Лекция 7

Програмен език Асемблер

За разлика от други програмни езици Асемблер не е просто един език, а по скоро група програмни езици. Всяка процесорна фамилия има собствен език Асемблер.

Програмният език Асемблер дава едно специфично разбиране за работата на компютъра на ниско ниво (близо до хардуерните процеси). В езиците от високо ниво има дистанция между апаратната част на компютъра и програмиста. Това е така, защото програмните езици от високо ниво са проектирани да са по-близо до алгоритмите (логиката) на програмата и да осигуряват лесен интерфейс с програмната среда. Това отдалечава процеса на създаването на програмата от процеситте на тяхното изпълнение. Тази дистанция се покрива от междинни инструменти, каквито са транслаторите, интерпретаторите, свързващите редактори и други.

Задачата на асемблерските езици е да байпасират тези междинни инструменти и да осигурят директен достъп до хардуерните елементи на компютъра. Текстът (source code) на програмите на асемблер са по-големи (с повече оператори), тъй като един оператор на програма от език високо ниво обикновено съдържа няколко асемблерски инструкции. Ето един пример:

```
Assembly Code          C Language Code
---------------        --------------------------------
mov AX,   value        size = value;     // Инициализация на променлива
mov size, AX

mov AX,   sum          sum += x + y + z; // Аритметична операция
add AX,   x
add AX,   y
add AX,   z
mov sum,  AX
```

Защо да се изучава език Асемблер

Първата причина да се работи и изучава Асемблер е, че той дава възможности за разбиране на основните процеси и функционалност на компютърната система. Това помага на програмистите да създават софтуерни продукти с по-логична и смислена структура.

Втората причина е осигуряването на много добър контрол на хардуерните и софтуерните възможности на компютърната система, тъй като в език Асемблер се работи с директни процесорни команди.

Друга причина е, че асемблерските програми са много бързи и имат по-голям обхват от програмите, които могат да се разработят с други програмни езици. Език Асемблер позволява добра оптимизация в програмите, както по отношение на техния размер, така и по отношение на изпълнението им.

Много полезно за прграмистите е да знаят Асемблер, особено в следните случаи:

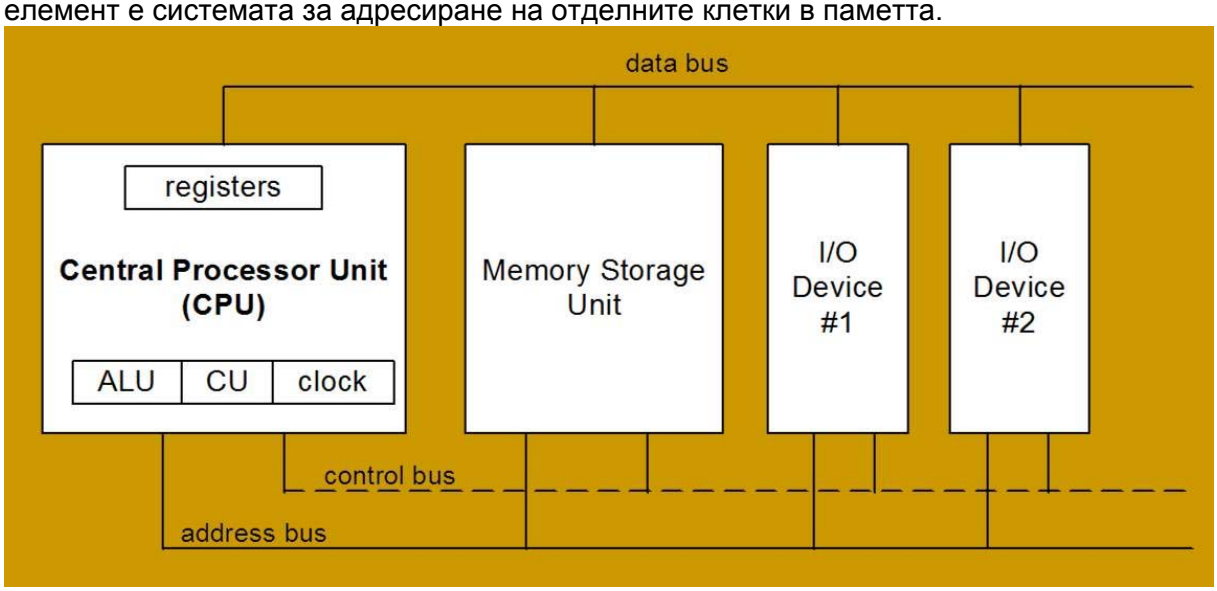
- Когато трябва да се анализират грешки в програмите;
- Когато дадена програма се изпълнява по различен начин от планирания;
- Когато език от високо ниво не подържа някои хардуерни възможности;
- Когато в линейните процедури се изисква някаква порция асемблерски команди.

