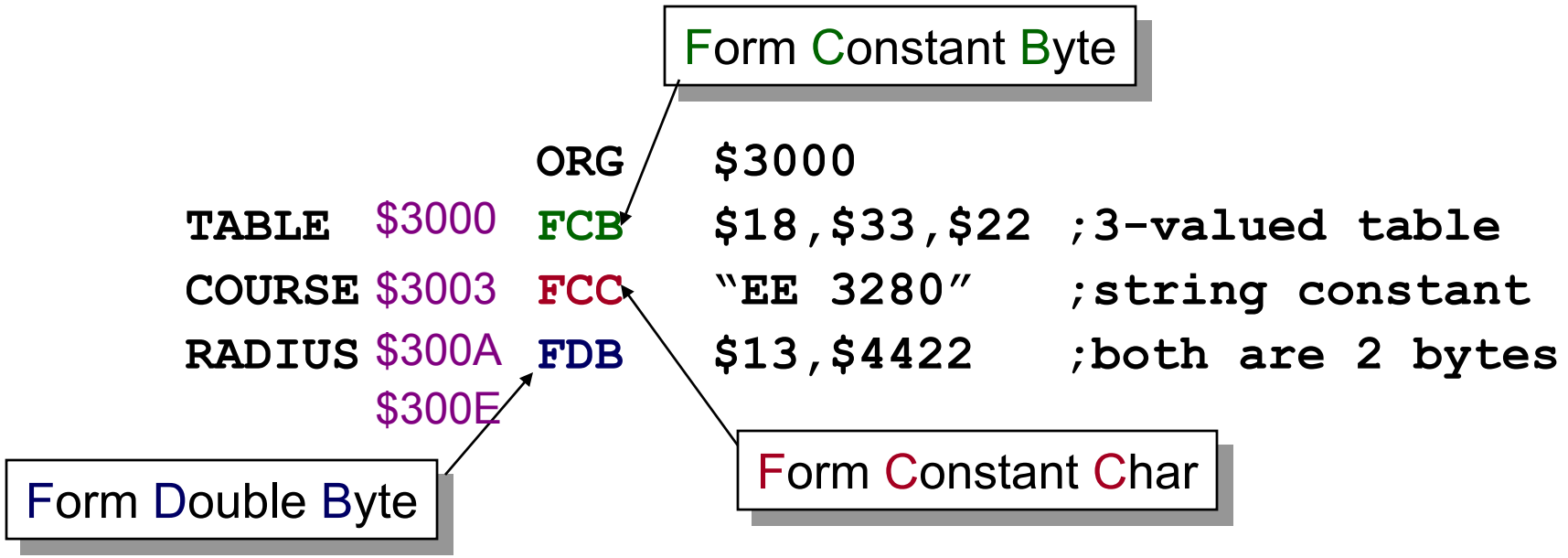
Labele omogočajo boljši pregled nad pomnilnikom

```
                ORG         $2600
BUFFER  $2600   RMB         40      ;reserve 40 bytes
BUFFER2 $2628   RMB         10      ;reserve 10 bytes
                ORG         $2000
BUFFER3 $2000   RMB         20      ;reserve 20 bytes
        $2014
```

- **Labela BUFFER dobi vrednost $2600 (rezervira prostor na naslovih $2600 - $2627)**
- **Labela BUFFER2 dobi vrednost $2628 (rezervira prostor na naslovih $2628 - $2632)**
- **Labela BUFFER3 dobi vrednost $2000 (rezervira prostor na naslovih $2000 - $2013)**

