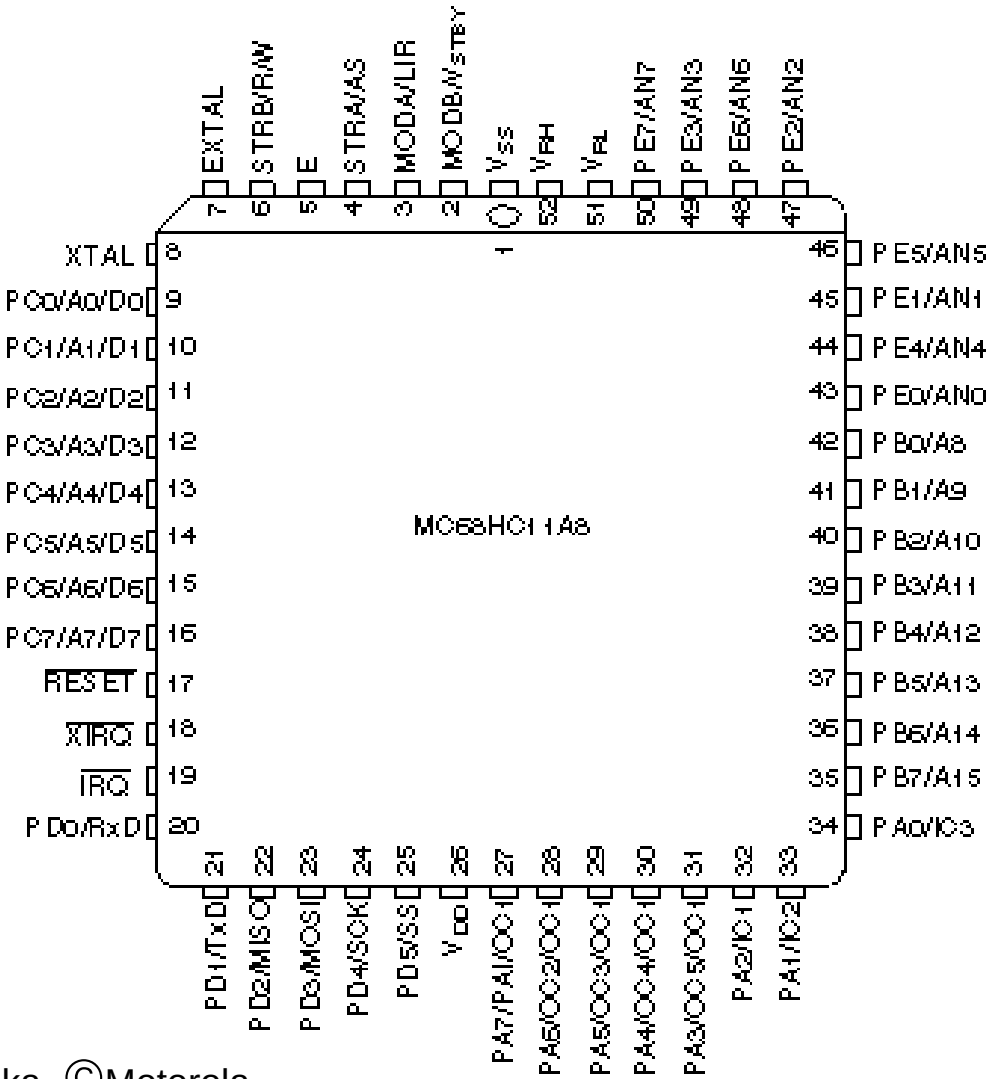
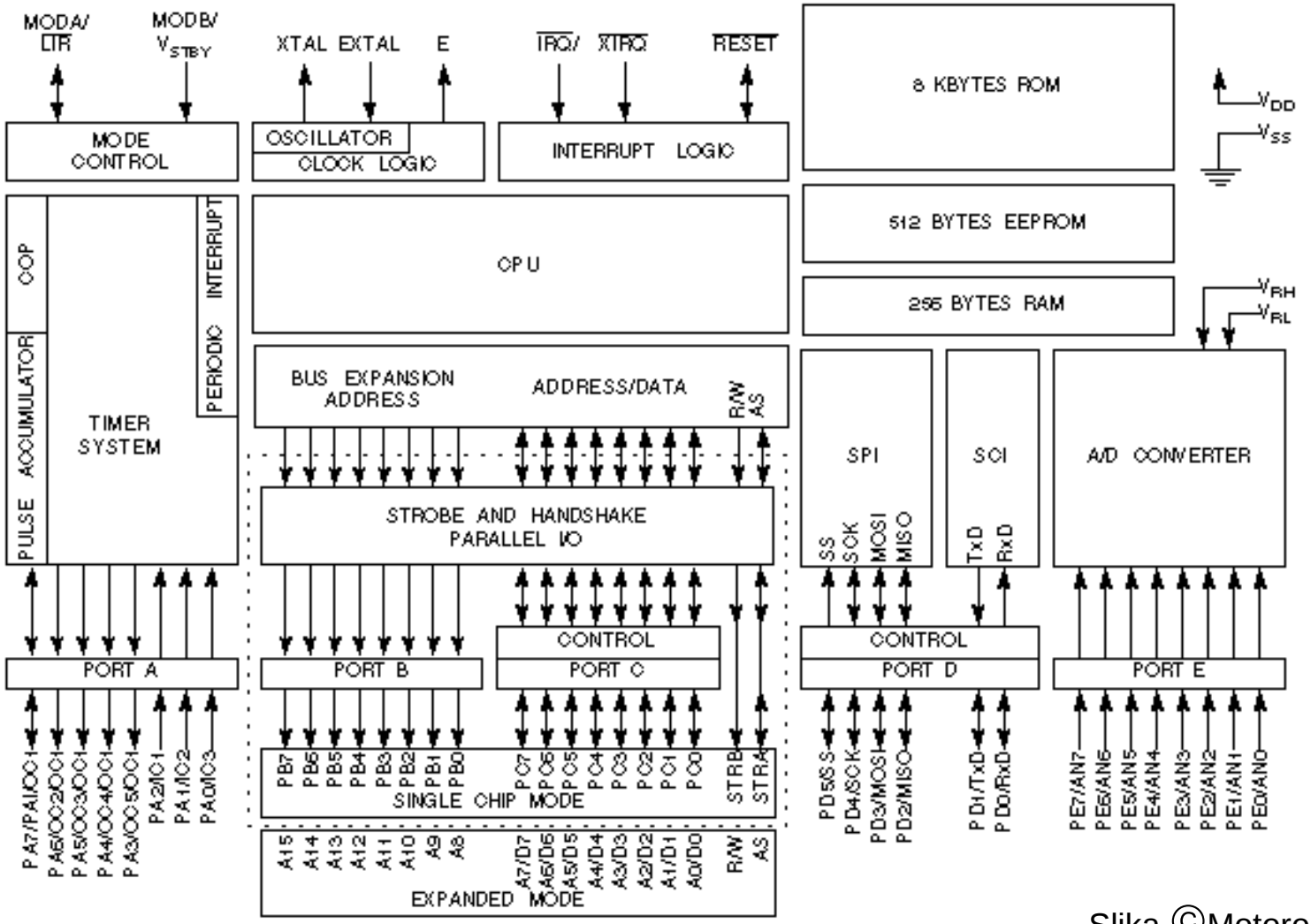
# Mikrokrmilnik Motorola MC68HC11