Разбирането на процесите на компилация и свързване на програми, написани на език от високо ниво изисква познаването на език Асемблер.

Особено важно е използването на език Асемблер в системите, работещи в реално време. При тях от голяма важност е контролът на отделните инструкции, тъй като се изисква прецизна съгласуваност по време на различни операции. В програмните езици

от високо ниво не може да се говори за отделни инструкции и съгласуването на операциите е почти невъзможно.

Как работи микропроцесорът (CPU – Central Process Unit)
Много важно за разбирането на език Асемблер е знанието за начина по който работи компютърът. Най-общо компютърната система се състои от микропроцесор, памет и периферни устройства (входно-изходни устройства) – фиг.1. Периферните устройства се използват за връзка с външния свят, докато паметта е вътрешния свят на микропроцесора. Микропроцесорът е ядрото на системата и е отговорен за изпълнение на командите (инструкции). Микропроцесорът чете команди от участък в паметта, която е заделене за съхранение на програми, транслира тези команди в изпълними стъпки и ги изпълнява.
Разбирането, че процесорът изпълнява операции, а паметта съдържа данни, върху които се извършват тези операции, предполага, че съществуват механизми за четене и запис на данните в паметта. Голяма част от командите на микропроцесора са свързани с механизмите на четене и запис на данни в паметта. При тези механизми важен елемент е системата за адресиране на отделните клетки в паметта.



Фиг. 1 Функционална схема на компютърна система

Основни концепции на език Асемблер

1. CPU Регистри

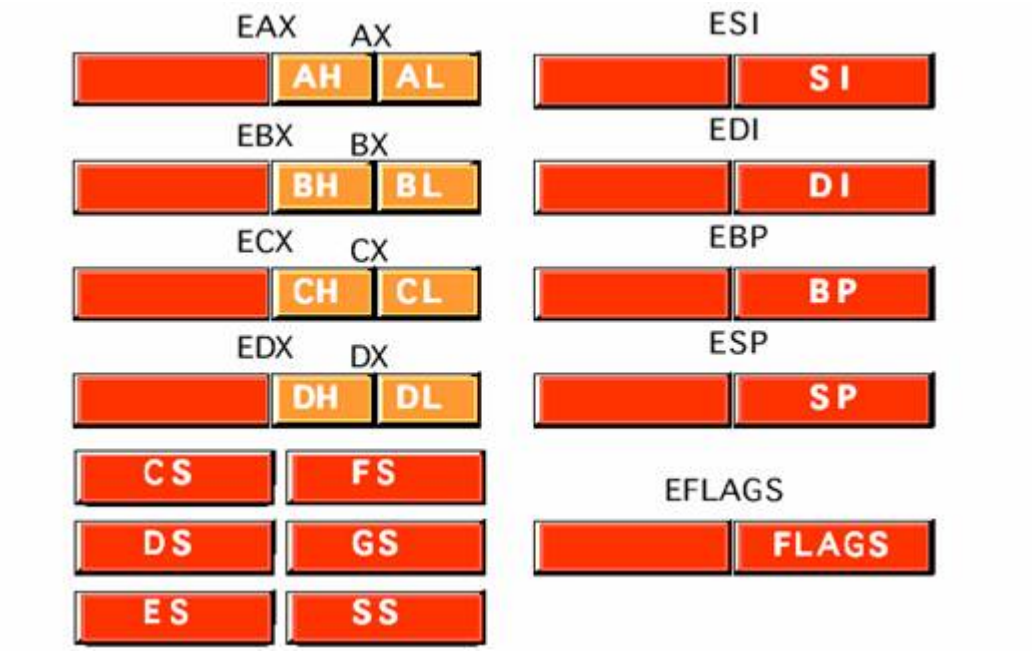
Основна задача на компютърната система е да изпълнява операции (аритметични, логически и други) и за тези операции се изискват операнди . Операндите са данни, върху които се извършват операции. Например при операцията събиране е необходимо да се зададат две числа за да се получи тяхната сума. Задаването на тези числа се извършва под формата на операнди за дадената операция. Може да се зададе точно адресът на едното число чрез адресната шина. Другото число (операнд) не може да се адресира с адресната шина в рамките на една инструкция (в този момент адресната шина е заета с първия операнд). За да се разреши тази проблем се използват временни клетки от паметта, които се намират в ядрото на процесора и се наричат ***регистри***. По този начин единият аргумент може да се зареди в регистъра и да се добави към аргумента, който се намира в паметта. Така двата аргумента са достъпни в рамките на една инструкция.