Rezervacija prostora z zač. vrednostmi

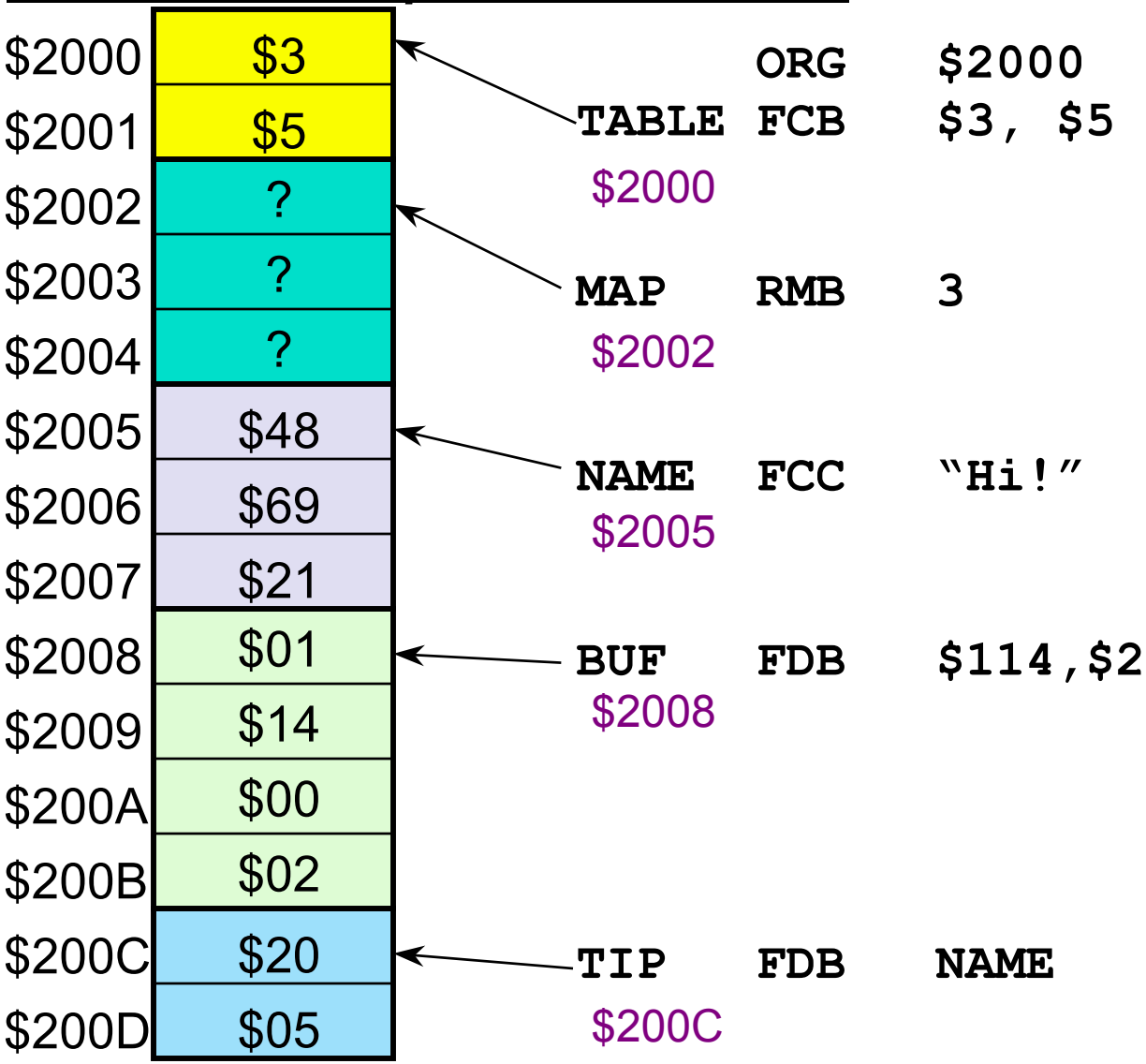
Večkrat želimo, da ima spremenljivka neko začetno vrednost