Slika ©Motorola

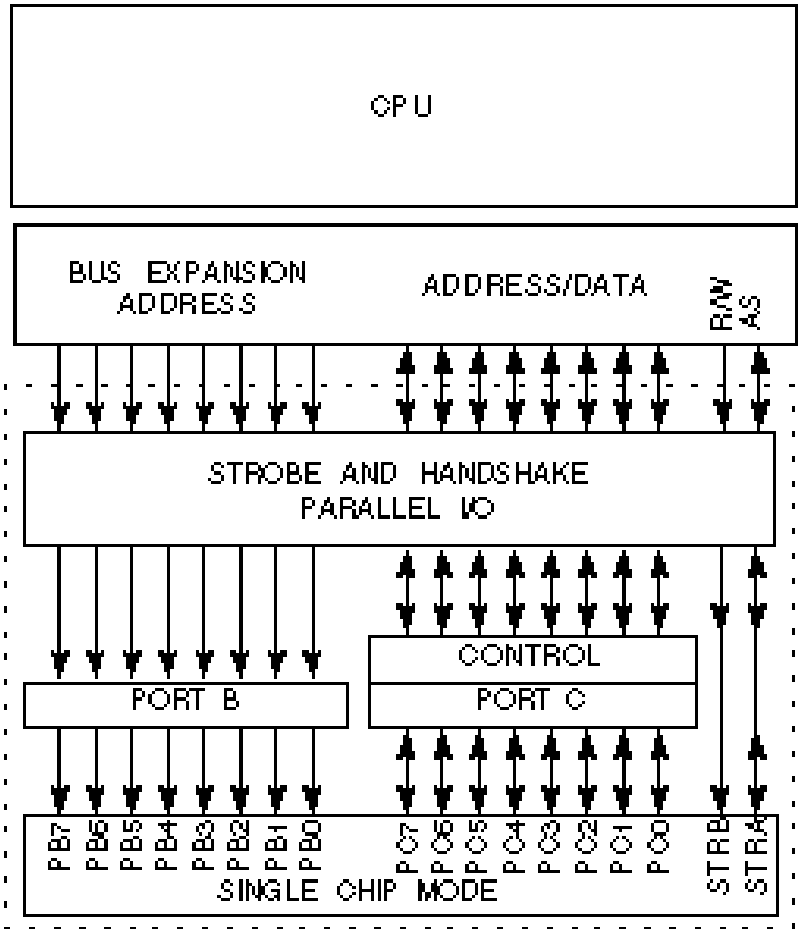
- **Običajno se nahaja v 52-pinskem PLCC ohišju (plastic leaded chip carrier)**
- **Poraba je 15-35 mA pri napajalni napetosti 5-voltov (največja poraba: 165 mW max)**
- **V posebnem režimu delovanja *sleep mode* porabi samo 250 µW , stanje se ohrani**