Регистрите са група от 8, 16 или 32 бита. Група от 8 бита се нарича ‘байт‘, а група от 16 бита е се нарича ‘дума‘.

Регистрите са клетки памет, намиращи се в микропроцесора. Достъпът до тях е много по-бърз от достъпа до клетки от основната памет. Различни инструкции на микропроцесора се използват за прехвърляне на данни в или от регистрите. В тях може да се прехвърлят данни от основната памет или да се записват данни в нея. Информацията в регистрите може да се обработва (манипулира) с помощта на инструкции на микропроцесора. Регистрите работят като временни променливи. Те са най-високото ниво в иерархията на разпределение на паметта в компютърните системи.

В съвременните компютърни системи се използват 32-битови регистри (фиг.2):

- EAX, EDX, ECX, EBX, EBP, EDI, и ESI регистри са 32-битови с общо предназначение и се използват за съхранение на текущи данни и достъп до паметта.
- AX, DX, CX, BX, BP, DI, и SI регистри са 16-битови еквивалентни на горните, те представят първите (младши) 16 бита на 32-битовите регистри.
- AH, BH, CH, и DH регистри представят последните (старши) 8 бита от съответните регистри AX, DX, CX, BX.



Фиг. 2. Регистри в съвременните компютърни системи

- Някои регистри имат специално предназначение
  - ECX за управление на LOOP (цикли) и REPeatable (повтарящи се) инструкции
- Четири сегментни регистри CS, DS, ES, SS
- Двата индексни регистъра ESI (индекс на източника) и EDI (индекс на адреса на изпращане - дестинация) могат да бъдат използвани като:
  - 16-битови или 32-битови регистри;
  - В инструкции за обработка на текстови низове;
  - ESI и EDI могат да бъдат използвани и като регистри с общо предназначение при обработка на данни.
- Двата регистри - указатели ESP (указател за стека) и EBP (указател за базов адрес) могат да бъдат използвани като:

- 16-битови или 32-битови регистри
- Използват се изключително за управление на стека

Таблица 1. Типично използване на регистрите на съвременните компютърни системи:

Регистър	Размер	Типично приложение
EAX	32-bit	Акумулатор за операнди и резултат
EBX	32-bit	Базов адрес за данни в сегмента за данни
ECX	32-bit	Брояч при изпълнение на цикли
EDX	32-bit	Указател за данни и за входно-изходни портове
EBP	32-bit	Указател за базов адрес
ESP	32-bit	Указател на стека – използван при PUSH и POP операции
ESI	32-bit	Индекс на източника – използва се при някои операции с масиви
EDI	32-bit	Индекс на адеса за изпращане (дестинация) - използва се при някои операции с масиви
EIP	32-bit	Указател за инструкции
EFLAGS	32-bit	Флагове за резултата – използва се в операции с условия

Акумулатор
Един от основните регистри е така наречения акумулатор. Обикновено всички аритметични и логически операции се изпълняват с помощта на акумулатора. 32 битовите микропроцесори имат 32 битов акумулатор, който се означава ***EAX***.

Флагов регистър (програмен статус)
Това е специален регистър ***EFLAGS***, който е 8, 16, или 32 битов и за разлика от акумулатора не съхранява определена стойност, а значимост имат отделните битове в него. Всеки бит от флаговия регистър има самостоятелно значение.

Програмен брояч (Указател на инструкции)
Програмите се дефинират като подредена последователност от инструкции. Командите се изпълняват една след друга и всяка команда има определена позиция в общата подредба на инструкциите. В микропроцесорите съществува специален регистър ***EIP***, който се нарича програмен брояч или указател на инструкциите, който осигурява последователното изпълнение на операциите, съгласно алгоритъма.
Това е специален регистър съдържащ адреса на следващата команда. В този регистър не могат да се записват данни и той е извън контрола на потребителите. Той се контролира и използва от микропроцесора.

Индексни регистри (SI и DI)
ESI и ***EDI*** се използват като индекс съответно на източника и дестинацията (адрес на изпращане). В архитектурата на Intel тези регистри се използват при много аритметични и логически операции като регистрите с общо предназначение. Наричат

се ***source*** и ***destination*** поради тяхната функция при използването на клас стрингови инстркции. Тяхното прилагане обаче не е ограничено само до тези операции. Те могат да се използват като регистри с общо предназначение, но за разлика от регистри AX, BX, CX и DX не могат да се използват като 8 битови, а само като 16 битови регистри.