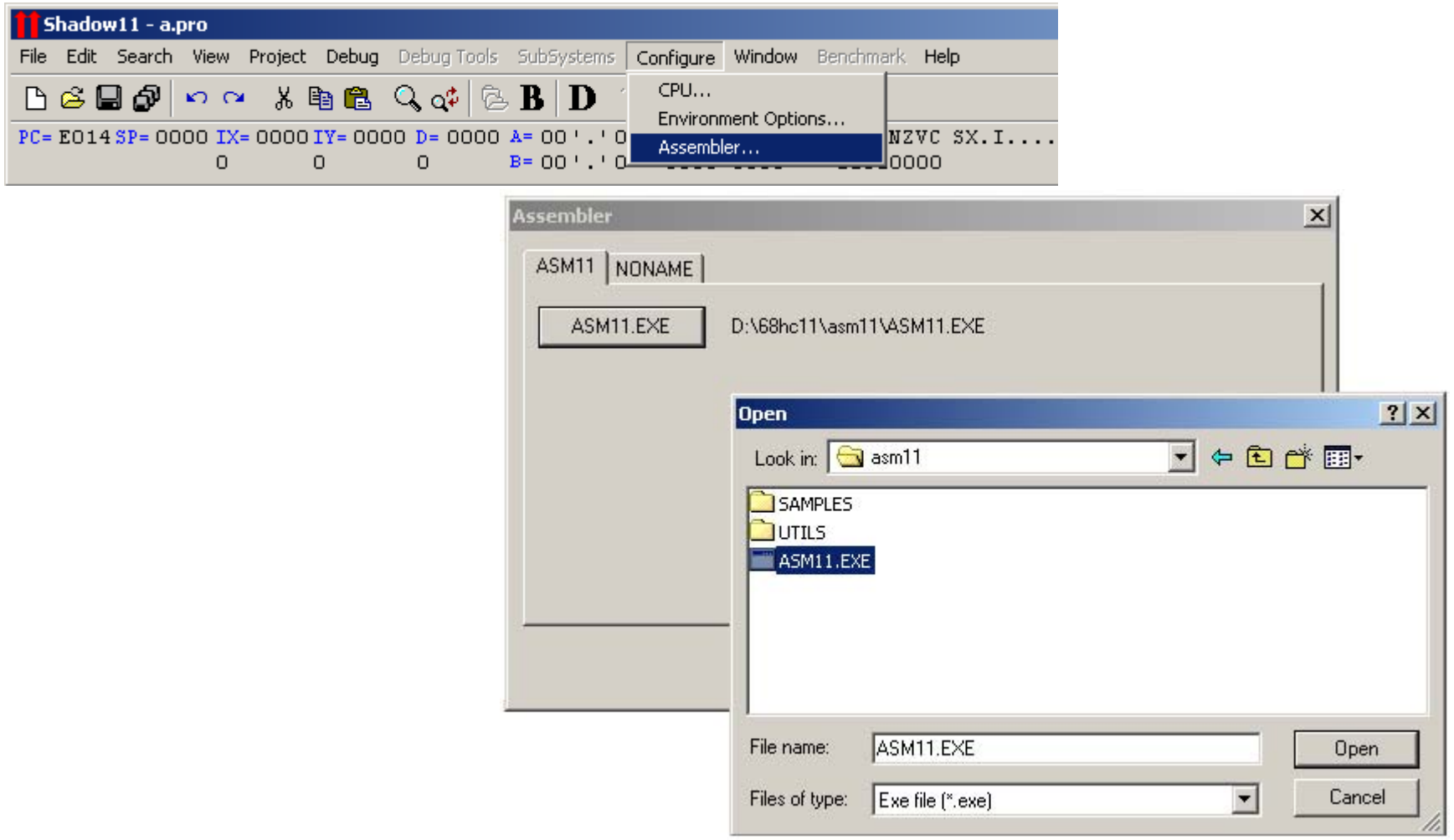
- **Spremenljivke, inicializirane na ta način, lahko kasneje v programu spremenimo**

Pozor: čeprav je podatek "takojšnji", pri psevdoukazih ni potreben znak '#'.