# Shema 68HC11

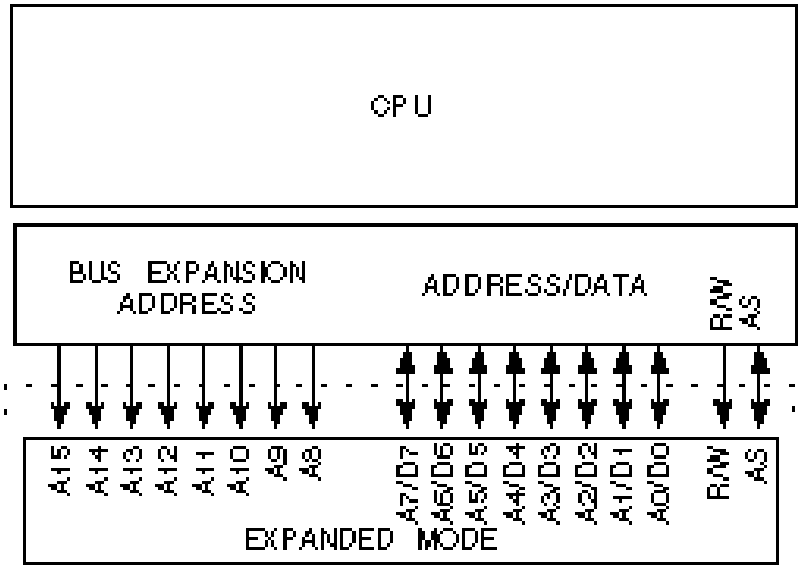



Slika ©Motorola

# Režima Single Chip in Expanded Mode




Dvoje vzporednih vrat




Vmesnik za zunanji pomnilnik

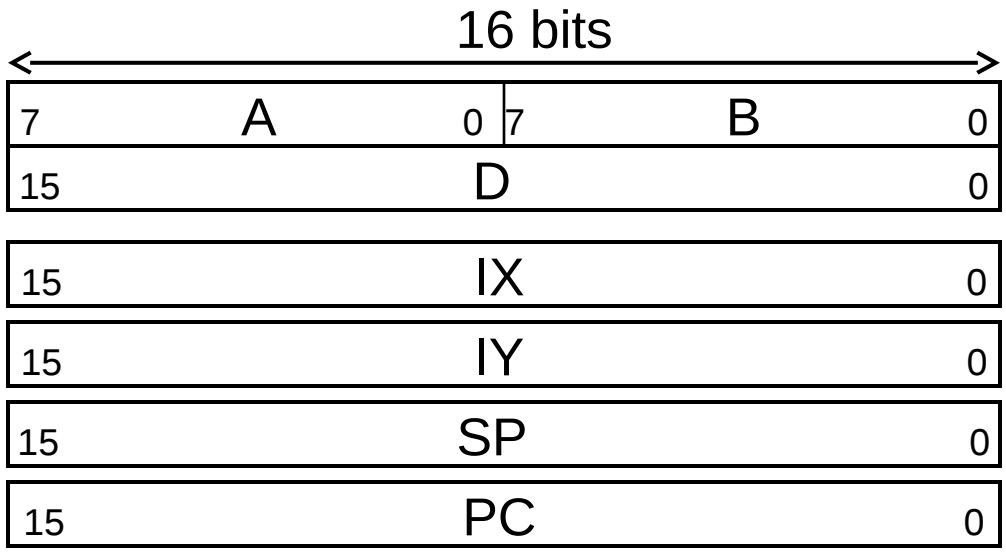
- 16 naslovnih bitov --> $2^{16}$ bajtni naslovni prostor (65536 = 64K bajtov)
- 8 podatkovnih bitov (en bajt)
- Vodili sta multipleksirani