Сегментни регистри (CS, DS, SS, and ES)
Това са регистри за адресиране на сегмента за код, сегмента за данни, сегмента за стека и допълнителен сегментен регистър. Те са специални регистри, използвани в архитектурата на Intel микропроцесорите.

2. **Програма Debug**

За създаване на програма на език Асемблер съществуват две възможности. Първата е да се използва специализирана програмна среда – например ***TASM*** (Turbo Assembler) на Borland, а втората е да се използва debugger (програма ***Debug***) – това е вградена във всеки PC програма, като част от MS-DOS.
Debug може да създава файлове с разширение ***.COM*** и поради характера на този тип програми те не могат да надхвърлят 64 kb, а също така те трябва да се стартират с изместване (offset) или адрес ***0100H*** в специален сегмент от паметта.
Програмата ***Debug*** осигурява команди, които позволяват извършването на някои полезни операции:
a – Асемблира символни инструкции в машинен код
d – Показва съдържанието на област от паметта
e – Въвеждат се данни в паметта, започвайки от определено място (адрес)
g – Изпълнява програма в паметта
n – Име на програма
p – Изпълнява група от свързани инструкции
q – Изход от програмата Debug
r– Показва съдържанието на един или повече регистри
t – Проследяване на изпълнението на една инструкция
u – Преобразуване на машинен код в символни инструкции
w – Запис на програма върху диск.

Възможно е да се визуализира съдържанието на вътрешните регистри на ***CPU*** с програма Debug. Стартирането на програмата става от конзолен прозорец на DOS (Command prompt):
C:/>**Debug** [Enter]
На следващия ред се появява пказалец ‚- тире‘, което е индикатор, че програмата Debug е стартирана и очаква въвеждане на команди. За показване на съдържанието на вътрешните регистри се използва следната команда:
-r [Enter]
Резултатът от изпълнението на тази команда може да изглежда по следния начин:

```
AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0D62 ES=0D62 SS=0D62 CS=0D62 IP=0100 NV EI PL NZ NA PO NC
0D62:0100 2E CS:
0D62:0101 803ED3DF00 CMP BYTE PTR [DFD3],00 CS:DFD3=03
```

Алтернативен начин за показване на съдържанието на регистри е задаване на командата и наименованието на регистъра, чието съдържание да се показва. Например:

```
-rbx
BX 0000
:
```

Тази инструкция показва съдържанието на регистъра ***BX*** и показалеца за команди се променя от „**-**„ на „**:**“. Когато показалецът се промени по този начин, може да се въведе нова стойност и да се промени съдържанието на този регистър.

3. Структура на програма на Асемблер

Основната форма на оператор (команда) на език асемблер има следния вид:

label <space> opcode <space> operands <space> ; comment

Всяко поле от команда на асемблер трябва да бъде разделено с един или повече пробели (<space>). В език Асемблер операторите (редовете) имат три части. Първата част се нарича ‚**етикет**‘ (**label**). Тя се използва за задаване на имена на операторите и въвежда точки в програмата към които може да насочва потока на изпълнение на операциите при командите за преход. Втората част е наименование на командата а третата част съдържа параметри (операнди). Всички части са опционални (не са задължителни). Когато отсъстват и трите части на командата се получава празен оператор (команда). Текстът след символа ‚**;**‘ се нарича коментар и се използва за въвеждане на пояснителен текст в програмите. Използването на коментари е елемент на добрия стил на програмиране.

Примери:

- ***add ah bh***

Тук "**add**" е команда, която трябва да се изпълни (в случая добавяне – addition), а "**ah**" и "**bh**" са параметри (операнди).

- **lab1 mov al 25**

В този пример се използва етикет (**lab1**) и инструкция (**mov**). Тази инструкция се използва за прехвърляне на данни. В случая трябва да се въведе стойност ***25*** в регистър ***al***.

Наименованията на инструкциите в език Асемблер са съставени от две, три или четири букви. Тези инструкции се наричат мнемонически наименования на инструкциите или кодове на операциите, тъй като те пояснявят каква функция трябва да изпълни процесорът.

Понякога инструкциите се използват и във вида:

add al [170]

Квадратните скоби във втория параметър показват, че се използва съдържанието на клетка от паметта с адрес ***170***, а не директна стойност. Този начин на използване на параметрите се нарича директно адресиране.