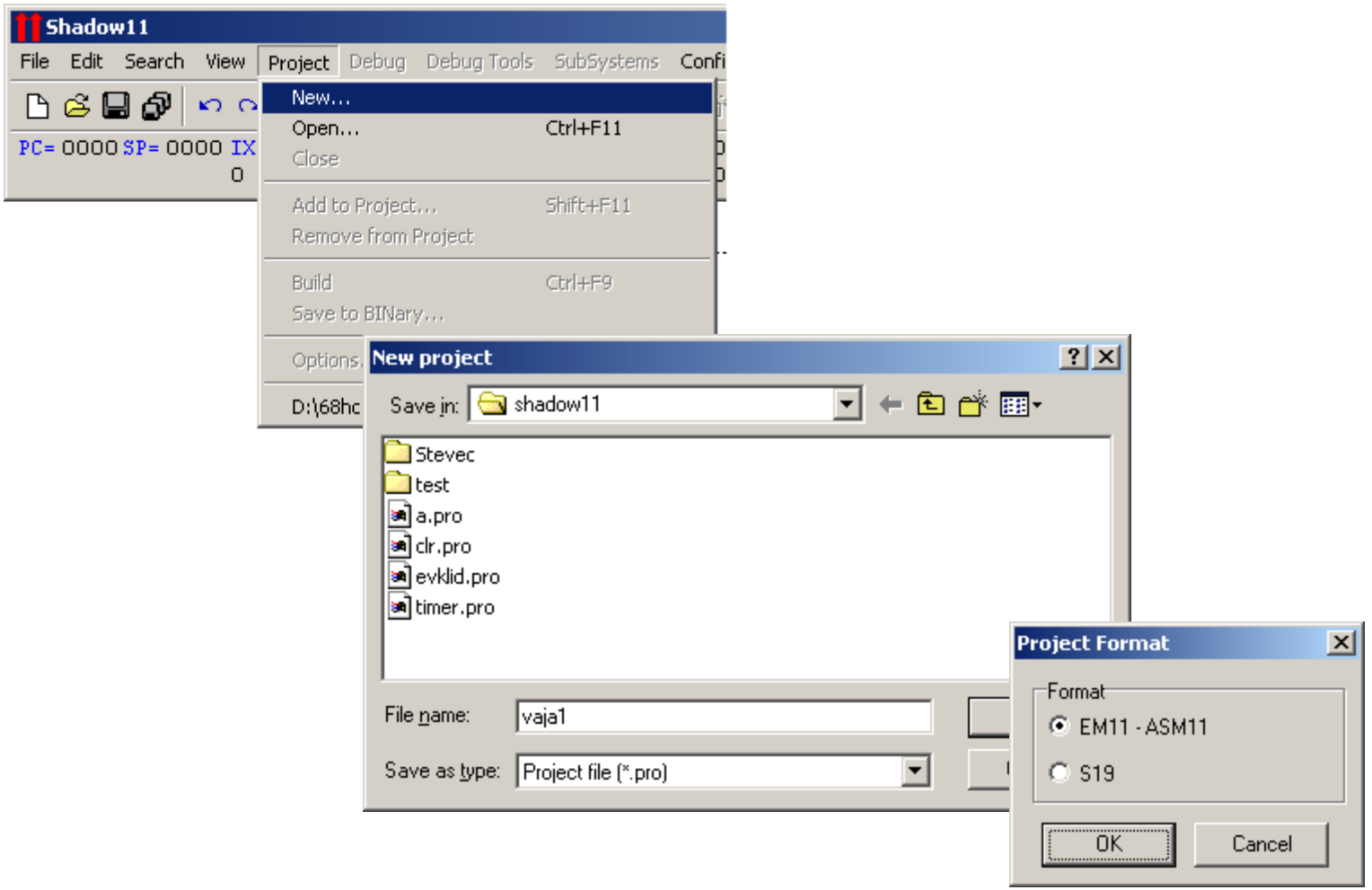
Povzetek - psevdoukazi



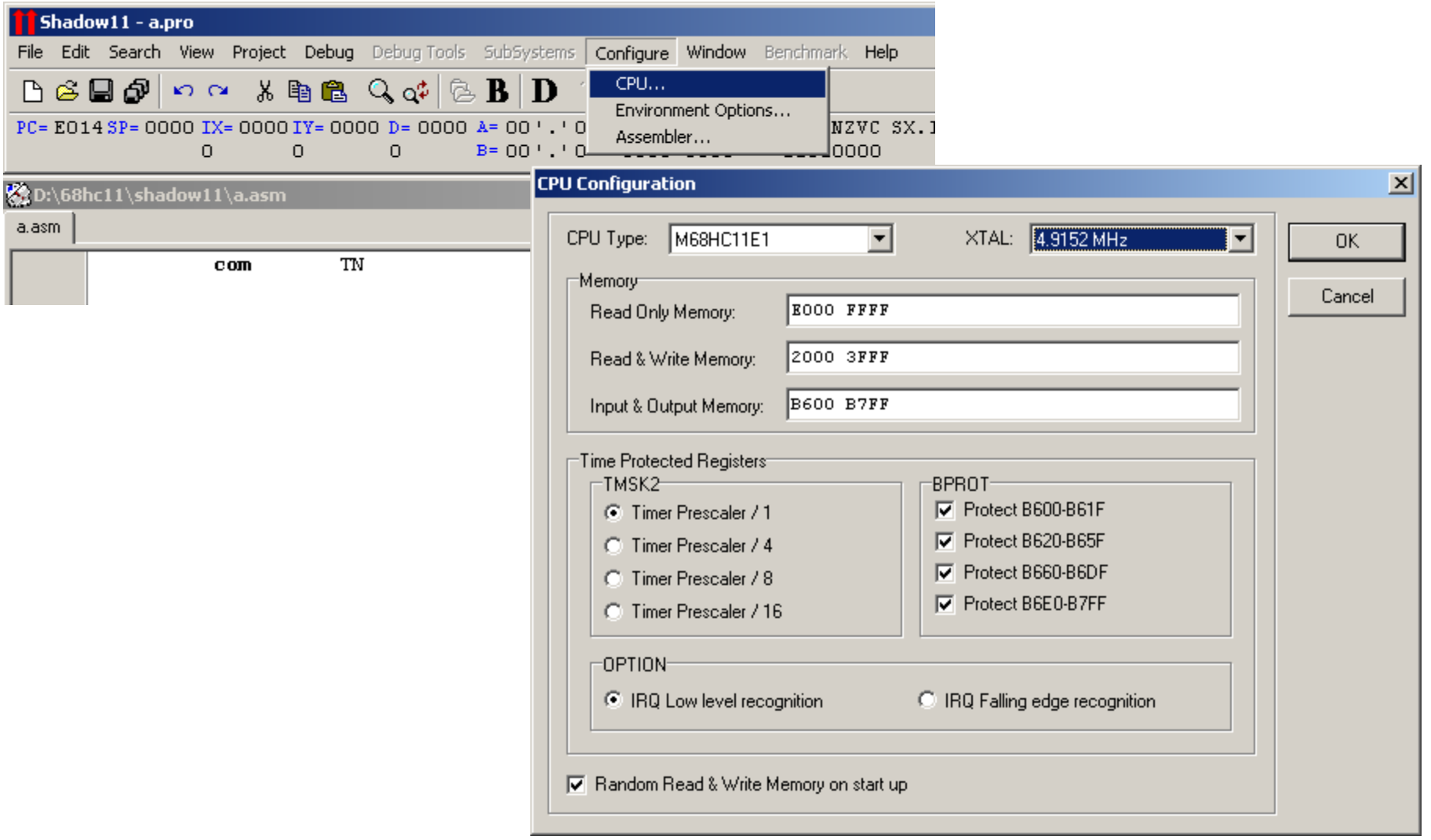
Shadow11 - namestitev



Shadow11 - delo



Shadow11 - nastavitve



VAJA 1: Ponovitev psevdo ukazov

1) Deklariraj konstanto PIA, ki ima vrednost $1800

2) V pomnilniku od naslova $2000 dalje naj se nahaja naslednje

- 16 bitna spremenljivka MAGIC z začetno vrednostjo $1234
- 8 bitna spremenljivka STANJE z začetno vrednostjo 99
- pomnilniški blok BUF1 dolžine 10 bajtov
- ascii znak 'A'
- pomnilniški blok BUF2 dolžine 5 bajtov
- niz 'ARS-VAJE'