Slika ©Motorola

# *Ko potrebujemo več pomnilnika*

- **Na HC11 lahko priključimo zunanji pomnilnik (ali naprave)**
  - Kadar potrebujemo več kot 256 bajtov RAMa
  - Priključimo lahko tudi ROM ali naprave, ki so namenjene priključevanju na vodilo
- **Zunanji pomnilnik priključimo na naslovno in podatkovno vodilo**

# 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*.

# Ukazi 68HC11

- **Ukazi potrebujejo operande (izvorne, ponorne)**
  - Operandi so lahko v akumulatorju, pomnilniku ali v samem ukazu
  - Načini naslavljanja določajo kje procesor najde operande
- **Operandi so 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,…)

# *Zgradba ukaza*

- **Ukaz je sestavljen iz:**
  - Operacijske kode - Določa vrsto ukaza
  - Operandov - 0 do 3 parametri za ukaz
    - *0*: **`ABA`** – prištej akum. B akumulatorju A
    - *1*: **`LDAA $34`** – naloži vsebino pomnilnika na naslovu $34 v akumulator A
    - *2*: **`BSET $02, #5`** - postavi bita 0 in 2 v pomnilniku na naslovu 2 na 1
    - *3*: **`BRCLR $82, 4, 14`** – skoči za 14 če je bit 2 v pomnilniku na naslovu $82 enak nič

# 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
  - Beri pom[$2002], pom[$2003], pom[$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 pom [15 + Y]**
  - To traja še en cikel
  - Beri pom[15 + Y]
- **Torej: Ukaz traja pet ciklov**

# Programiranje v zbirniku

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

| Oznaka | Ukaz | Operandi | ;komentar |
|---|---|---|---|
| `PRISTEJ` | `ADDA` | `#10` | `; Prištej 10 k vsoti` |
| | `STAA` | `VSOTA` | `; Shrani vsoto` |

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

# Oznake

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

```
LOOP        LDAB        #99    ; začetek zanke
            BRA         LOOP
```

Oznake 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 nekaj vrednosti, rezultat je v ak. A

```
                    ORG     $2400
BEGIN       2400    LDAA    #0          ;Vsota je 0
 = 2400     2402    STAA    $2500       ;Zapiši v pomnilnik
            2405    ABA                 ;Prištej B k A
Fudge:      2406    SUBA    #1          ;Odštej 1
 = 2406     2408    STAA    $2500       ;Shrani rezultat
            240B
                    END
```

Koda se začenja na lokaciji $2400

Konec kode

Samo oznake se začenjajo v stolpcu 1

# Deklariranje konstant

Za deklariranje konstant uporabljamo psevdoukaz ‘EQU’.

```
PI    EQU   31        ;PI dobi vrednost PI, (*10)
      LDAA  $2500     ;naloži premer v akum. A
      LDAB  #PI       ;naloži PI v akum. B
      MUL             ;izračunaj obseg
      STD   $2502     ;shrani rezultat v pomnilnik
```

To je oznaka - ime konstante

Obvezna uporaba ‘#’
- Naloži vrednost 31 v ak. B

~~LDAB PI~~
~~- Naloži pom[31] v ak. B~~

# Rezervacija pomnilnika za spremenljivke

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

```
        ORG    $2000
RADIUS  RMB    2; rezerviraj 2 bajta za premer
```