4. Създаване на програма на Асемблер

Първата стъпка е да се стартира програмата Debug. За да се асемблира програма с помощта на ***Debug*** се използва команда "***a***" (assemble). Когато се използва тази команда като параметър трябва да се зададе адрес от паметта, където се намира програмата на Асемблер. Ако такъв параметър не е зададен се предполага, че адресът от паметта, където е разположена програмата се задава с регистъра ***CS:IP***, обикновено ***0100h***, който е адреса, където се разполагат програми с **.COM** разширение. Този адрес се използва след като Debug създаде изпълнимата програма. Въпреки, че не е задължително да се задава параметър за командата „a“ се препоръчва да се използва адрес за асемблиране за да не се получават грешки с разполагане програмите.

Ето един пример:

a 100 [enter]
mov ax,0002 [enter]
mov bx,0004 [enter]
add ax,bx [enter]
nop [enter] [enter]

В тази програма стойност 0002 се прхвърля в регистър **ax**, стойност 0004 в регистър **bx**, добавя се (**add**) съдържанието на регистър **bx** към съдържанието на регистър **ax** и накрая се използва команда **nop** (no operation), за да завърши програмата.
В действителност записът и изпълнението на програмата се извършва по следния начин:

```
C:\>debug
-a 100
0D62:0100 mov ax,0002
0D62:0103 mov bx,0004
0D62:0106 add ax,bx
0D62:0108 nop
0D62:0109
```

Ако трябва да се проследи изпълнението на тази елементарна програма може да се използва команда **"t" (trace**):

```
-t
AX=0002 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0D62 ES=0D62 SS=0D62 CS=0D62 IP=0103 NV EI PL NZ NA PO NC
0D62:0103 BB0400 MOV BX,0004
```

Вижда се че стойност 0002 е прехвърлена в регистър AX. Следващото активиране на команда "t" (trace) показва измененията в регистрите след втората инструкция от програмата на Асемблер:

```
-t
AX=0002 BX=0004 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0D62 ES=0D62 SS=0D62 CS=0D62 IP=0106 NV EI PL NZ NA PO NC
0D62:0106 01D8 ADD AX,BX
```

Отново изпълнение на команда "t" (trace) визуализира промяната на регситрите след третата инструкция на асемблерската програма:

```
-t
AX=0006 BX=0004 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0D62 ES=0D62 SS=0D62 CS=0D62 IP=0108 NV EI PL NZ NA PE NC
0D62:0108 90 NOP
```

Изход от програмата ***Debug*** се извършва с командата "q" (quit).

5. **Запис и зареждане на програма на Асемблер**

За да е полезно създаването на програми е необходимо тези програми да могат да се съхраняват върху външен носител (диск) и след това да могат да се зареждат за ново изпълнение или за редактиране.
За да се съхрани (запише) дадена програма, която се намира в паметта на компютъра е необходимо:

- Да се определеи дължината на програмата, като от крайния адрес се извади началният адрес на паметта, в която се намира програмата (най-чест в шестнадесетичен вид).
- Да се зададе на програмата наименование и разширение.
- Да се зададе дължината на програмата в CX регистъра.
- Да се използва програма Debug за запис на асемблерската програма върху диска.

Използвайки горния пример на асемблерска програма е необходимо да се направи следното:

- Когато се асембилра програмата тя би изглеждала по следния начин:

```
0C1B:0100 mov ax,0002
0C1B:0103 mov bx,0004
```

0C1B:0106 add ax,bx
0C1B:0108 int 20
0C1B:010A

- За да се получи дължината на програмата се използва команда на Debug "h" - тя показва сумата и разликата (в шестнадесетичен вид) между два адреса въведени като параметри за командата. Съгласно горния пример трябва да се използва командата "h" в следния вид:

-h 10a 100
020a 000a

Първият резултат е сумата на двата адреса, а вторият е разликата между тях.
Командата "***n***" позволява да се зададе име на програмата :

-n test.com

Команда "rcx" се използва за да се промени съдържанието на регистър CX със стойността, която е получена от командата "h", в случая 000a.

-rcx
CX 0000
:000a

Накрая командата "***w***" записва асемблерската програма върху диска, отбелязвайки колко байта са записани.

-w
Writing 000A bytes

За зареждане на записан файл (програма) са необходими две стъпки:

- Да се зададе име на файла, който да се зареди.
- Да се зареди файла с използване на команда "l" (load).

За да се провери дали процесът на зареждане е протекъл правилно се използва команда „u“.

-n test.com
-l
-u 100 109
0C3D:0100 B80200 MOV AX,0002
0C3D:0103 BB0400 MOV BX,0004
0C3D:0106 01D8 ADD AX,BX
0C3D:0108 CD20 INT 20

Командата "u" се използва за проверка дали програмата е заредене в паметта. Това, което прави програмата е да дисасемблира кода, който е зареден в паметта и да го покаже като асемблерски команди. Програма Debug винаги зарежда програмите в паметта на адрес 100H.