3) Od naslova $E000 dalje naj bo glavni program (samo ukaz NOP)

4) Zagotovi, da **reset vektor** kaže na prvi ukaz glavnega programa

Rezultat preveri s pomnilniškim oknom simulatorja *SHADOW11.*

Načini naslavljanja

- **Ukazi potrebujejo operande (izvorne, ponorne)**
  - Operandi so lahko v akumulatorju, pomnilniku ali v samem ukazu
  - Načini naslavljanja določajo kje procesor najde operande
- **Operandi so v programu lahko decimalni (baza-10) ali šestnajstiški (baza-16) ali dvojiški**
  - Običajno so operandi decimalni
- **Števila, pred katerimi je znak '%' so dvojiška**
  - %00010001 (biti v registrih,…)
- **Števila, pred katerimi je znak '$' so šestnajstiška**
  - **$**32, **$**A2, **$**54B3 (pomnilniški naslovi,…)

Neposredno registrsko

- **Najbolj preprost način naslavljanja**
  - Operandi so v akumulatorjih - določeni so z operacijsko kodo ukaza
    - `ABA  ;add accumulator B to accumulator A (result goes in A)`
    - `INCB  ;increment acc. B by one`
    - `LSRD  ;logical shift right acc. D (by one)`

Takojšnje naslavljanje

- **Operand je konstanta in je del ukaza in se v CPE prenese skupaj z njim**

'#' je znak za takojšnji operand
'$' pomeni, da je operand šestnajstiški

Takojšnje, decimalno

- `LDAA #32` ;Load Acc. A with the value 32_{10}
- `LDAB #$C2` ;Load Acc. B with $C2_{16}$ (194_{10})
- `LDY #$123A` ;Load $123A_{16}$ (4666_{10}) into index Y

Takojšnje, šestnajstiško

- Konstante ne smejo biti prevelike
  - največ 255 ($FF) za A in B, 65535 ($FFFF) za ostale registre

Naslavljanje pomnilnika

- **Neposredno naslavljanje, kadar je operand podan s pomnilniškim naslovom**

Naslove običajno podajamo v šestnajst. sistemu

- `LDAA $00` ;loads memory[00_{16}] to acc. A
- `LDAB 21` ;loads memory[21_{10}] into acc. B
- `LDY $1C` ;loads value from mem[$1C_{16}$] into ;upper 8 bits of Y and from ;mem[$1D_{16}$] into lower 8 bits of Y
- Naslov je dolg 1 bajt (lokacije 0-255)
- `LDAA 314` ;not direct mode (address > 255)

- **Razširjeno neposredno naslavljanje omogoča 2-bajtne naslove (<65536)**
  - `LDAA $32A2` ;loads memory[$32A2] into acc. A
  - Razširjeno neposredno naslavljanje potrebuje en cikel več kot neposredno naslavljanje

Indeksno naslavljanje

- **Pri Indeksnem naslavljanju je naslov operanda določen kot vsota indeksnega registra in konstante - *odmika***
  - `LDAA  $20,X  ;loads mem[$20+X] into acc. A`
    - Če ima X vrednost $12, to pomeni branje pomnilniške lokacije $32. Rezultat gre v A.
    - Za indeksno naslavljanje lahko uporabimo samo indeksna registra X in Y
  - Indeksno naslavljanje je uporabno pri strukturah in nizih
    - *Strukture*: Indeksni register kaže na začetek strukture, odmik določa polje v strukturi
    - *Nizi*: Indeksni register kaže na začetek niza, do elementov dostopamo s povečevanjem indeksnega registra

Povzetek načinov naslavljanja

- **Neposredno registrsko**
  - Za operacije tipa register - register
- **Takojšnje**
  - Za konstante (Ki niso v akumulatorjih ali v pomnilniku)
- **Neposredno, Razširjeno neposredno**
  - Za dostop do določene pomnilniške lokacije
- **Indeksno**
  - Za dostop do pomnilniške lokacije, ki je odvisna od izračunane vrednosti
- **Relativno**
  - Uporablja se pri vejitvah, več o tem kasneje