Oznaka - ime spremenljivke

Reserve Memory Byte

Potrebujemo 2 bajta

```
STD    RADIUS      ; shrani akum. D v pomnilnik
```

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

# Rezervacija prostora v pomnilniku

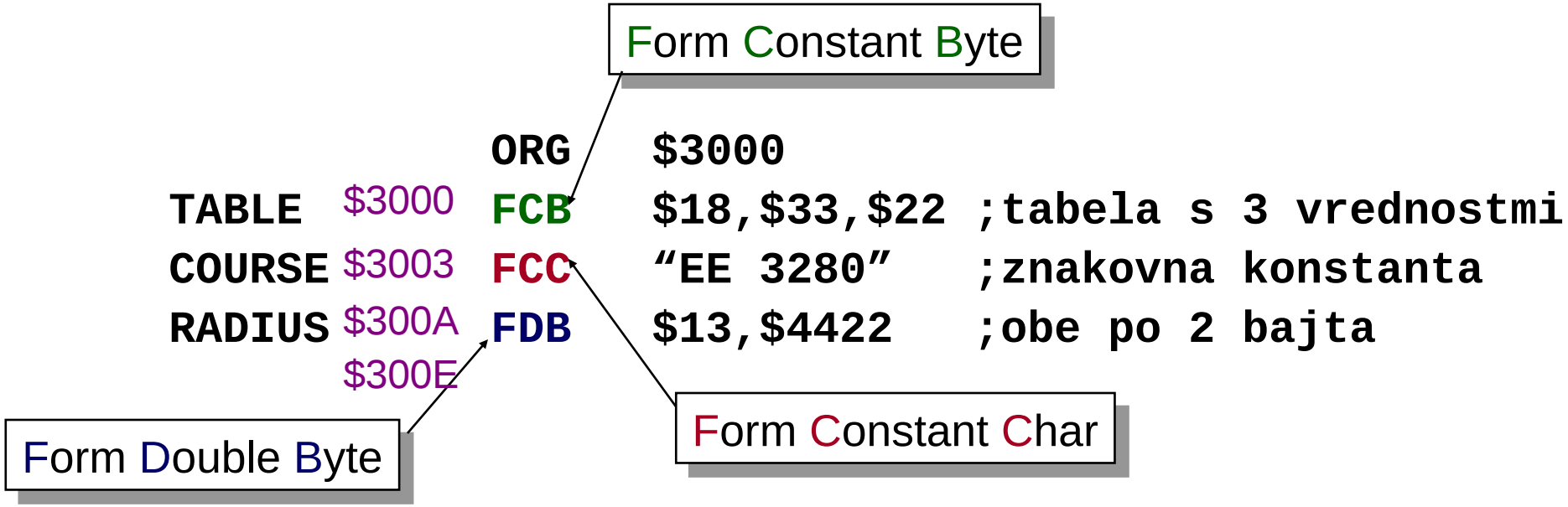
Oznake omogočajo boljši pregled nad pomnilnikom

```
                ORG         $2600
BUFFER  $2600   RMB         40      ;rezerviraj 40 bajtov
BUFFER2 $2628   RMB         10      ;rezerviraj 10 bajtov
                ORG         $2000
BUFFER3 $2000   RMB         20      ;rezerviraj 20 bajtov
        $2014
```

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

# Rezervacija prostora z zač. vrednostmi

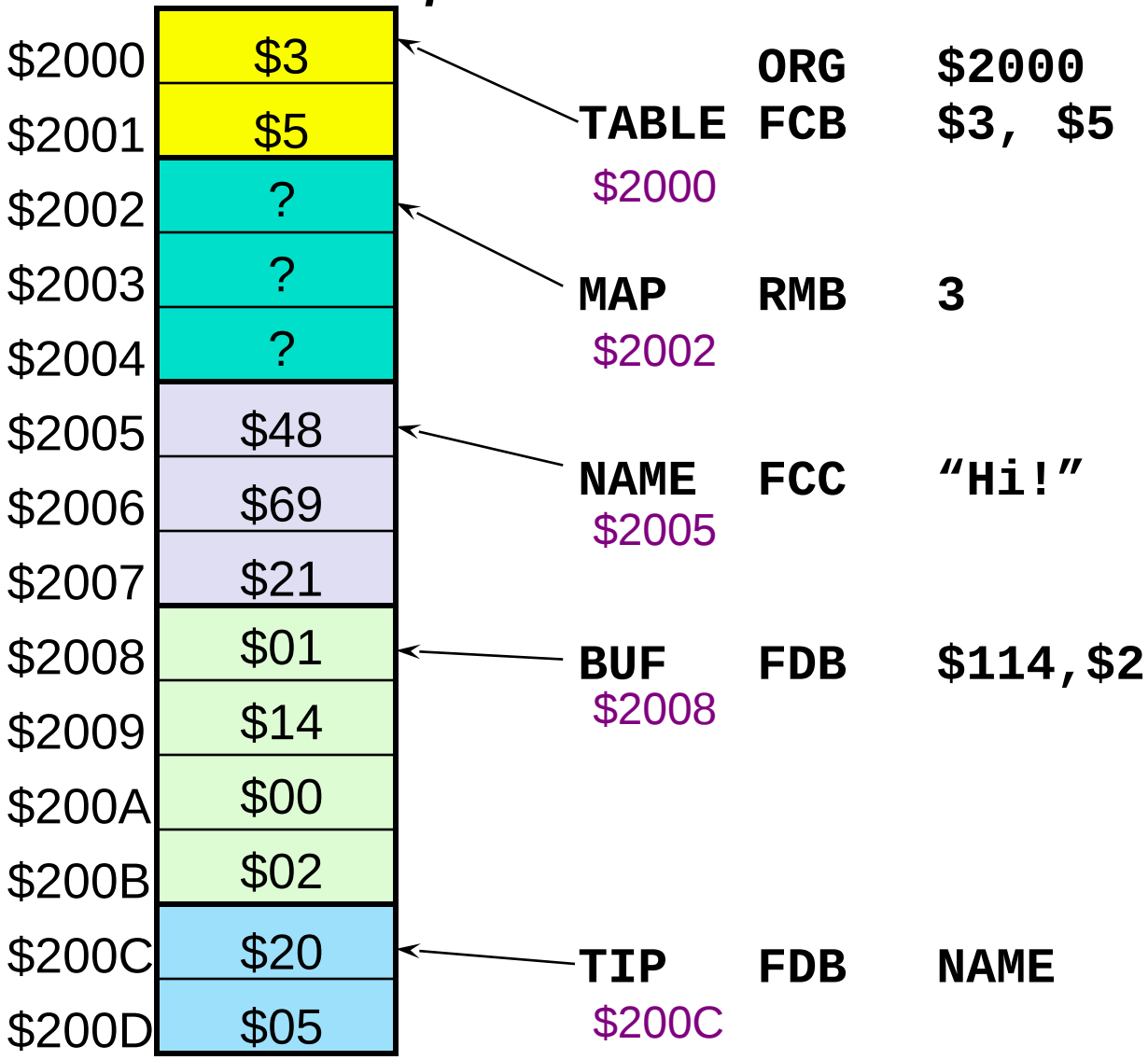
Večkrat želimo, da ima spremenljivka neko začetno vrednost




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

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

# Povzetek – psevdoukazi



# *Ukazi pri 68HC11*

- **Prenos podatkov**
  - LDAA, LDAB, LDD, LDX, LDY, LDS, STAA, STAB, STD, …
  - TAB, TBA, XGDX, XGDY, TSX, TSY, TXS, TYS
  - PSHA, PSHB, PSHX, PSHY, PULA, PULB, PULX, PULY
  - CLR, CLRA, CLRB
  - BSET, BCLR (operacije na posameznih bitih)
  - CLC, CLV, SEC, SEV (operacije na zastavicah)
- **Aritmetične operacije**
  - Seštevanje, odštevanje
    - ADDA, ADDB, ADDD, ABX, ABY, ABA, ADCA, ADCB
    - SUBA, SUBB, SUBD, SBA, SBCA, SBCB
  - Primerjanje (odštevanje brez shranitve rezultata)
    - CMPA, CMPB, CPD, CPX, CPY, TST, TSTA, TSTB
    - BITA, BITB

# *Ukazi pri 68HC11*

- **Aritmetične operacije**
  - Povečevanje/zmanjševanje za 1
    - INC, INCA, INCB, INX, INY, INS
    - DEC, DECA, DECB, DEX, DEY, DES
  - Množenje, deljenje
    - MUL, IDIV, FDIV
- **Logične operacije**
  - LSLA, ASLA, ROLA, LSLB, LSRD,…
  - ANDA, ANDB, ORAA, ORAB, EORA, EORB
  - NEG, NEGA, NEGB, COM, COMA, COMB
- **Vejitve (vplivajo na vrstni red izvajanja ukazov)**
  - BRA, BEQ, BCS, BRCLR, JMP, JSR, RTS, RTI, SWI. …
- **Sistemski ukazi (vplivajo na način delovanja)**
  - SEI, CLI, STOP, TAP

# *Načini naslavljanja pri 68HC11*

Določajo način dostopanja do podatkov

- Neposredno registrsko
- Takojšnje
- Neposredno, razširjeno neposredno
- Indeksno (bazno)
- Relativno

# *Neposredno registrsko*

- **Najbolj preprost način naslavljanja**
  - Operandi so akumulatorji, določeni z operacijsko kodo ukaza

```
ABA     ;prištej akumulator B
        ;akumulatorju A (vsota gre v A)
INCB    ;povečaj akum. B za ena
LSRD    ;logični pomik akum. D v
        ;desno(za ena)
```

# Takojšnje naslavljanje

- **Konstanta je del ukaza in se v pomnilnik prenese skupaj z njim**

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

Takojšnje, decimalno

Takojšnje, šestnajstiško

```
• LDAA #32      ;Naloži vrednost 32(10) v akum. A
• LDAB #$C2     ;Naloži C2(16) (194(10)) v akum. B
• LDY  #$123A   ;Naloži 123A(16) (4666(10)) v reg. Y
```

- 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      ;vsebina pom.[00_16] v akum. A
• LDAB 21       ;vsebina pom.[21_10] v akum. B
• LDY $1C       ;vsebina pomnilnika[1C_16] v
                ;zgornjih 8 bitov registra Y in
                ;vsebina pom.[1D_16] v spodnjih 8
                ;bitov registra Y
```

  - Naslov je dolg 1 bajt (lokacije 0-255)
  - `LDAA 314  ;ne gre neposredno (naslov > 255)`

- **Razširjeno neposredno naslavljanje omogoča 2-bajtne naslove (<65536)**
  - `LDAA $32A2  ;vsebina pom.[$32A2] v akum. A`
  - Razširjeno neposredno naslavljanje potrebuje en cikel več kot neposredno naslavljanje

# *Indeksno oziroma bazno naslavljanje*

- **Pri Indeksnem naslavljanju je naslov določen kot vsota indeksnega registra in konstante - *odmika***
  - **`LDAA  $20,X  ;naloži pom[$20+X] v akum. 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*: Odmik kaže na začetek niza, izračunamo razdaljo do želenega elementa - v indeksni register

# *Načini naslavljanja pri 68HC11 - povzetek*

- **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 (bazno)**
  - Za dostop do pomnilniške lokacije, ki je odvisna od izračunane vrednosti
- **Relativno**
  - Uporablja se pri vejitvah, več o tem kasneje

# Ukaz LDA

***LDA* Load Accumulator**

***Operacija:*** ACCX <-- (M)

Določen akumulator dobi vrednost iz pomnilnika

***Opis:****Naloži vsebino pomnilnika v akumulator.*
*Zastavice se postavijo glede na podatek.*

***Zastavice:***

| S | X | H | I | N | Z | V | C |
|---|---|---|---|---|---|---|---|
| - | - | - | - | ⇳ | ⇳ | 0 | - |

N in Z se postavita ali ne, V je vedno 0 (ni postavjena)

*N R7*
*Se postavi, če je MSB v rezultatu 1; sicer N=0.*

*Z R7' • R6' • R5' • R4' • R3' • R2' • R1' • R0'*
*Postavi se, če je rezultat enak $00, sicer pa ne.*

*V 0*
*Ni postavljena*

***Oblike:*** *LDAA (opr); LDAB (opr)*

Obstajata dve obliki LDA ukaza, ena za vsak akumulator

# LDA

***Načini naslavljanja, Strojni ukazi, Izvajanje po ciklih:***

| | *LDAA #ii* | | | *LDAA $dd* | | | *LDAA $hhll* | | | *LDAA (IND,X)* | | | *LDAA (IND,Y)* | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *Addr* | *Data* | *R/W* | *Addr* | *Data* | *R/W* | *Addr* | *Data* | *R/W* | *Addr* | *Data* | *R/W* | *Addr* | *Data* | *R/W* |
| 1 | OP | 86 | 1 | OP | 96 | 1 | OP | B6 | 1 | OP | A6 | 1 | OP | 18 | 1 |
| 2 | OP+1 | ii | 1 | OP+1 | dd | 1 | OP+1 | hh | 1 | OP+1 | ff | 1 | OP+1 | A6 | 1 |
| 3 | | | | 00dd | (00dd) | 1 | OP+2 | ll | 1 | FFFF | — | 1 | OP+2 | ff | 1 |
| 4 | | | | | | | hhll | (hhll) | 1 | X+ff | (X+ff) | 1 | FFFF | — | 1 |
| 5 | | | | | | | | | | | | | Y+ff | (Y+ff) | 1 |

OP - Prevzem ukaza
- branje ukaza iz pomnilnika

ff - odmik za indeksno naslavljanje

ii - takojšnji operand

dd - pom. naslov za neposredno pom. nas.

FFFF --- - v tem ciklu na vodilu ni prenosa

hhll - pom. naslov za razširjeno neposredno pom. naslavljanje

# LDA

***Načini naslavljanja, Strojni ukazi, Izvajanje po ciklih:***

| *Cikel* | *LDAA (IMM)* | | | *LDAA (DIR)* | | | *LDAA (EXT)* | | | *LDAA (IND,X)* | | | *LDAA (IND,Y)* | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *Addr* | *Data* | *R/W* | *Addr* | *Data* | *R/W* | *Addr* | *Data* | *R/W* | *Addr* | *Data* | *R/W* | *Addr* | *Data* | *R/W* |
| 1 | OP | 86 | 1 | OP | 96 | 1 | OP | B6 | 1 | OP | A6 | 1 | OP | 18 | 1 |
| 2 | OP+1 | ii | 1 | OP+1 | dd | 1 | OP+1 | hh | 1 | OP+1 | ff | 1 | OP+1 | A6 | 1 |
| 3 | | | | 00dd | (00dd) | 1 | OP+2 | ll | 1 | FFFF | — | 1 | OP+2 | ff | 1 |
| 4 | | | | | | | hhll | (hhll) | 1 | X+ff | (X+ff) | 1 | FFFF | — | 1 |
| 5 | | | | | | | | | | | | | Y+ff | (Y+ff) | 1 |
| | 86=1000 0110 | | | 96=1001 0110 | | | B6=1011 0110 | | | A6=1010 0110 | | | A6=1010 0110 | | |

| *Cikel* | *LDAB (IMM)* | | | *LDAB (DIR)* | | | *LDAB (EXT)* | | | *LDAB (IND,X)* | | | *LDAB (IND,Y)* | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *Addr* | *Data* | *R/W* | *Addr* | *Data* | *R/W* | *Addr* | *Data* | *R/W* | *Addr* | *Data* | *R/W* | *Addr* | *Data* | *R/W* |
| 1 | OP | C6 | 1 | OP | D6 | 1 | OP | F6 | 1 | OP | E6 | 1 | OP | 18 | 1 |
| 2 | OP+1 | ii | 1 | OP+1 | dd | 1 | OP+1 | hh | 1 | OP+1 | ff | 1 | OP+1 | E6 | 1 |
| 3 | | | | 00dd | (00dd) | 1 | OP+2 | ll | 1 | FFFF | — | 1 | OP+2 | | 1 |
| 4 | | | | | | | hhll | (hhll) | 1 | X+ff | (X+ff) | 1 | FFFF | — | 1 |
| 5 | | | | | | | | | | | | | Y+ff | (Y+ff) | 1 |
| | C6=1100 0110 | | | D6=1101 0110 | | | F6=1111 0110 | | | E6=1110 0110 | | | E6=1110 0110 | | |

# *Pisanje v pomnilnik*

- **Pisanju v pomnilnik pravimo tudi *shranjevanje* podatkov**
- **Za to uporabljamo skupino ukazov STORE**
  - STAA $12,Y - shrani ak. A v pom[$12+Y]
  - STAB  $3412 - shrani ak. B v pom[$3412]
  - STD $102 - shrani ak. D v pom[$102],[$103]
  - STX $3FF2 - shrani X v pom[$3FF2],[$3FF3]
  - STY 18,X - shrani Y v pom[18+X],[18+X+1]
  - STS $44 - shrani SP v pom[$44],[$45]
- **Ukazi STORE vplivajo na zastavice enako kot ukazi LOAD:**

| S | X | H | I | N | Z | V | C |
|---|---|---|---|---|---|---|---|
| - | - | - | - | ⇳ | ⇳ | 0 | - |

# *Ukazi za seštevanje*

- **Skupina ukazov ADD prišteje vrednost registrov, takojšnjih operandov, ali pomnilniških lokacij k akumulatorju.**
  - ABA - prištej akum. B k ak. A
  - ABX - prištej akum. B k indeksnem reg. X
  - ABY - prištej akum. B k indeksnem reg. Y
  - ADDA #$13 - prištej $13 k akum. A
  - ADDB $64 - prištej pom[$64] k akum. B
  - ADDD 10,Y - prištej pom[10+Y],[10+Y+1] k akum. D
    - Če Y = 30 in pom[40] = $12, pom[41]=$A2, potem se k D prišteje $12A2
- **Zastavica C je uporabna za seštevanje “daljših” števil**

# *Seštevanje s prenosom, Odštevanje*

- **ADC prišteje dve vrednosti k akumulatorju**
  - Podan operand in bit C (prenos)
  - Omogoča seštevanje števil, daljših od 16 bitov
  - ADCA  #72 - prišteje 72 + C bit k ak. A
  - ADCB  $0112 - prišteje pom[$0112] + C k ak. B

- **Odštevanje je podobno seštevanju**
  - SBA, SUBA, SUBB, SUBD, SBCA, SBCB
- **Pri seštevnaju/odštevanju se ustrezno postavijo zastavice H*, N, Z, V, C**

  ***na H vplivajo samo ukazi za seštevanje**

# Seštevanje dolgih števil

- **Kako seštevamo števila, ki so daljša od 8 bitov?**

```
    11
  $22C7
+ $3159
  -----
  $5420
```

Želeni rezultat

```
          ORG    $2100
RESULT    RMB    2
          LDAA   #$C7
          ADDA   #$59    ;prištej nižji bajt
          STAA   RESULT+1
          LDAA   #$22
ADCA      ADDA   #$31    ;prištej višji byte
          STAA   RESULT
          END
```

ADCA - Add with carry to A

```
     1
  $22C7
+ $3159
  -----
  $5320
```

Rezultat

Program deluje le, če ukaza STAA and LDAA ne spremenita zastavice prenosa C! Preveri !!

- Odštevanje je podobno seštevanju, le da namesto prenosa upoštevamo **sposojeni bit (borrow)**. *(zastavica C pri odštevanju predstavlja sposojeni bit)*

# *Ukazi prištej/odštej 1 in zbriši*

- **`INC` - Povečaj za 1**
  - Oblike: INC pom, INCA, INCB, INX, INY, INS
- **`DEC` - Zmanjšaj za 1**
  - Oblike: DEC pom, DECA, DECB, DEX, DEY, DES
- **`CLR` - pobriši (postavi na 0)**
  - Oblike: CLR mem, CLRA, CLRB